

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-  
КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

*Кваліфікаційна наукова праця  
на правах рукопису*

**КОЛОДЮК АНДРІЙ ВАСИЛЬОВИЧ**

УДК 004.75:004.272:004.052.3

**ДИСЕРТАЦІЯ**

**МЕТОД УПОРЯДКОВАНОЇ ПАРАЛЕЛЬНОЇ ДОСТАВКИ ПОДІЙ ДЛЯ  
ПІДВИЩЕННЯ СТАБІЛЬНОСТІ Й МАСШТАБОВАНOSTІ  
МІКРОСЕРВІСНИХ СИСТЕМ НА ОСНОВІ ТЕХНОЛОГІЙ  
АСИНХРОННОГО ОБМІНУ ПОВІДОМЛЕННЯМИ ТА АНАЛІТИЧНОГО  
МОДЕЛЮВАННЯ**

121 «Інженерія програмного забезпечення»

12 «Інформаційні технології»

Подається на здобуття ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

---

**А.В. Колодюк**

(підпис, ініціали та прізвище здобувача)

Науковий керівник

**Аронов Андрій Олексійович**

кандидат технічних наук

## АНОТАЦІЯ

Колодюк А. В. Метод упорядкованої паралельної доставки подій для підвищення стабільності й масштабованості мікросервісних систем на основі технологій асинхронного обміну повідомленнями та аналітичного моделювання. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії з галузі знань 12 «Інформаційні технології» за спеціальністю 121 «Інженерія програмного забезпечення» – Державний університет інформаційно-комунікаційних технологій Міністерства освіти і науки України, Київ, 2026.

У дисертаційній роботі вирішено актуальну науково-прикладну задачу підвищення коректності, продуктивності та безперервності доставки подій у мікросервісних системах, що побудовані за подієво-орієнтованою архітектурою з асинхронним обміном повідомленнями. Широке впровадження мікросервісів та брокерів повідомлень із семантикою доставки at-least-once у поточному стані супроводжується низкою суперечливих вимог таких, як необхідністю одночасно забезпечувати впорядкованість обробки подій у межах логічного контексту, високу пропускну здатність за рахунок паралелізму, ідемпотентність обробки повторних доставок та безперервність функціонування за умов часткових збоїв. Відомі рішення, а саме партиціонування Apache Kafka за ключем, паралельне споживання RabbitMQ, патерни Saga та Transactional Outbox, розв'язують ці вимоги лише частково та не забезпечують їх у сукупності. Збільшення паралелізму руйнує впорядкованість, а механізми надійного відновлення суттєво уповільнюють реакцію системи на збій. Зазначена суперечність зумовлює актуальність дослідження.

Об'єктом дослідження є процеси доставки та обробки подій у розподілених мікросервісних системах. Предметом дослідження є методи, моделі та архітектурні засоби упорядкованої паралельної доставки подій із гарантуванням коректності, ідемпотентності та відмовостійкості.

**Метою** роботи є підвищення ефективності мікросервісних систем доставки подій шляхом розроблення методу упорядкованого паралелізму, аналітичної моделі продуктивності та архітектури безперервної доставки. Для

досягнення мети сформульовано й розв'язано такі **задачі**: проаналізовано наявні підходи та семантики доставки повідомлень; формалізовано подієво-орієнтовану модель системи на основі подання події у вигляді кортежу з ключем впорядкування, порядковим номером, часовою міткою та корисним навантаженням; розроблено метод прикладного впорядкування подій за паралельної обробки; побудовано аналітичну модель пропускну здатності; розроблено архітектуру резервування транспортних каналів та керування розподіленими процесами; виконано експериментальну перевірку отриманих результатів засобами імітаційного моделювання.

Методологічну основу дослідження становлять теорія розподілених систем, теорія масового обслуговування, методи паралельних обчислень (закони Амдала та Густафсона, формула Літла), апарат теорії ймовірностей і математичної статистики, а також методи імітаційного моделювання. Достовірність результатів підтверджено зіставленням аналітичних оцінок із даними багаторазових симуляцій із фіксованими seed-значеннями, реалістичними розподілами параметрів та непараметричними статистичними критеріями (критерій Манна–Уїтні, оцінка величини ефекту за Коеном).

#### **Наукова новизна одержаних результатів полягає в наступному:**

1. Набув подальшого розвитку метод упорядкованого паралелізму в межах логічного ключа у мікросервісних системах доставки подій на основі прикладного впорядкування поверх брокера повідомлень із використанням маркування подій ідентифікаторами послідовності та механізму відновлення пропущених станів, що дозволяє забезпечити коректне послідовне опрацювання взаємопов'язаних подій за одночасного паралельного виконання незалежних потоків обробки та підвищити продуктивність системи.

2. Вперше розроблено аналітичну модель впливу паралелізму на продуктивність і латентність системи впорядкованої доставки подій у режимі гарантованої доставки з можливими повторами на основі комплексного врахування ідемпотентної обробки, повторної доставки повідомлень, буферизації пропущених станів і використання резервного каналу доставки, що дозволяє формалізовано оцінювати накладні витрати, визначати оптимальний

рівень паралелізму та прогнозувати поведінку системи за умов зміни навантаження і часткових відмов.

3. Удосконалено архітектуру забезпечення безперервності доставки подій у мікросервісних системах на основі інтеграції методу упорядкованого паралелізму, аналітичної моделі оцінювання продуктивності та латентності, основного брокера повідомлень, резервного каналу доставки та механізму узгодженого виконання розподілених кроків, що дозволяє підвищити безперервність виконання процесів, скоротити час відновлення після відмов і забезпечити коректність послідовності виконання операцій в умовах часткових збоїв інфраструктури.

### **Практичне значення наукових результатів полягає у наступному:**

1. Набув подальшого розвитку метод упорядкованого паралелізму в межах логічного ключа у мікросервісних системах доставки подій, який на основі прикладного впорядкування поверх брокера повідомлень, використання ідентифікаторів послідовності, внутрішніх черг за ключами та механізму відновлення пропущених станів забезпечує детерміновану впорядкованість обробки подій у всіх досліджених сценаріях. Практичне застосування запропонованого методу дозволяє зберігати коректну послідовність виконання взаємопов'язаних операцій за одночасного паралельного опрацювання незалежних сутностей та забезпечує прискорення обробки подій до 2,42 раза порівняно з базовими підходами в умовах підвищеного навантаження і часткових збоїв.

2. Вперше розроблена аналітична модель впливу паралелізму на продуктивність і латентність системи впорядкованої доставки подій, побудована на основі формалізації процесів, реалізованих методом упорядкованого паралелізму, дозволяє враховувати вплив повторної доставки повідомлень, ідемпотентної обробки, буферизації пропущених станів та використання резервного каналу доставки на характеристики функціонування системи. Практичне використання моделі забезпечує оцінювання накладних витрат, визначення раціонального рівня паралелізму та прогнозування продуктивності системи з відносною похибкою не більше 5,5 %, що дозволяє використовувати її

під час проєктування, масштабування та модернізації високонавантажених мікросервісних систем.

3. Удосконалена архітектура забезпечення безперервності доставки подій, побудована на основі інтеграції методу упорядкованого паралелізму, аналітичної моделі оцінювання продуктивності та латентності, основного брокера повідомлень, резервного каналу доставки та механізму узгодженого виконання розподілених кроків, забезпечує скорочення середнього часу відновлення після збою на 76 % (223,5 мс проти 930,2 мс у базовій архітектурі), підвищення частки успішно доставлених подій до 99,79% порівняно з 95,3 % у традиційних рішеннях, а також підтримання частки успішно завершених Saga-транзакцій на рівні 98,9 % проти 31,8 % за умов нестабільної роботи брокера повідомлень. При цьому додаткові накладні витрати за латентністю у штатному режимі функціонування не перевищують 2,4 %, що дозволяє забезпечити високий рівень безперервності, надійності та доступності мікросервісних систем без істотного зниження їх продуктивності.

Дисертація складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі обґрунтовано теоретичні засади побудови мікросервісних систем та організації доставки подій і виконано постановку задачі. У другому розділі вперше розроблено метод упорядкованої паралельної доставки подій. У третьому розділі побудовано та верифіковано аналітичну модель продуктивності. У четвертому розділі удосконалено архітектуру забезпечення безперервності й проведено експериментальне дослідження, що підтвердило ефективність запропонованих рішень.

*Ключові слова:* мікросервісна архітектура, подієво-орієнтована система, доставка подій, упорядкованість, паралелізм, ідемпотентність, семантика at-least-once, відмовостійкість, безперервність, аналітична модель, пропускна здатність, оптимальний рівень паралелізму, Saga, брокер повідомлень, інформаційна система, інформаційні технології.

## SUMMARY

Kolodiuk A. V. Method of Ordered Parallel Event Delivery for Improving the Stability and Scalability of Microservice Systems Based on Asynchronous Messaging Technologies and Analytical Modeling. – Qualifying scientific work as a manuscript.

Dissertation for the degree of Doctor of Philosophy in the field of knowledge 12 “Information Technologies” in specialty 121 “Software Engineering” – State University of Information and Communication Technologies of the Ministry of Education and Science of Ukraine, Kyiv, 2026.

The dissertation solves a relevant scientific and applied problem of improving the correctness, performance, and continuity of event delivery in microservice systems built according to an event-driven architecture with asynchronous message exchange. The widespread adoption of microservices and message brokers with at-least-once delivery semantics is currently accompanied by a number of conflicting requirements, such as the need to simultaneously ensure ordered event processing within a logical context, high throughput through parallelism, idempotent processing of repeated deliveries, and continuity of operation under conditions of partial failures. Existing solutions, namely Apache Kafka key-based partitioning, RabbitMQ parallel consumption, Saga patterns, and Transactional Outbox, address these requirements only partially and do not provide them collectively. Increasing parallelism breaks ordering, while reliable recovery mechanisms significantly slow down system response to failures. This contradiction determines the relevance of the research.

The object of research is the processes of event delivery and processing in distributed microservice systems. The subject of research is methods, models, and architectural means of ordered parallel event delivery with guarantees of correctness, idempotency, and fault tolerance.

The aim of the work is to improve the efficiency of microservice event delivery systems through the development of an ordered parallelism method, an analytical performance model, and a event delivery continuity architecture. To achieve this goal, the following tasks were formulated and solved: existing approaches and message delivery semantics were analyzed; an event-driven system model was formalized based on representing an event as a tuple consisting of an ordering key, sequence number,

timestamp, and payload; a method of application-level event ordering under parallel processing was developed; an analytical throughput model was constructed; an architecture for transport channel redundancy and distributed process management was developed; and the obtained results were experimentally validated using simulation modeling.

The methodological basis of the research consists of distributed systems theory, queueing theory, parallel computing methods (Amdahl's law, Gustafson's law, and Little's formula), probability theory and mathematical statistics, as well as simulation modeling methods. The validity of the results was confirmed by comparing analytical estimates with data obtained from repeated simulations using fixed seed values, realistic parameter distributions, and nonparametric statistical tests (Mann–Whitney test and Cohen's effect size estimation).

The scientific novelty of the obtained results is as follows:

1. The method of ordered parallelism within a logical key in microservice event delivery systems has been further developed based on application-level ordering over a message broker using event labeling with sequence identifiers and a missing-state recovery mechanism, which makes it possible to ensure correct sequential processing of related events while simultaneously executing independent processing flows in parallel and improving system performance.

2. An analytical model of the influence of parallelism on the performance and latency of an ordered event delivery system operating under at-least-once delivery semantics has been developed for the first time based on the integrated consideration of idempotent processing, repeated message delivery, buffering of missing states, and the use of a backup delivery channel, which makes it possible to formally evaluate overhead costs, determine the optimal level of parallelism, and predict system behavior under varying workloads and partial failures.

3. The architecture for ensuring continuous event delivery in microservice systems has been improved based on the integration of the ordered parallelism method, the analytical model for evaluating performance and latency, the primary message broker, a backup delivery channel, and a coordinated execution mechanism for distributed steps, which makes it possible to increase process continuity, reduce

recovery time after failures, and ensure the correctness of operation sequencing under conditions of partial infrastructure failures.

The practical significance of the scientific results is as follows:

1. The method of ordered parallelism within a logical key in microservice event delivery systems has been further developed. Based on application-level ordering over a message broker, the use of sequence identifiers, internal key-based queues, and a missing-state recovery mechanism, it ensures deterministic event processing ordering in all investigated scenarios. Practical application of the proposed method makes it possible to preserve the correct sequence of related operations while simultaneously processing independent entities in parallel and provides up to a 2.42-fold increase in event processing speed compared to baseline approaches under increased load and partial failures.

2. For the first time, an analytical model of the influence of parallelism on the performance and latency of an ordered event delivery system, built upon the formalization of processes implemented by the ordered parallelism method, makes it possible to account for the impact of repeated message delivery, idempotent processing, missing-state buffering, and backup delivery channels on system performance characteristics. Practical use of the model enables overhead estimation, determination of a rational level of parallelism, and performance prediction with a relative error not exceeding 5.5%, making it suitable for the design, scaling, and modernization of high-load microservice systems.

3. The improved architecture for continuous event delivery, built upon the integration of the ordered parallelism method, the analytical model for evaluating performance and latency, the primary message broker, a backup delivery channel, and a coordinated execution mechanism for distributed steps, provides a 76% reduction in average recovery time after failure (223.5 ms versus 930.2 ms in the baseline architecture), increases the share of successfully delivered events to 99.79% compared to 95.3% in conventional solutions, and maintains the share of successfully completed Saga transactions at 98.9% versus 31.8% under unstable broker operation. At the same time, additional latency overhead in normal operating conditions does not exceed 2.4%,



making it possible to ensure a high level of continuity, reliability, and availability of microservice systems without significant performance degradation.

The dissertation consists of an introduction, four chapters, conclusions, a list of references, and appendices. The first chapter substantiates the theoretical foundations of microservice systems and event delivery organization and formulates the research problem. The second chapter develops the method of ordered parallel event delivery. The third chapter constructs and validates the analytical performance model. The fourth chapter improves the continuity assurance architecture and presents an experimental study confirming the effectiveness of the proposed solutions.

**Keywords:** microservice architecture, event-driven system, event delivery, ordering, parallelism, idempotency, at-least-once semantics, fault tolerance, continuity, analytical model, throughput, optimal parallelism level, Saga, message broker, information system, information technologies.

### **Список опублікованих праць за темою дисертації**

*Наукові праці, у яких опубліковані основні наукові результати дисертації:*

1. A. KOLODIUK, O.VOLOSCHUK. Per-order ordered parallelism method in microservice event delivery systems. Зв'язок, 2025, №6 (178), с. 3-12. DOI:10.31673/2412-9070.2025.061213A.
2. Колодюк А.В., Аронов А.О. Аналітична модель впливу паралелізму на продуктивність системи впорядкованої доставки подій з AT-LEAST-ONCE семантикою. Сучасний захист інформації, 2026, №1
3. Дзюба В. В., Колодюк А. В., Олейніков І. А., Бугайов Д. М. Метод упорядкованого паралелізму per-замовлення у мікросервісних системах доставки подій. Зв'язок. 2025, №3 (175), с.101-107. DOI: 10.31673/2412-9070.2025.029944
4. Andrii Kolodiuk, Vadym Chytulian, Ivan Oleinikov, Viktoriia Zhebka, Valeriia Balatska Formation of a conceptual model for cyber-physical monitoring of critical infrastructure environmental objects. CPITS-II 2025: Cybersecurity

Providing in Information and Telecommunication Systems II 2025, p.330-341,  
<https://ceur-ws.org/Vol-4145/paper23.pdf> (Scopus)

5. Колодюк А.В. Метод упорядкованої паралельної доставки подій у мікросервісних системах із використанням RabbitMQ, резервних HTTP-каналів та патерну Saga для підвищення живучості та зменшення MTTR. Зв'язок. 2026, №3
6. Майборода М. В., Бажан Т. О., Жебка С. В., Колодюк А. В. Методи метричного та неметричного багатовимірного шкалювання для довільної матриці даних в мові R. Телекомунікаційні та інформаційні технології. 2024, №3 (84), с. 94 -101. DOI: 10.31673/2412-4338.2024.039411
7. Жебка С. В., Власенко В. О., Аронов А. О., Колодюк А. В. Вирішення дилеми вибору алгоритму консенсусу у розподілених системах. Телекомунікаційні та інформаційні технології. 2024, №1 (82) , с. 130–136. DOI: 10.31673/2412-4338.2024.019904

*Наукові праці, які засвідчують апробацію матеріалів дисертації:*

1. Колодюк А.В. Партнерство між технологічними компаніями та університетами: роль колаборацій у впровадженні новітніх розробок. *II Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу»*. Збірник тез. – К.: ДУІКТ, 2024. С. 371-372
2. Колодюк А.В. Оцінка ефективності мікрсервісної архітектури в умовах високих навантажень: досвід та перспективи для корпоративних веб-додатків. *II Міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії»*. Збірник тез. – К.: ДУІКТ, 2024. С. 21-23.
3. Колодюк А.В. Розробка чат-боту для автоматизації клієнтської підтримки у сфері малого та середнього бізнесу. *Всеукраїнська науково-технічна*

*конференція «Виклики та рішення в програмній інженерії»*. Збірник тез. – К.: ДУІКТ, 2024, с. 209-211.

4. Колодюк А.В. Формалізація впливу архітектурних рішень мікросервісних систем на споживчі характеристики корпоративних веб-додатків. *Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. Збірник тез. – К.: ДУІКТ, 2025, с. 97-99.
5. Колодюк А.В. Новітні парадигми в програмній інженерії. *II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії»*. Збірник тез. – К.: ДУІКТ, 2025, с.69-71

## ЗМІСТ

ВСТУП.....	15
РОЗДІЛ 1. ЗАГАЛЬНІ ПІДХОДИ ДО ПОБУДОВИ МІКРОСЕРВІСНИХ СИСТЕМ ТА ОРГАНІЗАЦІЇ ДОСТАВКИ ПОДІЙ.....	22
1.1. Теоретичні основи мікросервісної архітектури та подієво-орієнтованих систем .....	22
1.2. Семантики доставки повідомлень та їх вплив на коректність обробки подій .....	29
1.3. Методи забезпечення впорядкованості подій у розподілених системах ..	39
1.4. Механізми забезпечення узгодженості та ідемпотентності.....	51
1.5. Підходи до підвищення відмовостійкості та безперервності функціонування .....	59
1.6. Аналіз обмежень існуючих рішень та постановка задачі дослідження....	71
Висновки до розділу 1.....	76
РОЗДІЛ 2. РОЗРОБКА МЕТОДУ УПОРЯДКОВАНОЇ ПАРАЛЕЛЬНОЇ ДОСТАВКИ ПОДІЙ У МІКРОСЕРВІСНИХ СИСТЕМАХ.....	78
2.1. Формалізація подієво-орієнтованої моделі системи.....	78
2.2. Побудова методу упорядкованого паралелізму в умовах асинхронної доставки.....	94
2.3. Реалізація механізмів прикладного рівня впорядкування подій .....	103
2.4. Алгоритм обробки подій із забезпеченням послідовності та паралелізму .....	111
2.5. Механізми відновлення пропущених станів та забезпечення ідемпотентності .....	119
2.6. Дослідження властивостей коректності та живучості методу.....	123
Висновки до розділу 2.....	132

РОЗДІЛ 3. АНАЛІЗ ТА АНАЛІТИЧНЕ МОДЕЛЮВАННЯ СИСТЕМ УПОРЯДКОВАНОЇ ДОСТАВКИ ПОДІЙ.....	135
3.1. Огляд сучасних підходів до аналізу продуктивності подієво-орієнтованих систем .....	135
3.2. Аналіз впливу паралелізму на латентність та пропускну здатність .....	142
3.3. Формалізація параметрів системи та потоків подій .....	150
3.4. Побудова аналітичної моделі системи доставки подій .....	156
3.5. Урахування повторної доставки, ідемпотентності та буферизації.....	163
3.6. Оцінка накладних витрат та визначення оптимального рівня паралелізму .....	169
3.7. Дослідження умов стабільності та масштабованості системи.....	176
Висновки до розділу 3.....	183
РОЗДІЛ 4. УДОСКОНАЛЕННЯ АРХІТЕКТУРИ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ ДОСТАВКИ ПОДІЙ .....	187
4.1. Розробка архітектури забезпечення безперервності доставки подій.....	187
4.2. Інтеграція механізмів резервування каналів та керування розподіленими процесами.....	195
4.3. Забезпечення живучості та зменшення часу відновлення системи .....	201
4.4. Методика експериментального дослідження .....	212
4.5. Оцінка продуктивності, латентності та масштабованості системи .....	215
4.6. Аналіз коректності обробки подій та впорядкованості.....	220
4.7. Порівняння з існуючими підходами та узагальнення результатів .....	226
Висновки до розділу 4.....	232
ВИСНОВКИ.....	234
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	237
Додаток А. Лістинг програмного коду реалізації методу упорядкованої паралельної доставки подій.....	253

Додаток Б. Лістинг програмного коду реалізації аналітичної моделі пропускної здатності системи упорядкованої доставки подій.....	255
Додаток В. Лістинг програмного коду реалізації архітектури безперервної доставки подій .....	256
Додаток Г. Акти впровадження .....	257

## ВСТУП

*Актуальність теми.* Сучасний розвиток цифрових платформ, хмарних обчислень, електронної комерції, фінансових сервісів та телекомунікаційних систем супроводжується активним впровадженням мікросервісних архітектур, які забезпечують гнучкість розроблення, незалежне масштабування окремих компонентів та можливість ефективної роботи в умовах високого навантаження. Основою взаємодії між компонентами таких систем дедалі частіше виступає подієво-орієнтований підхід, за якого обмін інформацією здійснюється шляхом асинхронної передачі повідомлень через брокери подій. Використання асинхронної взаємодії дозволяє суттєво підвищити рівень паралелізму, збільшити пропускну здатність системи та зменшити залежність між окремими сервісами. Разом із тим асинхронний характер доставки повідомлень породжує низку проблем, пов'язаних із забезпеченням коректності виконання бізнес-процесів, підтриманням узгодженості станів та збереженням логічної послідовності обробки подій.

Особливої актуальності зазначені проблеми набувають у високонавантажених інформаційних системах, де доставка повідомлень реалізується із використанням семантики at-least-once. За таких умов система гарантує доставку кожної події щонайменше один раз, однак допускає її повторне надходження, порушення порядку доставки, виникнення розривів у послідовності подій та необхідність відновлення пропущених станів. У результаті збільшення рівня паралелізму, яке є необхідною умовою підвищення продуктивності та масштабованості мікросервісних систем, одночасно підвищує ймовірність виникнення ситуацій, за яких логічно пов'язані події можуть бути оброблені у неправильному порядку, що призводить до порушення узгодженості даних та некоректного виконання бізнес-операцій.

Аналіз сучасних наукових досліджень показав, що наявні підходи переважно орієнтовані або на забезпечення високої продуктивності шляхом паралельної обробки повідомлень, або на підтримання впорядкованості та узгодженості виконання розподілених операцій. При цьому більшість відомих

рішень не враховує комплексного впливу повторної доставки повідомлень, механізмів ідемпотентності, буферизації подій поза порядком, резервних каналів передачі та нерівномірності розподілу навантаження між потоками подій на характеристики продуктивності та латентності системи. Внаслідок цього залишається невирішеним питання побудови формалізованого механізму, який дозволяв би одночасно забезпечувати впорядкованість обробки подій, високий рівень паралелізму та безперервність функціонування системи за умов часткових відмов транспортного середовища.

Таким чином, виникає наукова суперечність між необхідністю підвищення продуктивності та масштабованості мікросервісних систем шляхом збільшення рівня паралельної обробки подій і необхідністю забезпечення коректної впорядкованої доставки та обробки логічно пов'язаних повідомлень в умовах асинхронної взаємодії, повторної доставки повідомлень і можливих часткових відмов компонентів системи.

Для усунення зазначеної суперечності необхідно розв'язати наукове завдання, яке полягає у розробленні методу упорядкованої паралельної доставки подій для підвищення стабільності й масштабованості мікросервісних систем на основі технологій асинхронного обміну повідомленнями та аналітичного моделювання.

*Зв'язок роботи з науковими програмами, планами, темами.* Дисертаційне дослідження пов'язане з науковими дослідженнями, які проводились у межах науково-дослідних робіт Державного університету інформаційно-комунікаційних технологій на кафедрі технологій цифрового розвитку, зокрема: «Підвищення ефективності процесу управління 3D принтером з використанням методів машинного навчання» (державний реєстраційний номер РК 0124U001849, термін виконання 2024-2026, ДУІКТ, м. Київ).

*Мета і завдання дослідження.* Метою роботи є підвищення ефективності мікросервісних систем доставки подій шляхом розроблення методу упорядкованого паралелізму, аналітичної моделі продуктивності та архітектури безперервної доставки.

Для досягнення мети сформульовано й розв'язано такі **завдання**:



1. Проаналізовано наявні підходи та семантики доставки повідомлень.
2. Формалізовано подієво-орієнтовану модель системи на основі подання події у вигляді кортежу з ключем впорядкування, порядковим номером, часовою міткою та корисним навантаженням.
3. Розроблено метод прикладного впорядкування подій за паралельної обробки.
4. Побудовано аналітичну модель пропускну здатності.
5. Розроблено архітектуру резервування транспортних каналів та керування розподіленими процесами.
6. Виконано експериментальну перевірку отриманих результатів засобами імітаційного моделювання.

Об'єктом дослідження є процеси доставки та обробки подій у розподілених мікросервісних системах.

Предметом дослідження є методи, моделі та архітектурні засоби упорядкованої паралельної доставки подій із гарантуванням коректності, ідемпотентності та відмовостійкості.

*Методи дослідження.* Для розв'язання поставленого наукового завдання використано методи теорії розподілених обчислень і подієво-орієнтованих систем – для формалізації процесів взаємодії мікросервісів та побудови методу упорядкованої паралельної доставки подій; методи математичного моделювання, теорії масового обслуговування та системного аналізу – для розроблення аналітичної моделі впливу паралелізму на продуктивність і латентність системи доставки подій; методи дискретної математики та теорії алгоритмів – для формалізації механізмів впорядкування, буферизації, ідемпотентної обробки та відновлення пропущених станів; методи теорії ймовірностей і математичної статистики – для дослідження впливу повторної доставки повідомлень, часткових відмов транспортного середовища та оцінювання достовірності отриманих результатів; методи імітаційного моделювання та обчислювального експерименту – для перевірки властивостей коректності, живучості, продуктивності й масштабованості запропонованого методу; методи програмної інженерії, контейнеризації та мікросервісного проєктування – для реалізації

експериментального програмного комплексу та апробації розроблених моделей і методів.

*Наукова новизна одержаних результатів:*

1. Набув подальшого розвитку метод упорядкованого паралелізму в межах логічного ключа у мікросервісних системах доставки подій на основі прикладного впорядкування поверх брокера повідомлень із використанням маркування подій ідентифікаторами послідовності та механізму відновлення пропущених станів, що дозволяє забезпечити коректне послідовне опрацювання взаємопов'язаних подій за одночасного паралельного виконання незалежних потоків обробки та підвищити продуктивність системи.

2. Вперше розроблено аналітичну модель впливу паралелізму на продуктивність і латентність системи впорядкованої доставки подій у режимі гарантованої доставки з можливими повторами на основі комплексного врахування ідемпотентної обробки, повторної доставки повідомлень, буферизації пропущених станів і використання резервного каналу доставки, що дозволяє формалізовано оцінювати накладні витрати, визначати оптимальний рівень паралелізму та прогнозувати поведінку системи за умов зміни навантаження і часткових відмов.

3. Удосконалено архітектуру забезпечення безперервності доставки подій у мікросервісних системах на основі інтеграції методу упорядкованого паралелізму, аналітичної моделі оцінювання продуктивності та латентності, основного брокера повідомлень, резервного каналу доставки та механізму узгодженого виконання розподілених кроків, що дозволяє підвищити безперервність виконання процесів, скоротити час відновлення після відмов і забезпечити коректність послідовності виконання операцій в умовах часткових збоїв інфраструктури.

*Практичне значення дисертаційного дослідження:*

1. Набув подальшого розвитку метод упорядкованого паралелізму в межах логічного ключа у мікросервісних системах доставки подій, який на основі прикладного впорядкування поверх брокера повідомлень, використання ідентифікаторів послідовності, внутрішніх черг за ключами та механізму

відновлення пропущених станів забезпечує детерміновану впорядкованість обробки подій у всіх досліджених сценаріях. Практичне застосування запропонованого методу дозволяє зберігати коректну послідовність виконання взаємопов'язаних операцій за одночасного паралельного опрацювання незалежних сутностей та забезпечує прискорення обробки подій до 2,42 раза порівняно з базовими підходами в умовах підвищеного навантаження і часткових збоїв.

2. Вперше розроблена аналітична модель впливу паралелізму на продуктивність і латентність системи впорядкованої доставки подій, побудована на основі формалізації процесів, реалізованих методом упорядкованого паралелізму, дозволяє враховувати вплив повторної доставки повідомлень, ідемпотентної обробки, буферизації пропущених станів та використання резервного каналу доставки на характеристики функціонування системи. Практичне використання моделі забезпечує оцінювання накладних витрат, визначення раціонального рівня паралелізму та прогнозування продуктивності системи з відносною похибкою не більше 5,5 %, що дозволяє використовувати її під час проєктування, масштабування та модернізації високонавантажених мікросервісних систем.

3. Удосконалена архітектура забезпечення безперервності доставки подій, побудована на основі інтеграції методу упорядкованого паралелізму, аналітичної моделі оцінювання продуктивності та латентності, основного брокера повідомлень, резервного каналу доставки та механізму узгодженого виконання розподілених кроків, забезпечує скорочення середнього часу відновлення після збою на 76 % (223,5 мс проти 930,2 мс у базовій архітектурі), підвищення частки успішно доставлених подій до 99,79% порівняно з 95,3 % у традиційних рішеннях, а також підтримання частки успішно завершених Saga-транзакцій на рівні 98,9 % проти 31,8 % за умов нестабільної роботи брокера повідомлень. При цьому додаткові накладні витрати за латентністю у штатному режимі функціонування не перевищують 2,4 %, що дозволяє забезпечити високий рівень безперервності, надійності та доступності мікросервісних систем без істотного зниження їх продуктивності.

*Особистий внесок здобувача.* Усі результати, подані до захисту, розроблені автором особисто. У наукових публікаціях, підготовлених у співавторстві, здобувачеві належать такі результати: розроблено метод упорядкованого паралелізму в межах логічного ключа для мікросервісних систем доставки подій та досліджено його вплив на забезпечення впорядкованості й масштабованості обробки повідомлень [1]; побудовано аналітичну модель впливу паралелізму на продуктивність системи впорядкованої доставки подій із семантикою at-least-once та виконано оцінювання накладних витрат, пов'язаних із повторною доставкою повідомлень і підтриманням впорядкованості [2]; розроблено базову модель методу упорядкованого паралелізму в межах логічного ключа та обґрунтовано принципи його застосування в подієво-орієнтованих мікросервісних системах [3]; сформовано концептуальну модель моніторингу складних розподілених кіберфізичних систем та запропоновано підходи до інтеграції подієво-орієнтованих механізмів взаємодії компонентів критичної інфраструктури [4]; розроблено архітектуру забезпечення безперервності доставки подій на основі використання RabbitMQ, резервних HTTP-каналів доставки та механізму Saga для підвищення живучості системи та скорочення часу відновлення після відмов [5]; виконано дослідження методів багатовимірного шкалювання та їх програмної реалізації для аналізу багатовимірних даних [6]; проведено аналіз алгоритмів консенсусу в розподілених системах та визначено критерії їх вибору залежно від вимог до продуктивності, надійності та узгодженості даних [7].

*Апробація результатів дисертації.* Ключові положення та практичні результати дисертаційної роботи пройшли апробацію шляхом представлення на науково-практичних конференціях:

II Всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу» (18 листопада 2024 року, м. Київ);

II Міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії» (19-21 грудня 2024 року, м. Київ);

Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії» (26 листопада 2024 року, м. Київ);

Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» (24 квітня 2025 року, м. Київ);

II Всеукраїнська науково-технічна конференція «Виклики та рішення в програмній інженерії» (26 листопада 2025 року, м. Київ).

*Структура та обсяг дисертації.* Дисертаційна робота складається з анотації, змісту, вступу, чотирьох розділів, загальних висновків, списку використаних джерел та додатків. Робота містить 57 рисунків, 45 таблиць та 6 сторінок додатків. Список використаних джерел налічує 138 найменувань. Загальний обсяг дисертації становить 264 сторінка, з них 222 сторінки основного тексту.

## РОЗДІЛ 1. ЗАГАЛЬНІ ПІДХОДИ ДО ПОБУДОВИ МІКРОСЕРВІСНИХ СИСТЕМ ТА ОРГАНІЗАЦІЇ ДОСТАВКИ ПОДІЙ

### 1.1. Теоретичні основи мікросервісної архітектури та подієво-орієнтованих систем

Сучасний етап розвитку розподілених інформаційних систем характеризується суттєвим ускладненням програмних платформ, збільшенням інтенсивності потоків даних та необхідністю забезпечення безперервного функціонування сервісів в умовах змінного навантаження і високих вимог до масштабованості. Традиційні монолітні архітектурні підходи, у межах яких функціональна логіка системи реалізується у вигляді єдиного програмного модуля, поступово втрачають ефективність при побудові високонавантажених середовищ, оскільки централізована структура ускладнює горизонтальне масштабування, підвищує залежність між компонентами та збільшує ризик каскадних відмов. У зв'язку з цим у сучасній практиці проектування програмного забезпечення широкого поширення набули мікросервісні архітектури, які орієнтовані на декомпозицію системи на сукупність незалежних функціональних сервісів, здатних автономно виконувати окремі бізнес-функції та взаємодіяти між собою через стандартизовані механізми обміну повідомленнями (табл. 1).

Таблиця 1.1

#### Порівняння монолітної та мікросервісної архітектури

Характеристика	Монолітна архітектура	Мікросервісна архітектура
Масштабування	централізоване	незалежне
Відмовостійкість	низька	підвищена
Зв'язність	висока	низька
Оновлення	складне	незалежне
Паралелізм	обмежений	високий

Концепція мікросервісної архітектури базується на принципі функціональної ізолюваності компонентів, відповідно до якого кожен сервіс реалізує завершену предметно-орієнтовану функцію та може розроблятися, масштабуватися й розгортатися незалежно від інших компонентів системи. Такий підхід забезпечує гнучкість еволюції програмної платформи, спрощує оновлення окремих модулів та дозволяє адаптувати обчислювальні ресурси відповідно до поточного навантаження. Водночас децентралізований характер мікросервісної взаємодії істотно ускладнює забезпечення узгодженості даних, коректності виконання розподілених операцій та синхронізації процесів у системі, оскільки відсутність єдиного глобального контексту виконання призводить до необхідності використання спеціалізованих механізмів координації та обміну подіями.

Однією з фундаментальних характеристик мікросервісних систем є переважання асинхронної взаємодії між компонентами, при якій передача інформації здійснюється не через безпосередні синхронні виклики, а шляхом генерації та обробки подій. Такий підхід отримав розвиток у межах подієво-орієнтованої архітектури (Event-Driven Architecture, EDA), яка передбачає організацію системи у вигляді сукупності незалежних виробників і споживачів подій, взаємодія між якими реалізується через проміжне транспортне середовище. У цьому випадку подія розглядається як факт зміни стану певного об'єкта або бізнес-процесу, що ініціює подальшу реакцію інших компонентів системи.

Для узагальненого представлення структури подієво-орієнтованої мікросервісної системи доцільно розглянути її типову архітектурну модель, наведену на рисунку 1.1.

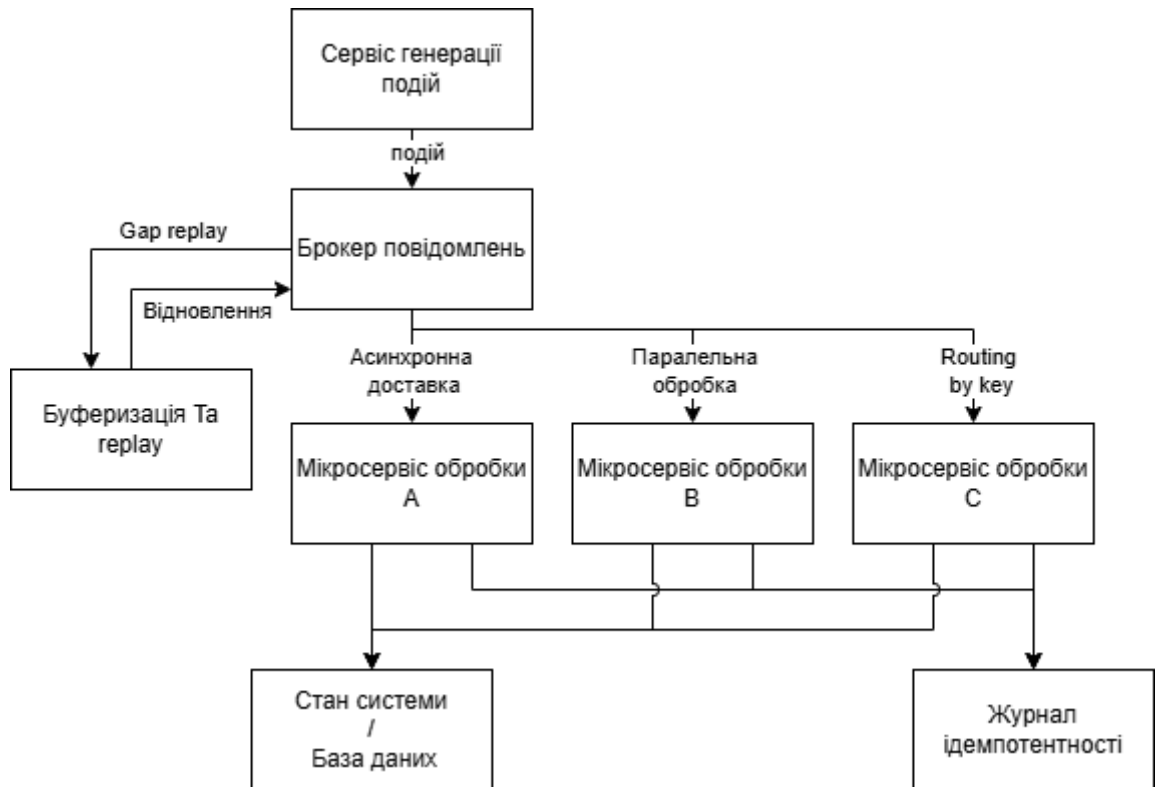


Рис. 1.1. Узагальнена архітектура мікросервісної подієво-орієнтованої системи

Як показано на рисунку 1.1, у подієво-орієнтованій мікросервісній системі взаємодія між компонентами здійснюється через брокер повідомлень, який виконує функції маршрутизації, буферизації та асинхронної доставки подій між незалежними сервісами. Генерація подій відбувається у сервісі-джерелі, після чого повідомлення передаються до транспортного середовища, де можуть бути розподілені між декількома мікросервісами обробки відповідно до механізмів маршрутизації та балансування навантаження.

Особливістю наведеної архітектури є можливість паралельного виконання декількох потоків обробки подій, що забезпечує високу масштабованість системи. Водночас асинхронний характер взаємодії створює ризик порушення послідовності доставки повідомлень, у зв'язку з чим у структурі системи додатково використовуються механізми буферизації, повторної доставки (*replay*) та контролю ідемпотентності.

Наявність журналу ідемпотентності дозволяє фіксувати вже оброблені події та запобігати повторній зміні стану системи у випадку дублювання



повідомлень. У свою чергу, механізм буферизації та відновлення забезпечує можливість реконструкції пропущених станів і підтримання коректної послідовності обробки подій навіть за умов часткових збоїв або асинхронних затримок доставки.

Наведена архітектурна модель демонструє, що сучасні мікросервісні системи поєднують механізми асинхронного паралелізму з додатковими засобами забезпечення узгодженості та надійності, що формує основу для побудови методів упорядкованої обробки подій у розподіленому середовищі.

Використання подієво-орієнтованого підходу дозволяє суттєво знизити рівень зв'язності між сервісами, оскільки виробник події не потребує інформації про конкретних споживачів або порядок їх виконання. Це створює передумови для побудови масштабованих і відмовостійких систем, у яких окремі компоненти можуть функціонувати незалежно та змінюватися без необхідності модифікації всієї платформи. Крім того, асинхронна передача подій дозволяє більш ефективно використовувати обчислювальні ресурси, оскільки обробка повідомлень може виконуватися паралельно декількома екземплярами сервісів.

Разом із тим перехід до подієво-орієнтованої взаємодії породжує низку принципово нових проблем, які не є характерними для централізованих монолітних систем. Насамперед це стосується забезпечення узгодженості станів у розподіленому середовищі, оскільки різні компоненти системи можуть отримувати та обробляти події з різними затримками, а порядок фактичного надходження повідомлень не завжди відповідає логічній послідовності виконання бізнес-процесів. Додатково ситуація ускладнюється використанням брокерів повідомлень, які реалізують асинхронну маршрутизацію та буферизацію потоків подій, але у більшості випадків не гарантують глобального впорядкування повідомлень між незалежними потоками.

Особливого значення зазначена проблема набуває у системах, де послідовність виконання операцій безпосередньо впливає на коректність зміни стану. До таких систем належать платформи електронної комерції, фінансові сервіси, системи управління замовленнями, телекомунікаційні платформи та інші високонавантажені інформаційні середовища, у яких порушення порядку

обробки подій може призводити до виникнення неузгоджених транзакцій, дублювання операцій або втрати частини логічного контексту процесу. За таких умов забезпечення високого рівня паралелізму повинно поєднуватися із підтриманням детермінованої послідовності виконання подій у межах окремих логічних процесів.

У більшості сучасних реалізацій подієво-орієнтованих систем доставка повідомлень здійснюється із використанням семантики *at-least-once*, відповідно до якої система гарантує доставку кожної події щонайменше один раз, однак допускає її повторне надходження у випадках мережових збоїв, втрати підтвердження або перезапуску компонентів (табл. 1.2). Такий підхід підвищує надійність передачі повідомлень, проте одночасно створює необхідність реалізації механізмів ідемпотентної обробки та контролю послідовності виконання, оскільки одна й та сама подія потенційно може бути оброблена декілька разів або надійти до системи із порушенням порядку.

Таблиця 1.2

Особливості семантики доставки повідомлень

Семантика	Особливість	Недоліки
At-most-once	без повторів	можлива втрата
At-least-once	гарантована доставка	дублювання
Exactly-once	строгість	висока складність

У контексті подієво-орієнтованих мікросервісних систем особливого значення набуває роль транспортного середовища, яке виконує функції приймання, маршрутизації, буферизації та доставки повідомлень між компонентами системи. Як правило, для реалізації таких функцій використовуються брокери повідомлень, що забезпечують асинхронну взаємодію між сервісами та дозволяють відокремити процес генерації подій від процесу їх обробки. Використання брокерів повідомлень створює можливість незалежного масштабування компонентів системи, зменшує рівень часової залежності між сервісами та дозволяє реалізовувати складні сценарії розподіленої взаємодії.

Водночас застосування брокерів повідомлень істотно впливає на характер функціонування системи, оскільки процес доставки подій перестає бути строго детермінованим. Залежно від конфігурації транспортного середовища, кількості споживачів, параметрів балансування навантаження та механізмів підтвердження доставки одна й та сама подія може бути оброблена із різними затримками або повторно надіслана до системи. Крім того, за умов паралельної обробки декілька потоків подій можуть надходити до сервісів у порядку, відмінному від початкової логічної послідовності їх генерації. У результаті виникає проблема забезпечення причинно-наслідкової узгодженості процесів, яка є однією з центральних проблем сучасних розподілених систем.

Для часткового вирішення зазначеної проблеми у практиці побудови подієво-орієнтованих систем використовуються різні моделі організації потоків повідомлень. Одним із найбільш поширених підходів є використання логічного групування подій за певним ключем маршрутизації, який визначає належність події до конкретного бізнес-процесу або об'єкта предметної області. Такий підхід дозволяє локалізувати послідовність обробки в межах окремого контексту та забезпечити коректність зміни стану для конкретного логічного потоку. Однак навіть за наявності механізмів групування проблема впорядкованості не усувається повністю, оскільки паралельне виконання сервісів та асинхронна доставка повідомлень продовжують створювати ризик виникнення розривів у послідовності подій.

У теорії розподілених систем зазначена проблема безпосередньо пов'язана із суперечністю між вимогами до масштабованості та необхідністю підтримання строгого порядку виконання операцій. Підвищення рівня паралелізму дозволяє збільшити пропускну здатність системи та скоротити час очікування обробки повідомлень, проте одночасно ускладнює координацію між потоками виконання. У випадку повної синхронізації між сервісами досягається висока узгодженість, однак це призводить до зростання затримок та втрати переваг асинхронної архітектури. Навпаки, повністю незалежне паралельне виконання підвищує продуктивність, але створює ризик порушення послідовності обробки подій та виникнення неузгоджених станів системи.

У зв'язку з цим у сучасних мікросервісних системах значного поширення набули гібридні підходи, які поєднують механізми локального впорядкування із засобами асинхронного паралелізму. Сутність таких підходів полягає у забезпеченні строгого порядку обробки лише в межах окремого логічного ключа, тоді як події, що належать до різних ключів, можуть оброблятися паралельно незалежними потоками. Це дозволяє суттєво знизити рівень глобальної синхронізації та одночасно зберегти коректність виконання бізнес-процесів.

Додатковим фактором, що впливає на функціонування подієво-орієнтованих систем, є наявність часткових збоїв, які є невід'ємною характеристикою розподіленого середовища. На відміну від монолітних систем, у мікросервісній архітектурі відмова окремого компонента не завжди призводить до повного припинення роботи системи, однак може викликати втрату частини повідомлень, порушення узгодженості станів або накопичення необроблених подій у транспортному середовищі. У таких умовах особливого значення набувають механізми повторної доставки повідомлень, буферизації, журналювання та компенсаційної обробки, що дозволяють забезпечити безперервність функціонування системи навіть у випадках тимчасової недоступності окремих сервісів.

Не менш важливим аспектом є проблема забезпечення ідемпотентності операцій у середовищі асинхронної доставки повідомлень. Оскільки повторна доставка є типовою властивістю систем із семантикою *at-least-once*, алгоритми обробки подій повинні гарантувати, що повторне надходження однієї й тієї ж події не призведе до некоректної зміни стану системи. Це вимагає використання механізмів контролю вже виконаних операцій, ведення журналів оброблених подій або реалізації бізнес-логіки таким чином, щоб повторне виконання операції не впливало на кінцевий результат.

З теоретичної точки зору мікросервісна подієво-орієнтована система являє собою слабко зв'язану динамічну структуру, у якій процеси генерації, передачі та обробки подій відбуваються незалежно та можуть бути просторово й часово розподіленими. Саме ця особливість створює передумови для виникнення конфлікту між вимогами до масштабованості та необхідністю підтримання

детермінованого порядку виконання операцій, що є однією з ключових проблем сучасних розподілених систем. Саме тому забезпечення коректної обробки подій у розподіленому середовищі є комплексною задачею, яка охоплює проблеми впорядкування повідомлень, координації паралельних потоків виконання, відновлення після збоїв та підтримання узгодженості станів. Саме поєднання зазначених аспектів формує передумови для розроблення методів упорядкованого паралелізму, здатних забезпечити одночасне виконання вимог до масштабованості, надійності та детермінованості функціонування сучасних мікросервісних систем.

Таким чином, аналіз теоретичних основ мікросервісної архітектури та подієво-орієнтованих систем показав, що використання асинхронної взаємодії та брокерів повідомлень створює передумови для побудови масштабованих і відмовостійких інформаційних платформ, однак одночасно породжує проблеми підтримання узгодженості станів і коректності обробки подій у розподіленому середовищі. Особливого значення при цьому набуває механізм доставки повідомлень, оскільки саме він визначає порядок надходження подій, можливість їх дублювання, втрати або повторної передачі, що безпосередньо впливає на стабільність функціонування мікросервісної системи.

У зв'язку з цим доцільним є більш детальний розгляд семантик доставки повідомлень та аналіз їх впливу на коректність обробки подій у подієво-орієнтованих архітектурах.

## **1.2. Семантики доставки повідомлень та їх вплив на коректність обробки подій**

У сучасних подієво-орієнтованих системах механізм доставки повідомлень є одним із ключових компонентів, що визначає характер взаємодії між сервісами та впливає на загальну надійність функціонування розподіленої системи. На відміну від централізованих монолітних платформ, у яких передача даних між модулями відбувається у межах єдиного процесу виконання, мікросервісні архітектури використовують асинхронну взаємодію через транспортне

середовище, що призводить до виникнення затримок, повторної доставки повідомлень та потенційної втрати частини подій. У таких умовах саме семантика доставки визначає рівень гарантій, які система надає щодо факту та порядку передачі повідомлень між компонентами.

З теоретичної точки зору семантика доставки визначає множину допустимих станів транспортного середовища та правила переходу повідомлення між етапами генерації, передачі, буферизації та підтвердження отримання. Інакше кажучи, вона формалізує умови, за яких подія вважається успішно доставленою, а також визначає допустимість повторної передачі або втрати повідомлення. Від вибору відповідної семантики безпосередньо залежать вимоги до механізмів обробки подій, синхронізації станів та забезпечення узгодженості даних у розподіленому середовищі.

У більшості сучасних систем доставки повідомлень використовуються три базові моделі семантики передачі: *at-most-once*, *at-least-once* та *exactly-once*. Кожна з них визначає різний баланс між продуктивністю, надійністю та складністю реалізації системи. При цьому жодна із зазначених моделей не може вважатися універсальною, оскільки їх ефективність залежить від вимог конкретного прикладного середовища та допустимого рівня ризику втрати або дублювання повідомлень.

Семантика *at-most-once* передбачає, що кожне повідомлення може бути доставлене не більше одного разу. У цьому випадку транспортне середовище не виконує повторних спроб передачі у разі втрати повідомлення або відсутності підтвердження отримання. Такий підхід характеризується мінімальними накладними витратами та високою швидкістю, однак не забезпечує гарантованої доставки подій, що робить його непридатним для критичних систем, у яких втрата повідомлення може призвести до порушення цілісності бізнес-процесу.

На відміну від цього, семантика *at-least-once* орієнтована на забезпечення гарантованої доставки повідомлень навіть за умов часткових мережескопних збоїв або тимчасової недоступності сервісів. У межах цієї моделі повідомлення вважається доставленим лише після отримання підтвердження від споживача, а

у випадку відсутності такого підтвердження транспортне середовище ініціює повторну передачу події. Саме ця властивість робить модель *at-least-once* найбільш поширеною у сучасних брокерах повідомлень та системах асинхронної взаємодії.

Разом із тим використання повторної доставки створює принципово нову проблему – можливість дублювання подій. У ситуаціях, коли повідомлення було успішно оброблено, але підтвердження не було доставлено до брокера через мережевий збій або перезапуск компонента, система змушена повторно передати вже виконану подію. У результаті один і той самий логічний процес може бути ініційований декілька разів, що без додаткових механізмів контролю призводить до порушення узгодженості стану системи.

У зв'язку з цим використання семантики *at-least-once* потребує впровадження механізмів ідемпотентної обробки та контролю послідовності виконання подій. Ідемпотентність забезпечує незмінність кінцевого стану системи при повторній обробці одного й того ж повідомлення, тоді як механізми впорядкування дозволяють підтримувати правильну послідовність виконання операцій навіть у випадку асинхронного надходження повідомлень або повторної доставки подій. Саме необхідність поєднання високого рівня паралелізму із забезпеченням коректності обробки подій формує основу для подальшого дослідження методів упорядкованого паралелізму у мікросервісних системах.

Для більш повного аналізу особливостей різних моделей доставки повідомлень доцільно провести їх порівняння за основними характеристиками, що впливають на функціонування подієво-орієнтованих систем.

Як видно з таблиці 1.3, семантика *at-most-once* характеризується мінімальними накладними витратами, однак не забезпечує необхідного рівня надійності для критичних систем. У свою чергу, модель *exactly-once* забезпечує найвищий рівень узгодженості, проте її реалізація потребує складних механізмів глобальної координації, транзакційної синхронізації та підтвердження станів, що істотно знижує продуктивність і масштабованість системи. Саме тому в сучасних мікросервісних архітектурах найбільшого поширення набула семантика *at-least-*

once, яка забезпечує компроміс між надійністю доставки та ефективністю функціонування системи.

Таблиця 1.3

Порівняльна характеристика семантик доставки повідомлень

Семантика доставки	Гарантія доставки	Можливість дублювання	Ризик втрати повідомлень	Складність реалізації
At-most-once	відсутня	відсутня	висока	низька
At-least-once	гарантована	наявна	низька	середня
Exactly-once	гарантована	відсутня	мінімальна	висока

Разом із тим використання семантики *at-least-once* породжує проблему дублювання повідомлень та порушення порядку їх обробки. Для наочного представлення процесу повторної доставки подій доцільно розглянути відповідну схему функціонування транспортного середовища.

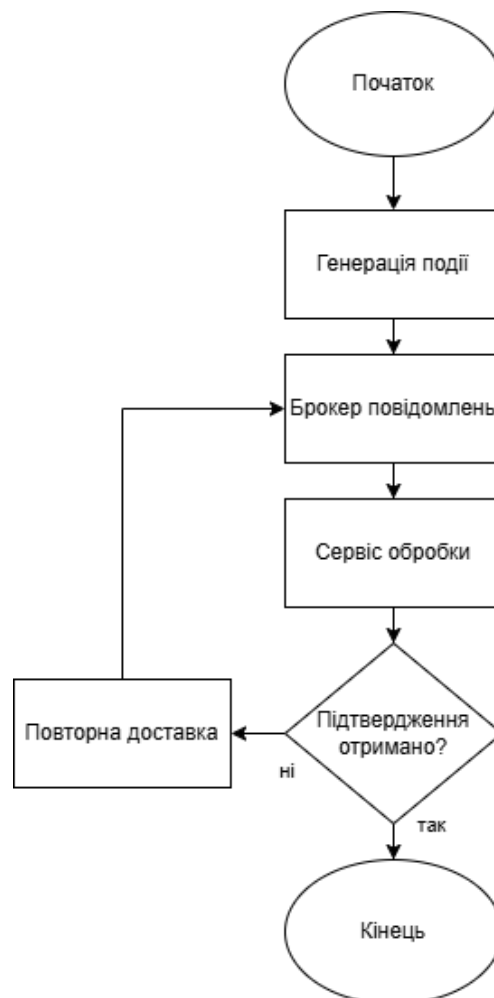


Рис. 1.2. Схема повторної доставки повідомлень у моделі *at-least-once*



Як показано на рисунку 1.2, у випадку відсутності підтвердження доставки транспортне середовище ініціює повторну передачу повідомлення. Причиною такої ситуації можуть бути мережеві затримки, відмова сервісу обробки, втрата підтвердження або перезапуск окремих компонентів системи. У результаті одна й та сама подія може надходити до сервісу декілька разів, що створює необхідність реалізації механізмів ідемпотентної обробки.

З формальної точки зору процес доставки повідомлення можна представити як стохастичний процес переходу події між станами транспортного середовища. Нехай  $P_d$  — ймовірність успішної доставки повідомлення за одну спробу, тоді ймовірність того, що повідомлення буде доставлене після не більше ніж  $R$  повторних спроб, визначається співвідношенням:

$$P_{success} = 1 - (1 - P_d)^R \quad (1.1)$$

Наведене співвідношення демонструє, що збільшення кількості повторних спроб дозволяє асимптотично наблизити ймовірність доставки до одиниці, однак одночасно призводить до зростання навантаження на транспортне середовище та підвищення ризику дублювання повідомлень.

Для ілюстрації залежності ймовірності успішної доставки від кількості повторних спроб доцільно розглянути графік, наведений на рисунку 1.3.

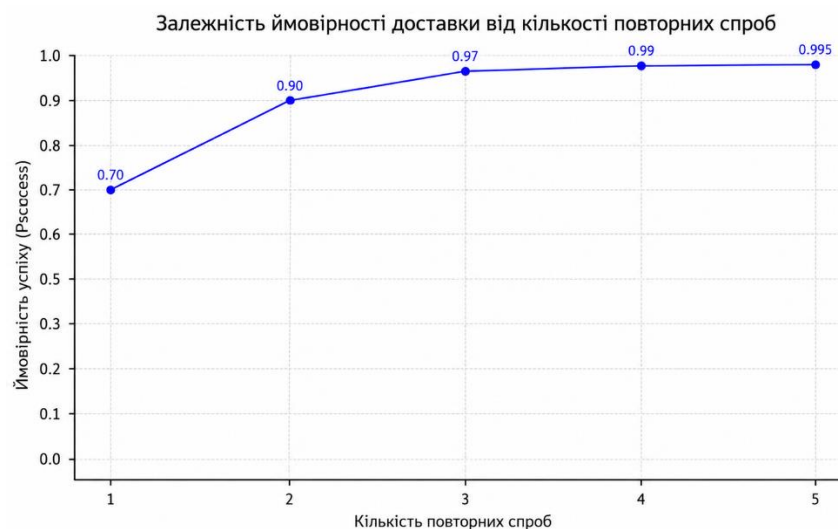


Рис. 1.3. Залежність ймовірності успішної доставки повідомлення від кількості повторних спроб

З рисунка 1.3 видно, що зі збільшенням кількості повторних спроб імовірність успішної доставки швидко наближається до одиниці. Проте після певного значення кількості спроб приріст імовірності стає незначним, тоді як навантаження на систему продовжує зростати. Це свідчить про необхідність оптимального вибору параметрів повторної доставки залежно від вимог до надійності та продуктивності системи.

Окремої уваги потребує проблема порушення порядку обробки подій, яка виникає внаслідок асинхронного характеру взаємодії між компонентами системи. У випадку паралельної обробки повідомлень різними сервісами фактичний порядок надходження подій може не відповідати логічній послідовності їх генерації. Така ситуація є особливо критичною для систем, у яких стан об'єкта формується послідовністю залежних операцій.

Для формалізації цієї проблеми розглянемо послідовність подій:

$$E = e_1, e_2, \dots, e_n, \quad (1.2)$$

де кожна подія має порядковий номер:

$$s(e_i) = i \quad (1.3)$$

Умовою коректної обробки є виконання співвідношення:

$$e_i \prec e_j \Rightarrow s(e_i) < s(e_j) \quad (1.4)$$

де  $\prec$  означає фактичний порядок виконання подій у системі.

Порушення цієї умови призводить до некоректної зміни стану системи та виникнення неузгоджених транзакцій. Саме тому у подієво-орієнтованих мікросервісних системах виникає необхідність реалізації механізмів локального впорядкування та контролю послідовності обробки повідомлень.

Для наочного представлення впливу асинхронної доставки на порядок виконання подій доцільно розглянути схему, подану на рисунку 1.4.

Як показано на рисунку 1.4, навіть за гарантованої доставки повідомлень система може переходити у неузгоджений стан у випадку порушення логічної послідовності виконання операцій. Це свідчить про те, що проблема коректності обробки подій у мікросервісних системах пов'язана не лише із фактом доставки повідомлення, але й із забезпеченням правильного порядку його виконання.

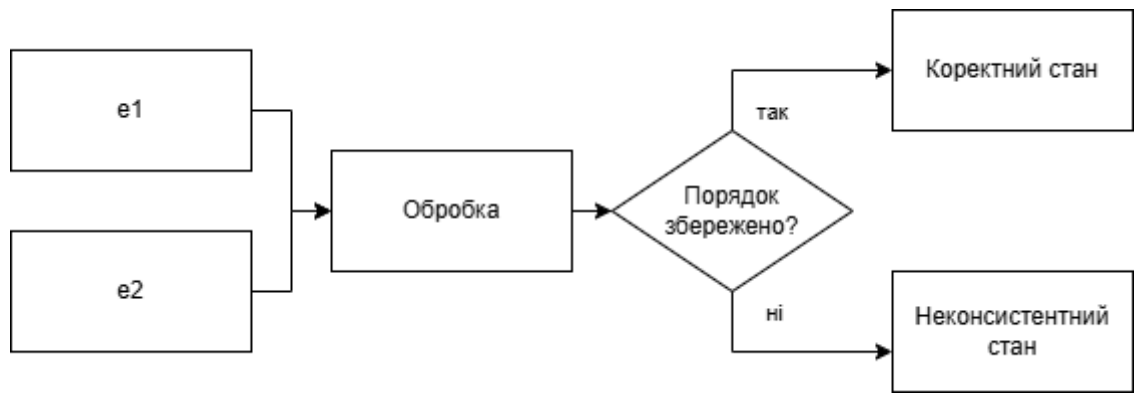


Рис. 1.4. Вплив асинхронної доставки на коректність обробки подій

Таким чином, аналіз семантик доставки повідомлень показує, що найбільш практичною моделлю для сучасних масштабованих систем є семантика *at-least-once*, однак її використання потребує впровадження механізмів ідемпотентності, буферизації та впорядкованої обробки подій. Саме ці аспекти формують основу подальших досліджень методів забезпечення коректності та узгодженості в асинхронних мікросервісних архітектурах.

Однією з фундаментальних особливостей асинхронних систем є відсутність глобального часу та єдиного центру координації, внаслідок чого кожен сервіс формує власне локальне уявлення про порядок надходження повідомлень. Це означає, що навіть за умови використання одного брокера повідомлень різні споживачі можуть отримувати та обробляти події з різними затримками, а отже, фактична послідовність виконання операцій залежить не лише від моменту генерації події, але й від характеристик мережі, швидкодії окремих сервісів та параметрів балансування навантаження.

Для ілюстрації цієї особливості доцільно розглянути узагальнену часову діаграму асинхронної доставки подій.

Як показано на рисунку 1.5, навіть якщо події були згенеровані у правильному порядку, фактичний момент їх обробки різними сервісами може відрізнитися через асинхронний характер взаємодії. У результаті подія  $e_2$  потенційно може бути оброблена раніше за  $e_1$ , що призводить до порушення причинно-наслідкової узгодженості системи.

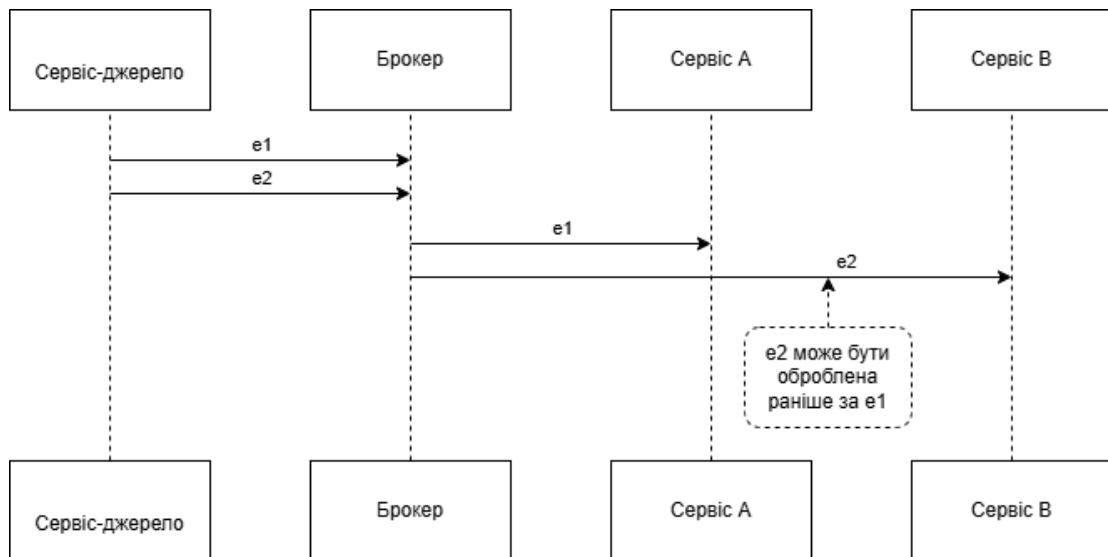


Рис. 1.5. Приклад порушення логічної послідовності подій в асинхронному середовищі

З точки зору теорії розподілених систем така ситуація є наслідком конкуренції між вимогами до продуктивності та необхідністю підтримання детермінованого порядку виконання операцій. Підвищення рівня паралелізму дозволяє збільшити пропускну здатність системи:

$$Throughput \sim c, \quad (1.5)$$

де  $c$  – кількість паралельних потоків обробки. Однак зі зростанням рівня паралелізму збільшується ймовірність порушення порядку виконання:

$$P_{order} \sim 1 - e^{-\alpha c}, \quad (1.6)$$

де  $\alpha$  – коефіцієнт інтенсивності конкуренції між потоками.

Наведені співвідношення відображають фундаментальну суперечність сучасних подієво-орієнтованих систем: збільшення продуктивності шляхом паралельної обробки подій одночасно підвищує ризик втрати коректної послідовності виконання операцій. Саме ця суперечність обумовлює необхідність побудови спеціалізованих методів упорядкованого паралелізму, які дозволяють поєднати високий рівень масштабованості із підтриманням локальної детермінованості обробки подій.

Для узагальнення основних проблем, що виникають у системах асинхронної доставки повідомлень, доцільно подати їх у вигляді таблиці 1.4.

Таблиця 1.4

## Основні проблеми асинхронної доставки подій

Проблема	Причина виникнення	Наслідки
Повторна доставка	втрата підтвердження	дублювання операцій
Порушення порядку	паралельна обробка	неузгоджений стан
Втрата повідомлень	мережеві збої	неповнота даних
Затримки доставки	перевантаження системи	зростання латентності
Блокування обробки	очікування пропущених подій	деградація продуктивності

З аналізу таблиці 1.4 видно, що проблеми асинхронної доставки мають комплексний характер та взаємно впливають одна на одну. Зокрема, повторна доставка повідомлень потребує впровадження механізмів ідемпотентності, порушення порядку обумовлює необхідність локального впорядкування подій, а затримки доставки можуть призводити до блокування логічних потоків обробки.

Для більш повного аналізу особливостей організації асинхронної взаємодії у мікросервісних системах доцільно розглянути характеристики найбільш поширених брокерів повідомлень, які використовуються для реалізації подієво-орієнтованих архітектур.

Як видно з таблиці 1.5, жоден із сучасних брокерів повідомлень не забезпечує універсального механізму глобального впорядкування подій у розподіленому середовищі. Більшість рішень підтримують лише локальне впорядкування в межах окремих черг, потоків або partition, тоді як при масштабуванні системи та паралельній обробці повідомлень проблема забезпечення глобальної узгодженості переноситься на прикладний рівень системи. Саме тому у сучасних мікросервісних архітектурах виникає необхідність реалізації додаткових механізмів контролю послідовності виконання подій та координації паралельних потоків обробки.

Таблиця 1.5

## Порівняльна характеристика брокерів повідомлень

Брокер повідомлень	Особливості архітектури	Механізми впорядкування	Основні переваги	Основні обмеження
RabbitMQ	черги повідомлень, AMQP	локальне впорядкування в межах черги	гнучка маршрутизація, висока надійність	складність глобального впорядкування
Apache Kafka	журнал подій (log-based)	впорядкування в межах partition	висока пропускна здатність	залежність від partitioning
NATS	lightweight messaging	обмежене впорядкування	мінімальні затримки	спрощена модель гарантій
Redis Streams	потокowa in-memory модель	часткове впорядкування	висока швидкодія	обмежена відмовостійкість

Окремої уваги потребує фундаментальне обмеження розподілених систем, відоме як CAP-теорема, відповідно до якої система не може одночасно забезпечувати строгі гарантії узгодженості (*Consistency*), доступності (*Availability*) та стійкості до мережевих розділень (*Partition tolerance*). У контексті подієво-орієнтованих мікросервісних архітектур це означає, що забезпечення високої доступності та масштабованості неминуче супроводжується послабленням глобальної синхронізації між компонентами системи. У результаті асинхронна доставка повідомлень та паралельна обробка подій стають не лише архітектурною перевагою, але й джерелом потенційних порушень узгодженості станів, що потребує застосування спеціалізованих методів локального впорядкування та ідемпотентної обробки.

Проведений аналіз показав, що асинхронна доставка повідомлень у мікросервісних системах створює фундаментальну суперечність між вимогами до масштабованості, відмовостійкості та необхідністю підтримання коректної

послідовності обробки подій. Використання семантики *at-least-once*, яка є найбільш поширеною у сучасних подієво-орієнтованих архітектурах, забезпечує високий рівень надійності доставки, однак одночасно породжує проблеми дублювання повідомлень, порушення порядку виконання операцій та виникнення неузгоджених станів системи.

У зв'язку з цим забезпечення коректності функціонування розподілених систем потребує застосування спеціалізованих механізмів контролю послідовності подій, координації паралельної обробки та підтримання узгодженості станів у середовищі асинхронної взаємодії. Саме тому наступним етапом дослідження є аналіз існуючих методів забезпечення впорядкованості подій у розподілених системах.

### **1.3. Методи забезпечення впорядкованості подій у розподілених системах**

Однією з ключових проблем функціонування подієво-орієнтованих мікросервісних систем є забезпечення коректного порядку обробки подій у середовищі асинхронної взаємодії та паралельного виконання процесів. На відміну від централізованих систем, у яких порядок виконання операцій визначається єдиним потоком управління, у розподілених архітектурах кожен компонент системи функціонує автономно та формує власну локальну послідовність обробки повідомлень. У результаті фактичний порядок виконання операцій може відрізнятися від логічної послідовності генерації подій, що створює ризик порушення причинно-наслідкових зв'язків між елементами бізнес-процесу.

Проблема впорядкованості подій є однією з фундаментальних задач теорії розподілених систем, оскільки саме порядок виконання операцій визначає коректність переходу системи між станами. У випадку, коли події обробляються у неправильній послідовності, система може перейти до неузгодженого стану навіть за умови гарантованої доставки всіх повідомлень. Особливо критично це проявляється у системах, де зміна стану формується послідовністю залежних

операцій, зокрема у фінансових сервісах, системах управління замовленнями, телекомунікаційних платформах та інших високонавантажених інформаційних середовищах.

З теоретичної точки зору задача впорядкування подій полягає у забезпеченні такого механізму обробки, за якого для будь-якої послідовності залежних подій зберігається причинно-наслідковий порядок їх виконання незалежно від особливостей транспортного середовища, затримок доставки або рівня паралелізму системи. У загальному випадку ця умова може бути подана у вигляді відношення передування:

$$e_i \rightarrow e_j, \quad \dots(1.7)$$

де подія  $e_i$  повинна бути оброблена раніше за подію  $e_j$ , символ  $\rightarrow$  визначає відношення причинно-наслідкового передування (*happened-before relation*).. Забезпечення виконання цього співвідношення в умовах асинхронної взаємодії є основною задачею механізмів впорядкування у розподілених системах.

Подальший розвиток методів забезпечення впорядкованості подій у розподілених системах безпосередньо пов'язаний із дослідженнями логічного часу та причинно-наслідкових залежностей між подіями, які були закладені у фундаментальних роботах Л. Лампорта. У роботі «Time, Clocks, and the Ordering of Events in a Distributed System» було вперше сформульовано відношення *happened-before*, що дозволило формалізувати причинно-наслідковий порядок подій у середовищі, де відсутній глобальний фізичний час.

Відповідно до підходу Лампорта, якщо подія  $e_i$  причинно передуює події  $e_j$ , то між ними існує відношення (1.7). Таке передування виникає у трьох основних випадках:

- події належать одному процесу та виконуються послідовно;
- одна подія є передачею повідомлення, а інша – його отриманням;
- передування є транзитивним.

На основі цього підходу було запропоновано механізм логічних годинників Лампорта, у межах якого кожен процес підтримує локальний лічильник подій:

$$LC_i \leftarrow LC_i + 1. \quad (1.8)$$



Під час передачі повідомлення значення логічного годинника додається до повідомлення, а при його отриманні виконується операція:

$$LC_j \leftarrow \max(LC_j, LC_i) + 1. \quad (1.9)$$

Наведений механізм забезпечує виконання умови:

$$e_i \rightarrow e_j \Rightarrow LC(e_i) < LC(e_j), \quad (1.10)$$

що дозволяє підтримувати частковий порядок подій у розподіленому середовищі.

Для наочного представлення принципу роботи логічних годинників доцільно розглянути схему синхронізації подій у розподіленій системі.

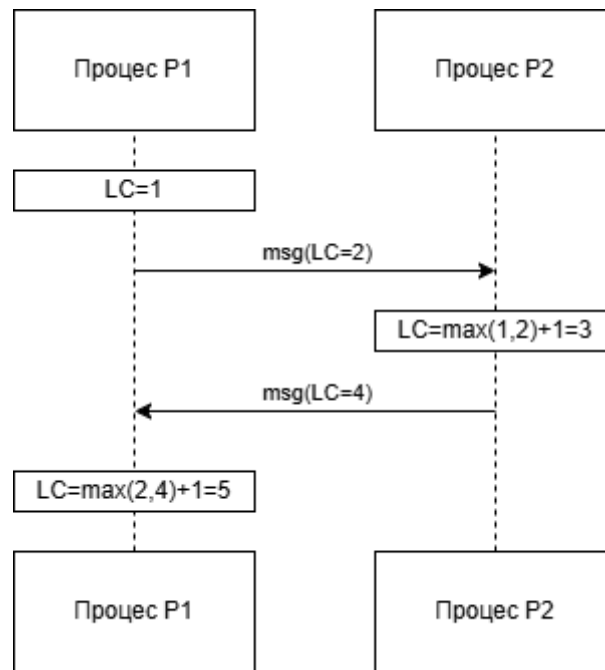


Рис. 1.6. Принцип функціонування логічних годинників Лампорта

Як показано на рисунку 1.6, логічні годинники дозволяють встановлювати причинно-наслідкові зв'язки між подіями без використання глобального фізичного часу. Разом із тим підхід Лампорта не дозволяє однозначно визначати незалежні або конкурентні події, оскільки виконання умови

$$LC(e_i) < LC(e_j) \quad (1.11)$$

не гарантує наявності причинного зв'язку між ними. Це означає, що логічні годинники забезпечують лише часткове впорядкування подій.

З метою усунення зазначеного обмеження подальшого розвитку набули механізми векторних годинників (*Vector Clocks*), у межах яких кожен процес підтримує не одне скалярне значення часу, а вектор:

$$VC_i = (v_1, v_2, \dots, v_n), \quad (1.12)$$

де  $n$  – кількість процесів у системі.

У цьому випадку кожен елемент вектора відображає локальне уявлення процесу про стан часу інших вузлів системи. При передачі повідомлення до нього додається поточний вектор часу, а при отриманні виконується операція:

$$VC_i[k] < \max(VC_i[k], VC_j[k]). \quad (1.13)$$

На відміну від годинників Лампорта, векторні годинники дозволяють не лише визначати порядок подій, але й виявляти конкурентні процеси, між якими відсутній причинно-наслідковий зв'язок.

Для систематизації особливостей логічних та векторних годинників доцільно подати їх порівняльну характеристику.

Таблиця 1.6

Порівняльна характеристика механізмів логічного часу

Метод	Тип впорядкування	Складність	Переваги	Недоліки
Логічні годинники Лампорта	часткове	низька	простота реалізації	не визначає конкурентність
Векторні годинники	причинно- наслідкове	висока	виявлення конкурентних подій	зростання розмірності
Total Order Broadcast	глобальне	дуже висока	повне впорядкування	значні накладні витрати

З аналізу таблиці 1.6 видно, що зі зростанням точності механізмів впорядкування одночасно збільшується складність їх реалізації та обсяг службової інформації, необхідної для підтримання узгодженості. Зокрема, використання векторних годинників у високонавантажених системах призводить до суттєвого збільшення обсягу метаданих, що негативно впливає на продуктивність транспортного середовища та масштабованість системи.

У сучасних мікросервісних архітектурах значного поширення набули також підходи, засновані на локальному впорядкуванні подій за ключем маршрутизації (*partition ordering, routing-key ordering*), у межах яких строгий порядок підтримується лише для подій, що належать одному логічному контексту. Такий підхід дозволяє поєднати високий рівень паралелізму із забезпеченням коректності виконання бізнес-процесів, оскільки незалежні потоки подій можуть оброблятися паралельно без глобальної синхронізації.

Формально множину подій можна подати як:

$$E = \bigcup_{k=1}^m E_k, \quad (1.14)$$

де  $E_k$  – підмножина подій для ключа  $k$ .

У такому випадку умова впорядкованості виконується лише локально:

$$\forall e_i, e_j \in E_k : e_i \rightarrow e_j. \quad (1.15)$$

Саме локальне впорядкування дозволяє істотно знизити накладні витрати на синхронізацію та уникнути необхідності глобального тотального порядку подій у системі. Однак навіть такий підхід не усуває повністю проблему порушення послідовності у випадках повторної доставки, відмов брокера повідомлень або конкурентної обробки декількома споживачами.

У сучасній літературі також активно досліджуються підходи до побудови верифікованих логічних годинників та масштабованих механізмів причинно-наслідкового впорядкування для децентралізованих середовищ і peer-to-peer систем. Зокрема, у роботах, присвячених системі Chrono, запропоновано механізми верифікованого причинно-наслідкового порядку, орієнтовані на функціонування у відкритих мережах із потенційно недовіренними вузлами. Разом із тим такі підходи характеризуються значною складністю реалізації та високими накладними витратами, що обмежує можливість їх використання у високонавантажених корпоративних мікросервісних системах.

Поряд із механізмами логічного часу та локального впорядкування значного поширення у сучасних розподілених системах набули комбіновані підходи, у межах яких забезпечення коректності обробки подій досягається шляхом поєднання декількох механізмів координації. Такі підходи передбачають

використання одночасно засобів маршрутизації за ключем, локальних буферів, механізмів повторної доставки, журналювання станів та контролю ідемпотентності. Необхідність їх комбінування обумовлена тим, що жоден окремий механізм не здатний повністю усунути суперечність між вимогами до масштабованості системи та необхідністю підтримання строгого порядку виконання подій.

Для систематизації існуючих підходів доцільно подати узагальнену класифікацію методів забезпечення впорядкованості подій у розподілених системах.

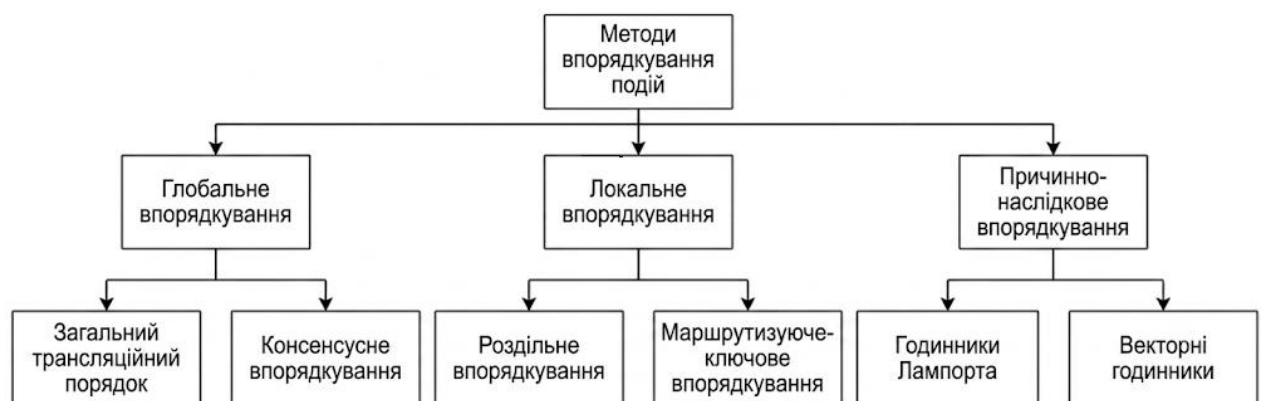


Рис. 1.7. Класифікація методів забезпечення впорядкованості подій

Як показано на рисунку 1.7, сучасні методи впорядкування можна умовно розділити на три основні групи: механізми глобального впорядкування, локального впорядкування та причинно-наслідкової координації. Методи глобального впорядкування забезпечують єдину послідовність подій для всієї системи, однак характеризуються значними накладними витратами та складністю масштабування. Натомість локальне впорядкування дозволяє підтримувати порядок лише в межах окремого логічного контексту, що суттєво підвищує продуктивність системи та знижує рівень глобальної синхронізації.

З точки зору високонавантажених мікросервісних систем найбільш практичними є саме локальні методи впорядкування, оскільки вони дозволяють забезпечити паралельну обробку незалежних потоків подій без необхідності підтримання глобального порядку для всієї системи. Проте навіть у цьому

випадку виникає проблема балансування між рівнем паралелізму та складністю координації потоків обробки.

З урахуванням проведеного аналізу особливого значення у сучасних високонавантажених мікросервісних системах набувають підходи, засновані на локальному впорядкуванні подій, оскільки саме вони дозволяють забезпечити баланс між рівнем паралелізму та складністю координації потоків обробки. На відміну від глобальних механізмів синхронізації, які потребують підтримання єдиного порядку для всієї системи, локальне впорядкування передбачає підтримання послідовності лише для подій, що належать одному логічному контексту або ключу маршрутизації.

Для узагальненого представлення принципу локального впорядкування подій доцільно розглянути відповідну структурну схему.

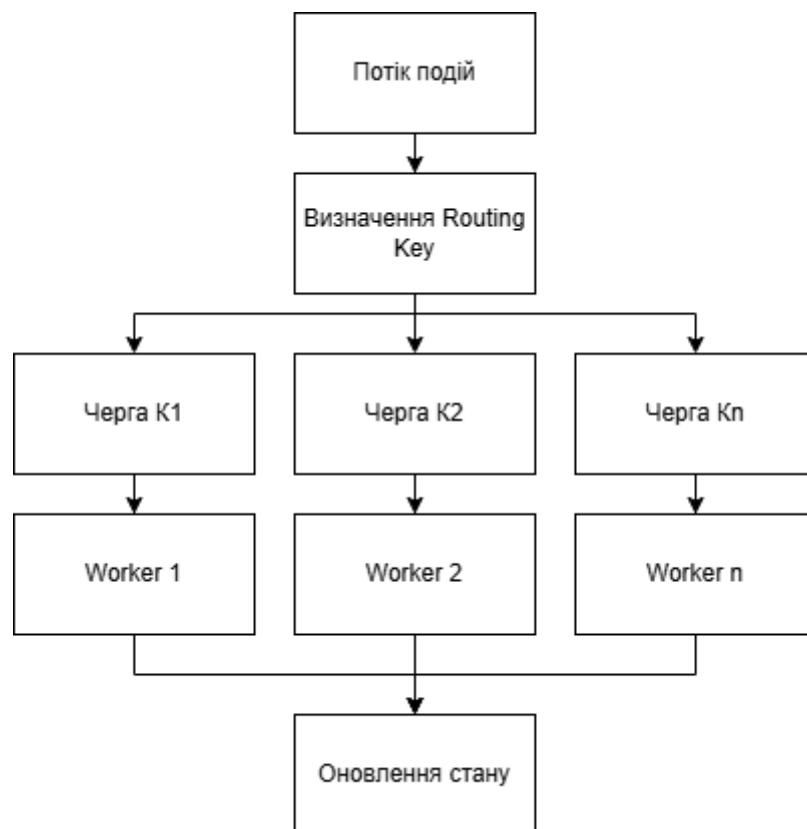


Рис. 1.8. Узагальнена схема локального впорядкування подій

Як показано на рисунку 1.8, потік подій розподіляється за ключами маршрутизації (*Routing Key*), після чого для кожного логічного ключа формується окремий потік обробки. У межах кожної локальної черги порядок подій

підтримується послідовно, тоді як незалежні ключі можуть оброблятися паралельно різними потоками виконання. Такий підхід дозволяє уникнути необхідності глобальної синхронізації між усіма компонентами системи та суттєво знизити накладні витрати на координацію процесів.

Важливою особливістю локального впорядкування є те, що воно дозволяє розглядати систему не як єдиний глобальний потік подій, а як множину незалежних логічних підпроцесів:

$$E = \bigcup_{k=1}^m E_k, \quad (1.16)$$

де  $E_k$  – множина подій, що належать ключу  $k$ .

У такому випадку умова впорядкованості забезпечується лише локально:

$$\forall e_i, e_j \in E_k : e_i \rightarrow e_j \quad (1.17)$$

тоді як між подіями різних множин допускається паралельне виконання:

$$E_i \parallel E_j, i \neq j. \quad (1.18)$$

Саме така модель є найбільш придатною для побудови масштабованих мікросервісних архітектур, оскільки дозволяє підтримувати коректність виконання бізнес-процесів без суттєвого зниження пропускної здатності системи.

Разом із тим ефективність локального впорядкування значною мірою залежить від способу реалізації механізмів маршрутизації, буферизації та координації потоків обробки. У практичних системах для цього використовуються різні підходи, які відрізняються рівнем складності, накладними витратами та ступенем підтримання причинно-наслідкової узгодженості.

Для систематизації особливостей існуючих методів впорядкування доцільно провести їх порівняльний аналіз.

З аналізу таблиці 1.7 видно, що глобальні механізми впорядкування забезпечують найвищий рівень узгодженості, однак супроводжуються значними

накладними витратами на координацію та погано масштабуються у високонавантажених системах. Натомість локальні підходи, засновані на розподілі потоків подій за ключами маршрутизації, дозволяють підтримувати високий рівень паралелізму та забезпечують прийнятний баланс між продуктивністю і коректністю обробки.

Таблиця 1.7

Порівняльна характеристика методів забезпечення впорядкованості подій

Метод	Тип впорядкування	Масштабованість	Накладні витрати	Основні обмеження
Логічні годинники Лампорта	часткове	висока	низькі	не визначає конкурентність
Векторні годинники	причинно-наслідкове	середня	високі	зростання розмірності
Total Order Broadcast	глобальне	низька	дуже високі	централізована координація
Partition Ordering	локальне	висока	помірні	локальність порядку
Routing-Key Ordering	локальне	висока	помірні	залежність від ключа

Разом із тим існуючі методи локального впорядкування не усувають повністю проблему порушення послідовності у випадках повторної доставки повідомлень, перезапуску сервісів або конкурентної обробки декількома споживачами. Крім того, більшість сучасних брокерів повідомлень забезпечують порядок лише в межах окремої черги або partition, тоді як підтримання узгодженості між незалежними потоками фактично переноситься на прикладний рівень системи.

У результаті сучасні підходи до впорядкування подій або характеризуються високими накладними витратами через необхідність глобальної синхронізації, або забезпечують лише частковий контроль послідовності виконання операцій. Це свідчить про актуальність розроблення методів, здатних поєднати переваги локального впорядкування, асинхронної доставки та паралельної обробки

повідомлень із механізмами забезпечення коректності функціонування системи в умовах часткових збоїв та повторної доставки подій.

Окремого поширення у сучасних мікросервісних архітектурах набули підходи, засновані на використанні *патерну Saga*, у межах якого глобальний бізнес-процес декомпозується на послідовність локальних транзакцій, що виконуються незалежними сервісами та координуються через події. На відміну від класичних розподілених транзакцій, які базуються на жорсткій синхронізації та централізованому управлінні станами, механізм *Saga* орієнтований на асинхронну координацію процесів та використання компенсаційних операцій у випадку виникнення часткових збоїв.

У загальному випадку модель *Saga* може бути подана у вигляді послідовності локальних транзакцій:

$$S = T_1, T_2, \dots, T_n, \quad (1.19)$$

де кожна транзакція  $T_i$  змінює локальний стан окремого сервісу та генерує подію для ініціації наступного етапу виконання бізнес-процесу.

У випадку виникнення помилки виконується компенсаційна операція:

$$C_i(T_i) \quad (1.20)$$

яка призначена для скасування результатів виконання відповідної локальної транзакції.

Для наочного представлення принципу функціонування *патерну Saga* доцільно розглянути відповідну схему координації розподілених операцій.

Як показано на рисунку 1.9, виконання розподіленого бізнес-процесу здійснюється як послідовність незалежних локальних транзакцій, взаємодія між якими реалізується через події. У випадку успішного завершення кожна транзакція ініціює виконання наступного етапу процесу, тоді як при виникненні помилки система переходить до виконання компенсаційних операцій, спрямованих на відновлення узгодженого стану.

Використання *патерну Saga* дозволяє суттєво підвищити масштабованість та відмовостійкість мікросервісної системи за рахунок відмови від централізованого блокування ресурсів. Разом із тим асинхронний характер координації транзакцій створює додаткові вимоги до забезпечення коректного



порядку обробки подій, оскільки порушення послідовності виконання локальних операцій або повторна доставка повідомлень можуть призводити до виникнення неузгоджених станів системи.

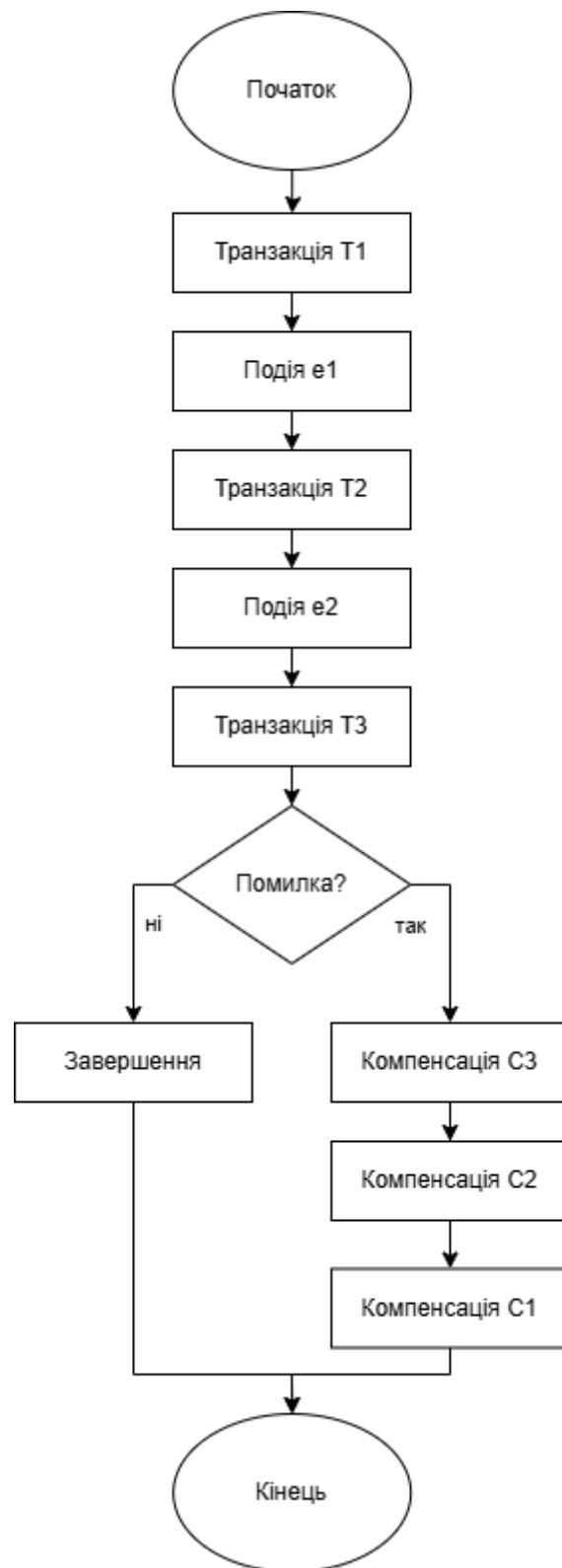


Рис. 1.9. Узагальнена схема реалізації патерну Saga у мікросервісній системі

Особливо критично зазначена проблема проявляється у випадках конкурентного виконання декількох бізнес-процесів, коли різні потоки подій можуть взаємодіяти із спільними ресурсами або змінювати один і той самий логічний стан системи. У таких умовах забезпечення коректності функціонування мікросервісної архітектури вимагає використання спеціалізованих механізмів локального впорядкування, контролю послідовності виконання подій та підтримання причинно-наслідкових залежностей між етапами розподіленого процесу.

Аналіз існуючих методів забезпечення впорядкованості подій показав, що підтримання коректної послідовності виконання операцій у розподіленому середовищі є складною багатофакторною задачею, ефективність вирішення якої безпосередньо залежить від особливостей транспортного середовища, рівня паралелізму системи та механізмів координації потоків обробки. При цьому навіть використання локального впорядкування та причинно-наслідкових механізмів синхронізації не усуває повністю проблему повторної доставки повідомлень, конкурентного виконання операцій та виникнення неузгоджених станів у випадках часткових збоїв або асинхронних затримок передачі подій.

Особливої актуальності зазначені проблеми набувають у системах, що використовують семантику *at-least-once*, оскільки повторна доставка повідомлень є невід'ємною характеристикою такого підходу до організації взаємодії між компонентами системи. У результаті забезпечення правильного порядку обробки подій повинно поєднуватися із механізмами контролю повторного виконання операцій та підтримання узгодженого стану системи навіть у випадку дублювання повідомлень або часткового порушення послідовності доставки.

У зв'язку з цим у сучасних подієво-орієнтованих мікросервісних архітектурах особливого значення набувають механізми забезпечення узгодженості станів та ідемпотентності операцій, які дозволяють гарантувати коректність функціонування системи за умов асинхронної взаємодії, повторної доставки подій та конкурентної обробки повідомлень.

## 1.4. Механізми забезпечення узгодженості та ідемпотентності

Однією з фундаментальних проблем функціонування сучасних подієво-орієнтованих мікросервісних систем є забезпечення узгодженості станів у середовищі асинхронної взаємодії та паралельної обробки подій. На відміну від централізованих монолітних систем, у яких зміна стану виконується в межах єдиного процесу та контролюється централізованим механізмом транзакційності, у розподілених архітектурах кожен сервіс підтримує власний локальний стан та виконує операції незалежно від інших компонентів системи. У результаті будь-які затримки доставки повідомлень, повторна передача подій або порушення порядку їх обробки можуть призводити до виникнення неузгоджених станів системи.

Проблема узгодженості особливо загострюється у системах, що використовують асинхронну модель взаємодії та семантику доставки *at-least-once*, оскільки в таких умовах одна й та сама подія потенційно може бути доставлена та оброблена декілька разів. За відсутності спеціалізованих механізмів контролю повторне виконання операції здатне призводити до дублювання транзакцій, некоректної зміни станів або накопичення помилкових даних у системі.

З теоретичної точки зору система вважається узгодженою, якщо після завершення обробки множини подій усі компоненти системи переходять до логічно несуперечливого стану:

$$S = s_1, s_2, \dots, s_n, \quad (1.21)$$

де  $s_i$  – локальний стан окремого сервісу.

Умову узгодженості можна подати у вигляді:

$$\forall s_i, s_j \in S : \text{Consistent}(s_i, s_j) = \text{true}, \quad (1.22)$$

де функція *Consistent* визначає відсутність суперечностей між локальними станами компонентів системи.

Разом із тим у середовищі асинхронної доставки досягнення абсолютної миттєвої узгодженості є практично неможливим через фундаментальні

обмеження розподілених систем та необхідність забезпечення високої доступності сервісів. Саме тому у сучасних мікросервісних архітектурах широкого поширення набув підхід *eventual consistency*, відповідно до якого система може тимчасово перебувати у неузгодженому стані, однак за відсутності нових подій усі компоненти з часом переходять до узгодженого стану.

Концепція *eventual consistency* є однією з базових моделей забезпечення узгодженості у сучасних розподілених системах, оскільки дозволяє поєднати високий рівень доступності сервісів із можливістю асинхронної обробки подій та незалежного масштабування компонентів системи. На відміну від моделей жорсткої синхронізації, у межах яких зміна стану повинна бути одночасно підтверджена всіма учасниками транзакції, модель поступової узгодженості допускає тимчасове розходження локальних станів окремих сервісів за умови їх подальшої синхронізації.

Для наочного представлення процесу досягнення поступової узгодженості доцільно розглянути відповідну часову діаграму.

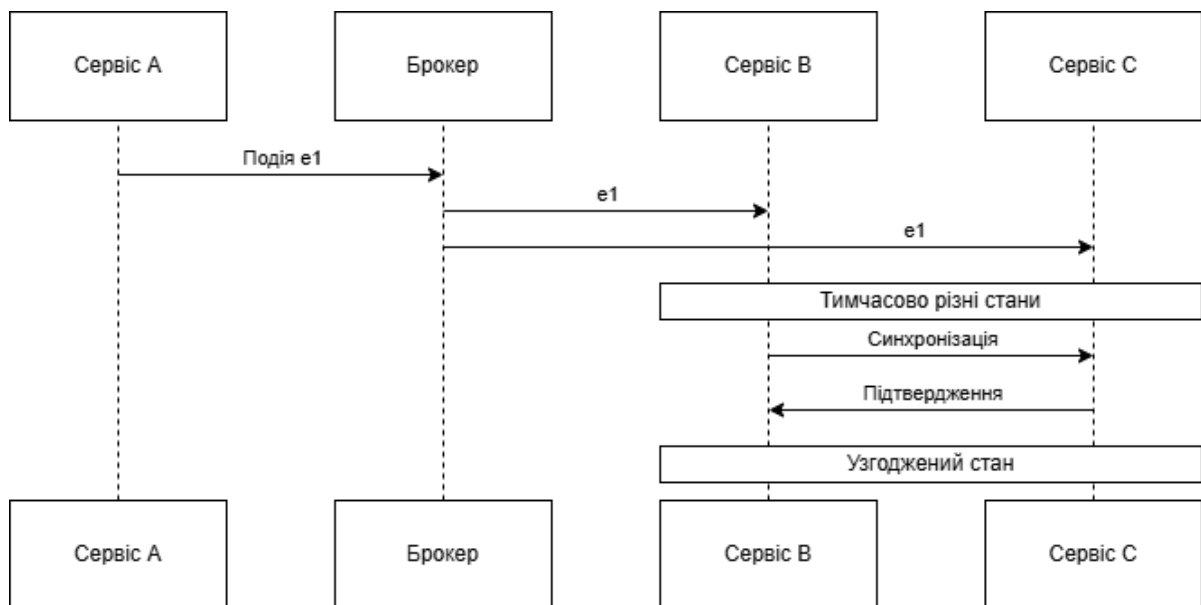


Рис. 1.10. Процес досягнення поступової узгодженості в асинхронній системі

Як показано на рисунку 1.10, після генерації події окремі сервіси можуть тимчасово перебувати у різних локальних станах через затримки доставки або асинхронний характер обробки повідомлень. Проте після завершення

синхронізації та обробки всіх подій система переходить до узгодженого стану. Такий підхід дозволяє уникнути централізованого блокування ресурсів та забезпечити високий рівень масштабованості мікросервісної архітектури.

Разом із тим використання поступової узгодженості створює необхідність впровадження механізмів контролю повторної обробки повідомлень, оскільки повторна доставка подій у системах із семантикою *at-least-once* є типовою ситуацією. У зв'язку з цим одним із ключових механізмів забезпечення коректності функціонування сучасних подієво-орієнтованих систем є ідемпотентність операцій.

У загальному випадку операція вважається ідемпотентною, якщо її повторне виконання не змінює кінцевий результат системи:

$$f(f(x)) = f(x). \quad (1.23)$$



Рис. 1.11. Узагальнена схема ідемпотентної обробки подій

У контексті мікросервісних систем це означає, що повторна обробка однієї й тієї ж події не повинна призводити до повторної зміни стану або дублювання результатів виконання операції.

Для ілюстрації принципу ідемпотентної обробки подій доцільно розглянути узагальнену схему контролю повторної доставки повідомлень.

Як показано на рисунку 1.11, перед виконанням операції система перевіряє наявність ідентифікатора події у журналі вже оброблених повідомлень. Якщо подія виконується вперше, відбувається зміна стану системи та збереження ідентифікатора події. У випадку повторної доставки повідомлення система ігнорує дубльовану операцію, що дозволяє уникнути некоректної зміни стану.

У більшості сучасних систем ідемпотентність реалізується через використання унікальних ідентифікаторів подій:

$$ID(e_i) = \text{hash}(\text{payload}_i, \text{timestamp}_i), \quad (1.24)$$

де  $\text{payload}_i$  – вміст події, а  $\text{timestamp}_i$  – момент її генерації.

У такому випадку множину вже виконаних операцій можна подати як:

$$\text{Processed} = ID(e_1), ID(e_2), \dots, ID(e_n). \quad (1.25)$$

Тоді умова ідемпотентної обробки визначається співвідношенням:

$$ID(e_i) \in \text{Processed} \Rightarrow \text{Skip}(e_i), \quad (1.26)$$

де  $\text{Skip}(e_i)$  означає пропуск повторного виконання події.

Для систематизації основних механізмів забезпечення узгодженості та ідемпотентності доцільно провести їх порівняльний аналіз.

Як видно з таблиці 1.8, більшість сучасних механізмів забезпечення узгодженості орієнтована не на повне усунення асинхронності, а на контроль її наслідків та підтримання коректності функціонування системи в умовах повторної доставки подій, конкурентної обробки та часткових збоїв. При цьому жоден із механізмів не є універсальним, а їх практичне використання зазвичай передбачає комбінування декількох підходів одночасно.

Основні механізми забезпечення узгодженості та ідемпотентності

Механізм	Призначення	Переваги	Основні обмеження
Eventual Consistency	поступова синхронізація станів	висока масштабованість	тимчасова неузгодженість
Ідемпотентні операції	захист від дублювання	коректність повторної доставки	необхідність журналювання
Distributed Locking	синхронізація доступу	строгий контроль стану	зниження продуктивності
Saga-pattern	координація транзакцій	відмовостійкість	складність компенсації
Event Sourcing	відновлення станів	повна історія подій	зростання обсягу журналів

Особливого значення у високонавантажених системах набуває проблема вибору балансу між рівнем узгодженості та продуктивністю системи. Використання жорстких механізмів синхронізації дозволяє підвищити рівень контролю над послідовністю зміни станів, однак супроводжується суттєвим зростанням латентності та зниженням рівня паралелізму. Натомість асинхронні механізми поступової узгодженості забезпечують високу масштабованість, але потребують додаткових засобів контролю повторного виконання операцій та локального впорядкування подій.

У результаті сучасні мікросервісні архітектури формують складне середовище взаємодії між механізмами асинхронної доставки, локального впорядкування, координації транзакцій та ідемпотентної обробки подій. Саме тому ефективне забезпечення коректності функціонування розподілених систем потребує побудови комплексних методів, здатних одночасно враховувати особливості повторної доставки повідомлень, причинно-наслідкових

залежностей між подіями та вимог до масштабованості сучасних високонавантажених інформаційних платформ.

Подальший розвиток механізмів забезпечення узгодженості у розподілених системах безпосередньо пов'язаний із необхідністю підтримання коректності функціонування системи в умовах часткових відмов окремих компонентів, асинхронних затримок доставки повідомлень та конкурентної зміни спільних даних декількома сервісами одночасно. У таких умовах проблема забезпечення узгодженості перестає бути виключно задачею синхронізації станів і набуває характеру комплексної задачі координації паралельних процесів, у межах якої необхідно враховувати не лише порядок виконання операцій, але й можливість відновлення після збоїв, повторної доставки повідомлень та реконструкції логічного стану системи.

Одним із найбільш поширених підходів до вирішення зазначеної проблеми є використання моделі *Event Sourcing*, відповідно до якої поточний стан системи не зберігається безпосередньо, а формується шляхом послідовного відтворення історії подій, що відбувалися у системі. У межах такого підходу кожна зміна стану розглядається як окрема подія:

$$e_i = (ID_i, t_i, payload_i), \quad (1.24)$$

де  $ID_i$  – унікальний ідентифікатор події,  $t_i$  – час її генерації, а  $payload_i$  – інформація про зміну стану.

У такому випадку поточний стан системи визначається як результат послідовного застосування множини подій:

$$S_n = f(e_1, e_2, \dots, e_n). \quad (1.25)$$

Основною перевагою моделі *Event Sourcing* є можливість реконструкції стану системи після відмови або втрати частини локальних даних, оскільки весь логічний процес функціонування системи представлений у вигляді журналу подій. Крім того, такий підхід забезпечує високий рівень аудитопритатності та дозволяє аналізувати історію змін стану системи.

Для наочного представлення принципу функціонування моделі *Event Sourcing* доцільно розглянути відповідну схему.





Рис. 1.12. Узагальнена схема реалізації Event Sourcing

Як показано на рисунку 1.12, усі події послідовно накопичуються у журналі, після чого поточний стан системи формується шляхом їх повторного відтворення. Це дозволяє забезпечити відновлення логічного стану навіть після часткових відмов компонентів системи або втрати локального контексту окремих сервісів.

Разом із тим використання *Event Sourcing* створює додаткові вимоги до механізмів впорядкування та ідемпотентної обробки подій, оскільки порушення послідовності відтворення журналу або повторне виконання окремих подій можуть призводити до формування некоректного стану системи. У зв'язку з цим у сучасних реалізаціях *Event Sourcing* широко використовуються механізми версіонування станів, контролю порядку виконання подій та локального впорядкування потоків обробки.

Окремої уваги потребують механізми блокування та координації конкурентного доступу до спільних ресурсів. У централізованих системах забезпечення узгодженості зазвичай реалізується через використання транзакційних блокувань, однак у мікросервісних архітектурах застосування глобального блокування є малоефективним через суттєве зниження рівня паралелізму та масштабованості системи.

У зв'язку з цим у сучасних розподілених системах більшого поширення набули механізми оптимістичної синхронізації, відповідно до яких система допускає паралельне виконання операцій із подальшою перевіркою коректності зміни стану. У загальному випадку умова успішного оновлення стану може бути подана у вигляді:

$$Version(S_i) = Version_{expected}, \quad (1.26)$$

де  $Version(S_i)$  – поточна версія стану системи.

У випадку невідповідності версій система виконує повторне зчитування стану або ініціює механізм компенсації конфлікту. Такий підхід дозволяє уникнути глобального блокування ресурсів та забезпечити високий рівень паралельної обробки подій.

Для порівняння механізмів синхронізації доцільно подати їх узагальнену характеристику.

Таблиця 1.9

Порівняльна характеристика механізмів синхронізації станів

Механізм	Принцип роботи	Переваги	Недоліки
Песимістичне блокування	блокування ресурсу	висока узгодженість	низький паралелізм
Оптимістична синхронізація	перевірка версій	висока масштабованість	можливість конфліктів
Event Sourcing	журналювання подій	відновлення станів	складність replay
CQRS	розділення читання/запису	продуктивність	складність координації

З аналізу таблиці 1.9 видно, що сучасні механізми забезпечення узгодженості орієнтовані переважно на мінімізацію глобальної синхронізації та підтримання високого рівня масштабованості системи. Водночас зменшення рівня централізованого контролю неминуче призводить до зростання складності механізмів координації подій, підтримання локального порядку обробки та забезпечення ідемпотентності операцій.

Особливо важливою проблемою у високонавантажених мікросервісних системах є забезпечення узгодженості у випадках каскадних відмов або втрати частини повідомлень транспортним середовищем. У таких умовах система повинна не лише підтримувати можливість повторної доставки подій, але й забезпечувати коректне відновлення логічного стану після повторного запуску сервісів або реконфігурації потоків обробки. Саме тому сучасні підходи до забезпечення узгодженості дедалі частіше комбінують механізми журналювання

подій, локального впорядкування, replay-відновлення та ідемпотентної обробки повідомлень у межах єдиної системи координації станів.

У результаті забезпечення узгодженості та ідемпотентності у сучасних мікросервісних архітектурах являє собою комплексну багаторівневу задачу, вирішення якої потребує одночасного врахування особливостей асинхронної доставки повідомлень, причинно-наслідкових залежностей між подіями, механізмів локального впорядкування та моделей відновлення станів після часткових збоїв системи.

Таким чином, забезпечення узгодженості та ідемпотентності у сучасних подієво-орієнтованих мікросервісних системах являє собою комплексну багаторівневу задачу, що охоплює механізми локального впорядкування подій, контролю повторної доставки повідомлень, replay-відновлення, журналювання станів та координації розподілених транзакцій. Проведений аналіз показав, що жоден із існуючих механізмів не забезпечує повного усунення проблем асинхронної взаємодії, а ефективне підтримання коректності функціонування системи можливе лише за умови інтеграції декількох взаємодоповнюючих підходів. Це створює теоретичні передумови для розроблення методу упорядкованого паралелізму, орієнтованого на забезпечення локальної послідовності подій, підтримання узгодженості станів та адаптацію до умов повторної доставки повідомлень у високонавантажених мікросервісних архітектурах.

## **1.5. Підходи до підвищення відмовостійкості та безперервності функціонування**

Однією з ключових вимог до сучасних подієво-орієнтованих мікросервісних систем є забезпечення відмовостійкості та безперервності функціонування в умовах високого рівня паралелізму, асинхронної взаємодії та динамічної зміни навантаження. На відміну від централізованих монолітних архітектур, у яких відмова окремого компонента зазвичай призводить до повного припинення функціонування системи, мікросервісні середовища

характеризуються значною кількістю незалежних сервісів, що взаємодіють через транспортне середовище передачі повідомлень. У результаті відмова одного компонента не обов'язково призводить до повної недоступності системи, однак може спричиняти порушення логічної послідовності виконання операцій, накопичення необроблених подій, втрату частини локального контексту або виникнення каскадних відмов між взаємозалежними сервісами.

У сучасній літературі відмовостійкість розглядається як здатність системи підтримувати коректне функціонування або переходити до контрольованого деградованого режиму роботи навіть у випадках часткових збоїв окремих компонентів, транспортного середовища чи інфраструктури виконання. При цьому безперервність функціонування визначає здатність системи зберігати доступність критичних бізнес-процесів протягом усього життєвого циклу обробки подій незалежно від виникнення локальних помилок або тимчасових відмов окремих сервісів.

З теоретичної точки зору рівень відмовостійкості системи можна подати через імовірність збереження працездатності після виникнення часткової відмови:

$$R(t) = P(T > t), \quad (1.27)$$

де  $R(t)$  – функція надійності системи, а  $T$  – випадкова величина часу безвідмовного функціонування.

Для складних розподілених систем загальний рівень надійності значною мірою залежить від архітектури взаємодії компонентів. У випадку послідовної залежності сервісів загальна надійність визначається співвідношенням:

$$R_{sys} = \prod_{i=1}^n R_i, \quad (1.28)$$

де  $R_i$  – надійність окремого компонента системи.

Наведене співвідношення демонструє, що зі збільшенням кількості взаємозалежних сервісів загальна ймовірність безвідмовного функціонування системи знижується, що є однією з фундаментальних проблем високонавантажених мікросервісних архітектур.

Для наочного представлення впливу часткової відмови на функціонування системи доцільно розглянути узагальнену архітектурну модель взаємодії компонентів.

Як показано на рисунку 1.16, у випадку відмови одного з сервісів транспортне середовище продовжує накопичення повідомлень у буфері повторної доставки, що дозволяє уникнути втрати подій та забезпечити можливість подальшого відновлення логічного стану системи після відновлення працездатності компонента. Саме використання асинхронного транспортного середовища є однією з ключових передумов підвищення живучості сучасних мікросервісних архітектур.



Рис. 1.16. Вплив часткової відмови сервісу на подієво-орієнтовану систему

Разом із тим використання асинхронної доставки повідомлень створює ризик накопичення застарілих подій, повторної доставки повідомлень та порушення причинно-наслідкової послідовності після відновлення сервісу. У зв'язку з цим сучасні системи забезпечення відмовостійкості поєднують механізми буферизації, replay-відновлення, локального впорядкування та ідемпотентної обробки повідомлень.

Одним із найбільш поширених підходів до підвищення відмовостійкості у мікросервісних системах є використання механізму *Circuit Breaker*, основне призначення якого полягає у запобіганні каскадному поширенню відмов між сервісами. У межах такого підходу система контролює кількість помилок взаємодії із зовнішнім компонентом та тимчасово блокує подальші запити у випадку перевищення допустимого порогу помилок.

У загальному випадку стан *Circuit Breaker* може бути поданий множиною:

$$CB = Closed, Open, HalfOpen, \quad (1.29)$$

де *Closed* – нормальний режим роботи; *Open* – режим блокування запитів; *HalfOpen* – режим тестового відновлення.

Для ілюстрації життєвого циклу *Circuit Breaker* доцільно розглянути відповідну діаграму станів.

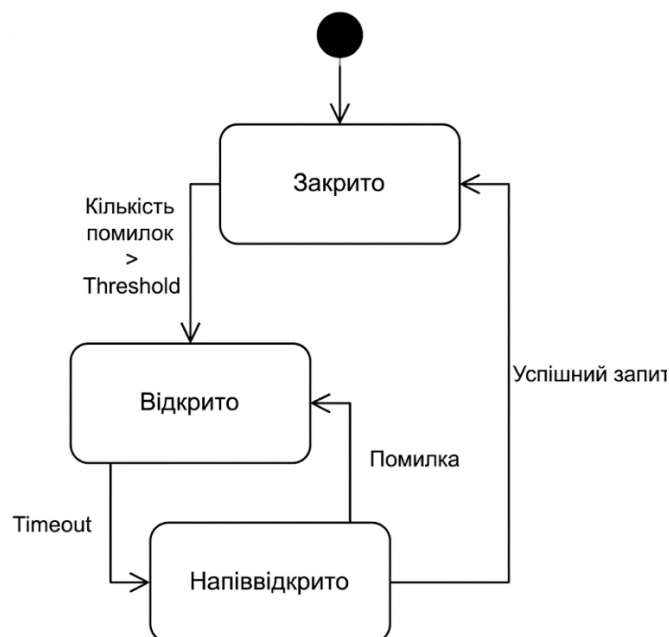


Рис. 1.17. Діаграма станів механізму *Circuit Breaker*

Як показано на рисунку 1.17, після перевищення допустимого рівня помилок система переходить у режим блокування запитів, що дозволяє локалізувати проблему та запобігти перевантаженню суміжних компонентів. Після завершення інтервалу очікування виконується тестова взаємодія із сервісом, за результатами якої система або повертається до нормального режиму роботи, або повторно переходить у стан блокування.

У сучасних високонавантажених системах значного поширення набули також механізми автоматичного *replay*-відновлення та повторної маршрутизації повідомлень. У випадку втрати частини подій або аварійного завершення роботи сервісу система формує процедуру повторного відтворення журналу подій:

$$Replay(E_k) = e_i, e_{i+1}, \dots, e_j, \quad (1.30)$$

що дозволяє реконструювати пропущений стан логічного процесу без необхідності глобального перезапуску всієї системи.

Разом із тим ефективність *replay*-відновлення безпосередньо залежить від наявності механізмів локального впорядкування та ідемпотентної обробки, оскільки повторне виконання невпорядкованих або дубльованих подій може призводити до формування некоректного стану системи. Саме тому сучасні підходи до забезпечення відмовостійкості дедалі частіше орієнтовані не на усунення асинхронності, а на контроль її наслідків та забезпечення можливості контрольованого відновлення після часткових збоїв.

Подальший розвиток підходів до підвищення відмовостійкості сучасних мікросервісних систем безпосередньо пов'язаний із дослідженнями механізмів локалізації відмов та підтримання живучості системи в умовах деградації окремих компонентів. У роботах, присвячених аналізу розподілених систем, зазначається, що однією з основних причин втрати працездатності високонавантажених платформ є не сама відмова окремого сервісу, а каскадне поширення помилок між взаємозалежними компонентами через механізми синхронного очікування відповіді або неконтрольоване накопичення повторних запитів [1, 2]. Саме тому сучасні архітектури дедалі частіше використовують асинхронні механізми координації та ізоляції потоків виконання, що дозволяє обмежити область поширення помилки в межах локального логічного контексту.

Одним із найбільш поширених підходів до локалізації відмов є використання механізму *Bulkhead Pattern*, основна ідея якого полягає у розділенні ресурсів системи на незалежні ізольовані області виконання. У межах такого підходу відмова або перевантаження одного потоку обробки не призводить до блокування інших компонентів системи, що дозволяє підтримувати працездатність критичних бізнес-процесів навіть у випадку деградації окремих сервісів [3].

Для наочного представлення принципу ізоляції потоків виконання доцільно розглянути відповідну архітектурну схему.

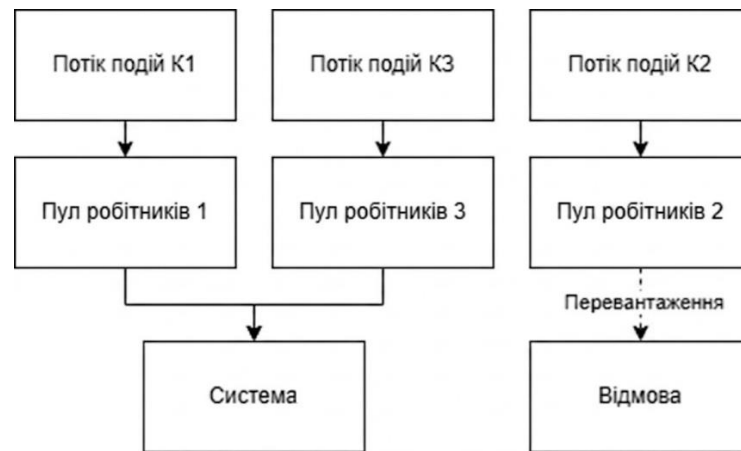


Рис. 1.18. Ізоляція потоків виконання за допомогою Bulkhead Pattern

Як показано на рисунку 1.18, кожен логічний потік обробки використовує окремий пул ресурсів, що дозволяє локалізувати вплив відмови або перевантаження в межах конкретного сегмента системи. У результаті навіть у випадку деградації одного потоку обробки інші сервіси продовжують функціонувати у штатному режимі, що суттєво підвищує живучість системи.

У сучасних подієво-орієнтованих системах механізми ізоляції потоків тісно пов'язані з концепцією локального впорядкування подій, оскільки розподіл повідомлень за ключами маршрутизації фактично формує незалежні області обробки, для яких можуть застосовуватися окремі ресурси, буфери та механізми replay-відновлення. Саме такий підхід дозволяє одночасно підтримувати високий рівень паралелізму та обмежувати область поширення відмов у межах окремих логічних процесів.



Окремої уваги потребують механізми забезпечення безперервності функціонування у випадках тимчасової недоступності транспортного середовища або брокера повідомлень. У сучасних мікросервісних архітектурах для цього широко використовуються резервні канали передачі даних та механізми автоматичного перемикавання маршрутів доставки повідомлень (*Failover Routing*) [4]. У випадку втрати зв'язку з основним брокером система тимчасово переходить до використання альтернативного каналу передачі повідомлень, що дозволяє уникнути повного блокування бізнес-процесу.

У загальному випадку процедура резервної маршрутизації може бути представлена як:

$$Route(e_i) = \begin{cases} MQ_{main}, & \text{якщо } Available(MQ_{main}) = true, \\ MQ_{backup}, & \text{інакше.} \end{cases} \quad (1.31)$$

Наведене співвідношення визначає правило вибору активного транспортного середовища залежно від стану доступності основного брокера повідомлень.

Для ілюстрації механізму резервної маршрутизації доцільно розглянути відповідну структурну схему.

Як показано на рисунку 1.19, у випадку втрати доступності основного брокера повідомлень система автоматично перенаправляє потік подій через резервний канал доставки. Такий підхід дозволяє підтримувати безперервність функціонування навіть у випадках часткових інфраструктурних збоїв або відмов транспортного середовища.

Разом із тим використання резервних каналів доставки створює додаткові вимоги до механізмів впорядкування та replay-відновлення, оскільки паралельне функціонування декількох транспортних маршрутів може призводити до дублювання повідомлень або порушення причинно-наслідкової послідовності подій. Саме тому резервна маршрутизація у сучасних системах зазвичай комбінується з механізмами ідемпотентної обробки та локального контролю порядку виконання операцій.



Рис. 1.19. Узагальнена схема резервної маршрутизації повідомлень

Для систематизації основних підходів до забезпечення відмовостійкості та безперервності функціонування доцільно провести їх порівняльний аналіз.

Таблиця 1.10

Основні підходи до підвищення відмовостійкості мікросервісних систем

Підхід	Призначення	Переваги	Основні обмеження
Circuit Breaker	локалізація помилок	запобігання каскадним відмовам	затримка відновлення
Bulkhead Pattern	ізоляція ресурсів	локалізація перевантаження	складність балансування
Replay-відновлення	реконструкція станів	відновлення після збоїв	додаткове навантаження
Failover Routing	резервна доставка	безперервність функціонування	ризик дублювання
Event Sourcing	журналювання подій	повне відновлення станів	збільшення обсягу даних

З аналізу таблиці 1.10 видно, що сучасні механізми забезпечення відмовостійкості орієнтовані не на повне усунення можливості виникнення збоїв, а на мінімізацію їх впливу на загальну працездатність системи та забезпечення можливості контрольованого відновлення після деградації окремих компонентів. При цьому більшість підходів базується на поєднанні асинхронної доставки повідомлень, локального впорядкування подій, replay-відновлення та ізоляції потоків виконання.

Важливою характеристикою сучасних мікросервісних систем є також живучість (*liveness*), яка визначає здатність системи продовжувати виконання бізнес-процесів навіть у випадку часткових відмов або тимчасового порушення доступності окремих компонентів [5]. На відміну від класичної надійності, що орієнтована переважно на відсутність помилок, властивість живучості передбачає підтримання прогресу виконання системи та недопущення повного блокування потоків обробки подій. Саме тому сучасні підходи до побудови високонавантажених подієво-орієнтованих систем дедалі більше орієнтуються на механізми деградованого, але безперервного функціонування, у межах якого система може тимчасово знижувати рівень продуктивності або точності синхронізації, однак продовжує підтримувати виконання критичних операцій.

Сучасні підходи до підвищення відмовостійкості та забезпечення безперервності функціонування подієво-орієнтованих мікросервісних систем базуються на поєднанні механізмів локалізації відмов, асинхронної доставки повідомлень, replay-відновлення, резервної маршрутизації та ізоляції потоків виконання. Проведений аналіз показав, що забезпечення живучості системи не може бути досягнуте виключно шляхом підвищення надійності окремих компонентів, оскільки ключову роль відіграє здатність системи підтримувати прогрес виконання бізнес-процесів в умовах часткових збоїв, повторної доставки повідомлень та деградації окремих сервісів. Саме тому ефективне забезпечення відмовостійкості сучасних мікросервісних архітектур потребує комплексного поєднання механізмів локального впорядкування подій, ідемпотентної обробки, replay-відновлення та адаптивного керування потоками асинхронної взаємодії.

Окремого значення у сучасних високонавантажених подієво-орієнтованих системах набувають механізми ізоляції помилкових повідомлень (*Dead Letter Queue*, DLQ), призначення яких полягає у локалізації подій, що не можуть бути коректно оброблені навіть після виконання повторних спроб доставки або replay-відновлення. Необхідність використання таких механізмів обумовлена тим, що в умовах асинхронної взаємодії окремі повідомлення можуть містити пошкоджені дані, порушувати логічну структуру бізнес-процесу або спричиняти нескінченні цикли повторної доставки, що потенційно створює ризик деградації всієї системи [6].

У загальному випадку множину помилкових повідомлень можна подати у вигляді:

$$DLQ = e_i | \text{Retry}(e_i) > R_{max}, \quad (1.32)$$

де  $\text{Retry}(e_i)$  – кількість спроб повторної обробки повідомлення;  $R_{max}$  – максимально допустима кількість повторних спроб.

Наведене співвідношення визначає умову переходу повідомлення до спеціалізованої черги помилкових подій після вичерпання механізмів replay-відновлення та повторної доставки.

Для ілюстрації ролі DLQ у системі забезпечення відмовостійкості доцільно розглянути узагальнену схему обробки повідомлень.

Як показано на рисунку 1.20, повідомлення, які не можуть бути коректно оброблені після визначеної кількості повторних спроб, ізолюються у спеціалізованій черзі помилкових повідомлень. Такий підхід дозволяє уникнути блокування основного потоку обробки та запобігти каскадному поширенню помилок між компонентами системи. Крім того, DLQ створює можливість подальшого аналізу причин виникнення помилки та повторного відновлення повідомлень після усунення джерела збою.

У сучасних системах забезпечення відмовостійкості дедалі більшого поширення набувають також підходи *Chaos Engineering*, у межах яких перевірка живучості та безперервності функціонування системи здійснюється шляхом контрольованого моделювання відмов окремих компонентів або транспортного середовища [7]. Основна ідея такого підходу полягає у тому, що система повинна

демонструвати здатність підтримувати виконання критичних бізнес-процесів навіть у випадку навмисного порушення функціонування окремих сервісів, каналів зв'язку або вузлів обробки подій.

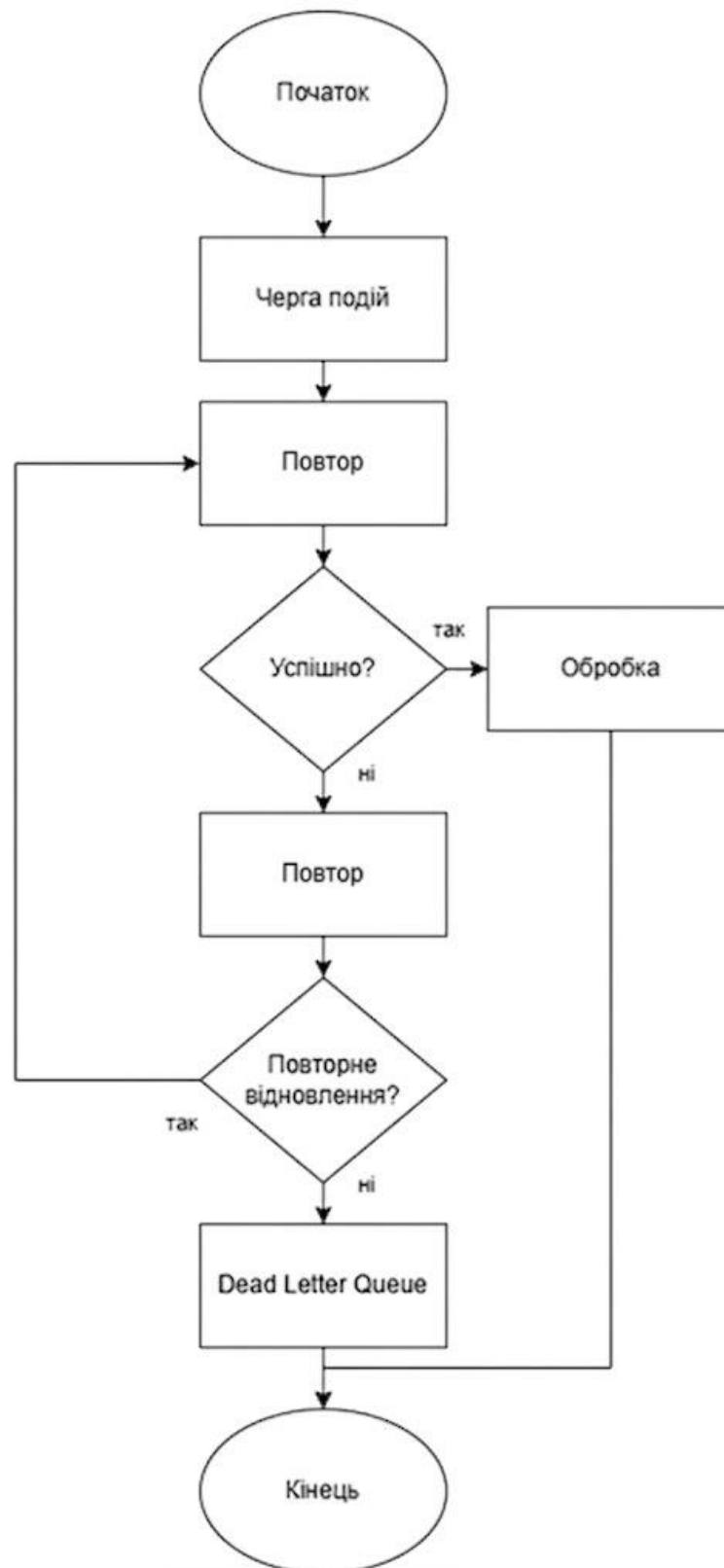


Рис. 1.20. Використання Dead Letter Queue у системі replay-відновлення

У контексті подієво-орієнтованих мікросервісних архітектур використання Chaos Engineering дозволяє оцінювати ефективність механізмів replay-відновлення, локального впорядкування, резервної маршрутизації та ідемпотентної обробки повідомлень в умовах реальних сценаріїв деградації системи.

Для узагальнення основних характеристик надійності сучасних розподілених систем доцільно подати їх порівняльну характеристику.

Таблиця 1.11

Основні характеристики функціональної стійкості розподілених систем

Властивість	Основна мета	Орієнтація
Reliability	мінімізація кількості відмов	стабільність функціонування
Availability	мінімізація часу простою	доступність сервісів
Liveness	підтримання прогресу виконання	безперервність бізнес-процесів

З аналізу таблиці 1.11 видно, що сучасні механізми забезпечення функціональної стійкості орієнтовані не лише на підвищення надійності окремих компонентів, але й на підтримання здатності системи продовжувати виконання бізнес-процесів навіть у випадках часткових збоїв або деградації окремих сервісів. Саме тому у сучасних подієво-орієнтованих мікросервісних архітектурах особливого значення набувають механізми replay-відновлення, локального впорядкування, резервної доставки повідомлень та ізоляції потоків виконання, які дозволяють підтримувати безперервність функціонування системи в умовах асинхронної взаємодії.

Проведений аналіз показав, що сучасні підходи до підвищення відмовостійкості та забезпечення безперервності функціонування базуються на комплексному поєднанні механізмів локалізації відмов, replay-відновлення, ідемпотентної обробки повідомлень, резервної маршрутизації та адаптивного керування потоками асинхронної взаємодії. Разом із тим навіть використання зазначених механізмів не забезпечує повного усунення проблем порушення послідовності обробки подій, накопичення дубльованих повідомлень та

деградації продуктивності системи в умовах високого рівня паралелізму. Особливо критично зазначені проблеми проявляються у високонавантажених мікросервісних архітектурах, де необхідність одночасного забезпечення масштабованості, локальної впорядкованості та живучості системи формує складну багатокритеріальну задачу побудови ефективних механізмів асинхронної доставки подій.

## **1.6. Аналіз обмежень існуючих рішень та постановка задачі дослідження**

Проведений аналіз сучасних підходів до організації асинхронної взаємодії, впорядкування подій, забезпечення узгодженості станів та підвищення відмовостійкості мікросервісних систем показав, що більшість існуючих рішень орієнтована на локальне усунення окремих проблем функціонування розподілених архітектур. Разом із тим комплексне забезпечення масштабованості, коректності обробки подій, підтримання локального порядку виконання операцій та безперервності функціонування системи в умовах асинхронної доставки повідомлень залишається невирішеною науково-прикладною задачею.

Сучасні брокери повідомлень, зокрема RabbitMQ, Apache Kafka та NATS, забезпечують підтримання порядку повідомлень лише в межах окремих черг або partition, однак не гарантують глобальної узгодженості у випадках паралельної обробки потоків подій декількома споживачами [4]. У результаті забезпечення коректної послідовності виконання бізнес-операцій фактично переноситься на прикладний рівень системи, що призводить до зростання складності реалізації мікросервісної архітектури та підвищення накладних витрат на координацію потоків обробки.

Крім того, більшість існуючих механізмів забезпечення впорядкованості або орієнтована на підтримання глобального порядку повідомлень, що супроводжується суттєвим зниженням рівня паралелізму та масштабованості системи, або використовує локальне впорядкування без достатнього рівня

контролю replay-відновлення та причинно-наслідкових залежностей між подіями. У результаті навіть за умови використання механізмів ідемпотентної обробки та replay-відновлення зберігається ризик виникнення неузгоджених станів у випадках повторної доставки повідомлень або порушення послідовності виконання залежних операцій.

Додатковою проблемою сучасних підходів до організації асинхронної взаємодії є відсутність ефективних механізмів адаптивного керування рівнем паралелізму залежно від поточного стану системи, інтенсивності потоку подій та кількості активних логічних процесів. У більшості існуючих реалізацій кількість потоків обробки, параметри буферизації та правила маршрутизації визначаються статично, що ускладнює підтримання стабільної продуктивності системи в умовах нерівномірного навантаження або динамічної зміни структури потоку подій. У результаті при збільшенні інтенсивності асинхронної взаємодії зростає ймовірність накопичення черг повідомлень, порушення локального порядку обробки та деградації латентності системи.

Особливо критично зазначені обмеження проявляються у високонавантажених мікросервісних системах, у яких один бізнес-процес може породжувати значну кількість взаємозалежних подій, що обробляються різними сервісами одночасно. У таких умовах навіть короткочасне порушення причинно-наслідкової послідовності здатне призводити до накопичення логічних конфліктів між станами окремих компонентів системи. При цьому використання глобальної синхронізації для усунення зазначених проблем супроводжується суттєвим зниженням масштабованості та збільшенням часу очікування виконання операцій, що суперечить вимогам сучасних високонавантажених інформаційних платформ.

Аналіз літературних джерел також показав, що значна частина існуючих механізмів replay-відновлення орієнтована переважно на реконструкцію втрачених станів, однак недостатньо враховує проблему повторної доставки повідомлень та підтримання коректного порядку повторного виконання подій. У випадку використання семантики доставки at least once повторне відтворення журналу подій без додаткових механізмів контролю послідовності може



призводити до дублювання бізнес-операцій або формування некоректного логічного стану системи. Саме тому ефективне replay-відновлення повинно поєднуватися з механізмами локального впорядкування, буферизації та ідемпотентної обробки повідомлень.

Крім того, більшість сучасних брокерів повідомлень не забезпечує достатнього рівня інтеграції між механізмами маршрутизації, контролю порядку обробки та системами забезпечення відмовостійкості. У результаті розробники змушені реалізовувати значну частину логіки координації на прикладному рівні, що призводить до ускладнення архітектури системи та збільшення кількості потенційних точок відмови. Особливо це стосується механізмів replay-відновлення, підтримання локального порядку подій та синхронізації розподілених транзакцій у середовищі асинхронної взаємодії.

Важливим недоліком більшості існуючих рішень є також відсутність комплексного підходу до забезпечення живучості системи в умовах часткових відмов та деградації окремих компонентів. Наявні механізми fault tolerance зазвичай орієнтовані або на локалізацію помилок, або на резервування транспортного середовища, однак недостатньо враховують взаємозв'язок між процесами повторної доставки повідомлень, replay-відновленням та підтриманням причинно-наслідкової узгодженості потоків подій. У результаті навіть за умови відновлення працездатності окремих сервісів система може переходити до стану логічної неузгодженості через порушення порядку повторного виконання подій.

Для узагальнення взаємозв'язку між основними проблемами сучасних подієво-орієнтованих мікросервісних систем доцільно розглянути інтегральну концептуальну схему, яка відображає вплив асинхронної взаємодії на процес забезпечення коректності функціонування.

Як показано на рисунку 1.21, асинхронна доставка повідомлень та паралельна обробка подій формують складне середовище взаємодії між механізмами replay-відновлення, підтримання локального порядку та забезпечення узгодженості станів системи.

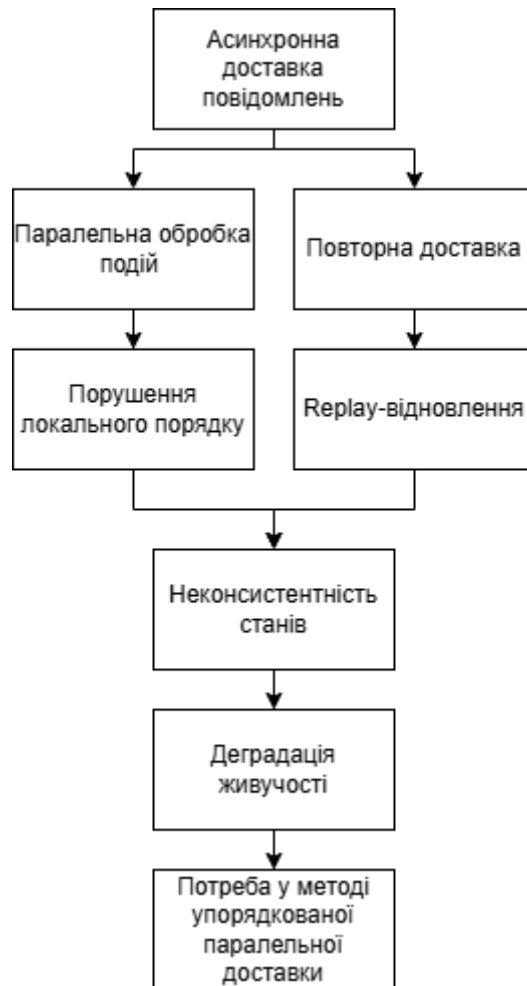


Рис. 1.21. Взаємозв'язок проблем асинхронної взаємодії у мікросервісних системах

Повторна доставка повідомлень та порушення причинно-наслідкової послідовності виконання операцій призводять до ризику виникнення неузгоджених станів, що, у свою чергу, негативно впливає на живучість та безперервність функціонування системи. Саме тому забезпечення коректності функціонування високонавантажених мікросервісних архітектур потребує побудови комплексних механізмів координації потоків асинхронної взаємодії.

Проведений аналіз дозволяє зробити висновок, що сучасні підходи до організації подієво-орієнтованих мікросервісних систем характеризуються наявністю фундаментальної суперечності між необхідністю забезпечення високого рівня паралелізму та потребою підтримання коректної послідовності виконання взаємозалежних операцій. Використання глобального впорядкування дозволяє підвищити рівень узгодженості системи, однак супроводжується

суттєвим зниженням масштабованості та продуктивності. Натомість локальні механізми паралельної обробки забезпечують високу пропускну здатність системи, але створюють ризик порушення причинно-наслідкових залежностей між подіями та формування некоректних станів у випадках повторної доставки повідомлень або часткових збоїв.

У зв'язку з цим актуальною науково-прикладною задачею є розроблення методу упорядкованої паралельної доставки подій, здатного забезпечити поєднання високого рівня паралелізму з підтриманням локальної послідовності виконання операцій у межах окремих логічних процесів. Такий метод повинен враховувати особливості асинхронної взаємодії мікросервісних компонентів, підтримувати механізми replay-відновлення та ідемпотентної обробки повідомлень, а також забезпечувати живучість системи в умовах повторної доставки подій, часткових відмов та деградації окремих компонентів транспортного середовища.

З урахуванням проведеного аналізу метою дисертаційного дослідження є підвищення стабільності та масштабованості мікросервісних систем шляхом розроблення методу упорядкованої паралельної доставки подій на основі механізмів локального впорядкування, replay-відновлення та адаптивної координації потоків асинхронної обробки повідомлень.

Для досягнення поставленої мети у роботі необхідно вирішити такі основні задачі дослідження:

- дослідити особливості функціонування сучасних подієво-орієнтованих мікросервісних систем та проаналізувати існуючі механізми забезпечення впорядкованості подій;

- проаналізувати сучасні підходи до реалізації replay-відновлення, ідемпотентної обробки повідомлень та забезпечення живучості розподілених систем;

розробити метод упорядкованої паралельної доставки подій, орієнтований на підтримання локальної послідовності виконання операцій при високому рівні паралелізму;

розробити механізми replay-відновлення та буферизації подій для підтримання

коректності функціонування системи у випадках часткових збоїв та повторної доставки повідомлень;

- провести дослідження властивостей коректності, живучості та масштабованості запропонованого методу;
- виконати експериментальне оцінювання ефективності запропонованого підходу та порівняти його з існуючими механізмами організації асинхронної взаємодії у мікросервісних системах.

Результати проведеного аналізу підтверджують актуальність задачі розроблення механізмів упорядкованої паралельної доставки подій та створюють теоретичне підґрунтя для побудови методу, здатного забезпечити поєднання масштабованості, локальної впорядкованості та відмовостійкості сучасних високонавантажених мікросервісних систем.

## **Висновки до розділу 1**

У першому розділі проведено комплексний аналіз сучасних підходів до організації подієво-орієнтованих мікросервісних систем, механізмів асинхронної доставки повідомлень, локального впорядкування подій, replay-відновлення, ідемпотентної обробки та забезпечення відмовостійкості розподілених архітектур. Досліджено особливості функціонування сучасних брокерів повідомлень, механізми підтримання причинно-наслідкової узгодженості та підходи до забезпечення живучості систем у випадках повторної доставки повідомлень і часткових відмов компонентів транспортного середовища.

Проведений аналіз показав, що більшість існуючих рішень орієнтована на локальне усунення окремих аспектів проблеми асинхронної взаємодії та не забезпечує комплексного поєднання масштабованості, локальної впорядкованості, replay-відновлення та підтримання коректності функціонування системи в умовах високого рівня паралелізму. Встановлено, що використання глобального впорядкування супроводжується суттєвим зниженням продуктивності системи, тоді як локальні механізми паралельної обробки створюють ризик порушення причинно-наслідкової послідовності виконання

операцій та виникнення неузгоджених станів у випадках повторної доставки повідомлень або replay-відновлення.

У результаті проведеного дослідження визначено основні обмеження сучасних підходів до організації асинхронної взаємодії та сформульовано науково-прикладну задачу розроблення методу упорядкованої паралельної доставки подій, орієнтованого на забезпечення локальної впорядкованості, replay-відновлення, ідемпотентної обробки та підтримання живучості високонавантажених мікросервісних систем в умовах асинхронної доставки повідомлень і часткових відмов компонентів системи.

## РОЗДІЛ 2. РОЗРОБКА МЕТОДУ УПОРЯДКОВАНОЇ ПАРАЛЕЛЬНОЇ ДОСТАВКИ ПОДІЙ У МІКРОСЕРВІСНИХ СИСТЕМАХ

### 2.1. Формалізація подієво-орієнтованої моделі системи

У сучасних розподілених інформаційних системах, побудованих на принципах мікросервісної архітектури, взаємодія між окремими компонентами здійснюється переважно за подієво-орієнтованою моделлю, яка передбачає асинхронний обмін повідомленнями та слабке зв'язування сервісів. Такий підхід забезпечує високий рівень масштабованості, гнучкість розгортання та адаптивність до змін навантаження, однак водночас породжує низку складних задач, пов'язаних із забезпеченням коректності обробки подій, їх впорядкованості та узгодженості станів системи.

Формалізація подієво-орієнтованої моделі системи передбачає введення абстрактного представлення процесу взаємодії між компонентами у вигляді потоків подій, які генеруються, передаються та обробляються у розподіленому середовищі. У рамках даного дослідження система розглядається як множина взаємодіючих сервісів, що обмінюються подіями через посередницький транспортний рівень, представлений брокером повідомлень або альтернативними каналами передачі.

Нехай множина подій системи визначається як  $E = \{e_i\}$ , де кожна подія є елементарною одиницею інформації, що відображає зміну стану певного бізнес-процесу. Для забезпечення формалізованого опису подія представляється у вигляді кортежу:

$$e = (k, s, t, p), \quad (2.1)$$

де  $k \in K$  – ключ впорядкування (ідентифікатор логічного контексту, наприклад, замовлення або транзакції),  $s \in \mathbb{N}$  – порядковий номер події у межах даного ключа,  $t$  – момент часу надходження події в систему,  $p$  – корисне навантаження події.

Введення ключа впорядкування  $k$  є принципово важливим, оскільки саме він визначає рівень локальної послідовності обробки подій, що відповідає окремому бізнес-процесу. Події з однаковим ключем формують впорядкований потік, тоді як події з різними ключами можуть оброблятися незалежно, що створює передумови для паралельного виконання.

У межах формалізованої моделі передбачається, що для кожного ключа  $k$  існує послідовність подій

$$E_k = \{e_{k,1}, e_{k,2}, \dots, e_{k,n}\}, \quad (2.2)$$

яка повинна оброблятися у строго визначеному порядку, що відповідає зростанню значення порядкового номера  $s$ . Така вимога обумовлена необхідністю збереження логічної узгодженості станів системи, оскільки порушення послідовності виконання операцій може призводити до некоректних результатів обробки.

З урахуванням асинхронного характеру взаємодії, доставка подій у системі здійснюється з використанням семантики *at-least-once*, яка гарантує доставку кожної події щонайменше один раз, роте допускає її повторне надходження або порушення порядку через різні затримки у каналах передачі. У таких умовах формалізована модель повинна враховувати можливість існування множини доставок для однієї події, а також потенційні розриви у послідовності подій.

З метою опису процесу обробки подій вводиться відображення стану системи

$$S: K \rightarrow \mathbb{N}, \quad (2.3)$$

де  $S(k)$  визначає номер останньої коректно обробленої події для ключа  $k$ . Обробка нової події  $e = (k, s, t, p)$  допускається лише за умови виконання співвідношення:

$$s = S(k) + 1, \quad (2.4)$$

що забезпечує строге дотримання порядку виконання. У випадку, коли

$$s > S(k) + 1, \quad (2.5)$$

подія вважається передчасною та підлягає тимчасовій буферизації до моменту отримання пропущених елементів послідовності. Якщо ж  $s \leq S(k)$ , подія

інтерпретується як дубль і не впливає на стан системи, що відповідає принципу ідемпотентності.

Особливістю подієво-орієнтованої моделі є також наявність проміжного транспортного середовища, яке здійснює маршрутизацію та буферизацію подій. При цьому гарантії впорядкованості обмежуються рамками окремих черг або каналів і не поширюються на глобальний порядок у системі, що зумовлює необхідність перенесення частини логіки впорядкування на прикладний рівень.

Подальший розвиток формалізованої моделі потребує врахування не лише структури подій та умов їх коректної обробки, але й характеристик середовища виконання, у якому здійснюється передача та обробка повідомлень. З цією метою до моделі вводиться множина обробників  $W = \{w_j\}$ , кожен з яких здатний виконувати обробку подій, що належать до різних ключів, за умови дотримання обмежень щодо впорядкованості. Кількість доступних обробників  $|W| = m$  визначає максимальний рівень паралелізму системи, тоді як ефективне використання цих ресурсів залежить від розподілу навантаження між потоками подій.

Для кожного ключа  $k \in K$  вводиться відповідна внутрішня черга  $Q_k$ , яка містить підмножину подій  $E_k$ , що надійшли до системи, але ще не були оброблені. Така декомпозиція дозволяє розглядати систему як сукупність незалежних підсистем обслуговування, кожна з яких відповідає за обробку подій певного логічного контексту. При цьому глобальна система функціонує як композиція локальних процесів, що взаємодіють через спільні обчислювальні ресурси.

Нехай інтенсивність надходження подій для ключа  $k$  визначається як  $\lambda_k$ , а середня швидкість обробки – як  $\mu_k$ . Тоді коефіцієнт завантаження відповідної черги визначається співвідношенням

$$\rho_k = \frac{\lambda_k}{\mu_k}. \quad (2.6)$$

Забезпечення стабільності системи вимагає виконання умови  $\rho_k < 1$  для кожного активного ключа, що означає перевищення обчислювальної спроможності над інтенсивністю надходження подій. У протилежному випадку



спостерігається накопичення подій у чергах, що призводить до зростання латентності та потенційного порушення вимог до часу обробки.

З урахуванням обмежених ресурсів система повинна динамічно керувати множиною активних черг  $\{Q_k\}$ , оскільки кількість можливих ключів може бути значно більшою за кількість доступних обробників. Це зумовлює необхідність введення політики планування, яка визначає порядок обслуговування черг та розподіл обчислювальних ресурсів між ними. Формально така задача може бути представлена як задача оптимізації, спрямована на мінімізацію середнього часу очікування подій при обмеженні на кількість одночасно обслуговуваних потоків.

Окрему увагу у формалізації моделі приділено опису механізму виникнення та обробки розривів у послідовності подій. У випадку, коли до черги  $Q_k$  надходить подія з номером  $s' > S(k) + 1$ , система фіксує наявність пропущеного елемента та ініціює процедуру очікування протягом інтервалу часу  $T_{gap}$ . Якщо протягом цього інтервалу відсутня подія з очікуваним номером, запускається механізм відновлення, який передбачає повторне запитування відсутнього елемента з джерела подій або з буферизованого сховища. Такий підхід дозволяє забезпечити відновлення коректної послідовності навіть у випадку втрат або затримок передачі повідомлень.

Важливим аспектом формалізації є також врахування імовірнісного характеру доставки подій. Нехай  $p_{loss}$  – імовірність втрати або недоставки події, а  $p_s$  – імовірність успішного відновлення пропущеної події під час однієї спроби. У цьому випадку ймовірність успішного відновлення після  $R$  спроб визначається як

$$P_{success} = 1 - (1 - p_s)^R, \quad (2.7)$$

що дозволяє оцінити надійність механізму відновлення та визначити оптимальні параметри його функціонування.

З позиції часових характеристик функціонування системи загальний час обробки події може бути представлений як сума базового часу обробки, часу очікування у черзі та додаткових витрат, пов'язаних із буферизацією та відновленням порядку. Таким чином, середній час перебування події у системі

визначається не лише параметрами обслуговування, але й імовірністю виникнення порушень послідовності та ефективністю механізмів їх усунення.

Крім того, для деталізації формалізованої моделі доцільно уточнити взаємозв'язки між компонентами системи, ввести додаткові змінні стану та розширити опис процесів генерації, передачі й обробки подій у часовому та логічному вимірах. З цією метою система розглядається як динамічна дискретна структура, стан якої змінюється у моменти надходження або завершення обробки подій, а також під час ініціювання процедур відновлення або повторної доставки.

Нехай  $T = \{t_i\}$  – множина моментів часу, у які відбуваються події системи, тоді функціонування системи може бути описане як відображення

$$\Phi : (E, T) \rightarrow S, \quad (2.8)$$

яке визначає зміну глобального стану системи  $S$  під впливом послідовності подій. При цьому глобальний стан є композицією локальних станів для кожного ключа  $k$ , тобто

$$S = \bigcup_{k \in K} S_k, \quad (2.9)$$

де  $S_k$  характеризує поточний стан обробки відповідного логічного процесу.

З урахуванням асинхронного характеру взаємодії необхідно розрізняти три основні етапи життєвого циклу події: генерацію, транспортування та обробку. На етапі генерації подія отримує унікальний ідентифікатор і порядковий номер  $s$ , що формується у межах відповідного ключа. На етапі транспортування подія передається через брокер повідомлень або альтернативний канал, при цьому можливі затримки, дублювання або порушення порядку доставки. На етапі обробки виконується перевірка коректності події та її вплив на стан системи.

Формально процес транспортування може бути описаний як стохастичне відображення

$$\Psi : E \rightarrow E', \quad (2.10)$$

де  $E'$  – множина подій, фактично доставлених до споживача. При цьому для кожної події  $e \in E$  може існувати множина доставок

$$\{e^{(1)}, e^{(2)}, \dots, e^{(m)}\}, \quad (2.11)$$

що відповідає семантиці *at-least-once* і потребує введення механізмів дедуплікації на рівні обробки.

З метою врахування впливу транспортного середовища до моделі вводиться функція затримки доставки

$$\tau : E \rightarrow \mathbb{R}^+, \quad (2.12)$$

яка визначає часовий інтервал між моментом генерації події та її фактичним отриманням споживачем. Нерівномірність значень  $\tau$  для різних подій одного ключа є основною причиною порушення їх природного порядку, що, у свою чергу, потребує застосування механізмів буферизації та відновлення послідовності.

Для формалізації процесу буферизації вводиться множина очікування

$$B_k(t) = \{e \in E_k \mid s > S(k) + 1\}, \quad (2.13)$$

яка містить події, що не можуть бути оброблені негайно через наявність розриву у послідовності. Розмір цієї множини безпосередньо впливає на використання пам'яті та латентність системи, а також залежить від інтенсивності надходження подій і характеристик транспортного середовища.

Додатково до цього доцільно ввести функцію індикатора розриву

$$\delta_k(s) = \begin{cases} 1, & \text{якщо } s > S(k) + 1, \\ 0, & \text{інакше,} \end{cases} \quad (2.14)$$

яка дозволяє формалізувати момент виникнення ситуації порушення порядку. При  $\delta_k(s) = 1$  система переходить у режим очікування або ініціює процедуру відновлення, що включає генерацію запитів на повторну доставку пропущених подій.

Важливою складовою формалізації є опис механізму прийняття рішень щодо обробки подій, який може бути представлений у вигляді функції

$$f(e, S(k)) = \begin{cases} \text{process, якщо } s = S(k) + 1, \\ \text{buffer, якщо } s > S(k) + 1, \\ \text{ignore, якщо } s \leq S(k), \end{cases} \quad (2.15)$$

що визначає подальшу долю кожної події залежно від її порядкового номера та поточного стану системи.

З урахуванням паралельної обробки доцільно також врахувати функцію розподілу обробників між чергами

$$\Pi: K \rightarrow W, \quad (2.16)$$

яка визначає, який саме обробник обслуговує події певного ключа у конкретний момент часу. Така функція може змінюватися динамічно залежно від навантаження, що дозволяє реалізувати адаптивне масштабування системи.

Окремим аспектом є врахування впливу «гарячих ключів», тобто таких значень  $k$ , для яких інтенсивність надходження подій  $\lambda_k$  значно перевищує середнє значення. У цьому випадку відповідна черга  $Q_k$  може стати вузьким місцем системи, що призводить до зростання затримок та нерівномірного використання ресурсів. Формально це явище може бути описане коефіцієнтом нерівномірності

$$\alpha = \frac{\max_k \lambda_k}{\left| \frac{1}{K} \sum_{k \in K} \lambda_k \right|}, \quad (2.17)$$

який характеризує ступінь дисбалансу навантаження у системі.

Запропоновану формалізацію доцільно здійснювати шляхом інтеграції описаних структурних та стохастичних характеристик у єдину узагальнену модель функціонування системи, яка дозволяє не лише якісно описувати поведінку потоків подій, але й забезпечує можливість кількісного аналізу її властивостей. З цією метою вводиться поняття траєкторії виконання подій для кожного ключа  $k$ , яка визначається як впорядкована у часі послідовність змін стану

$$\Gamma_k = \{S_k(t_0), S_k(t_1), \dots, S_k(t_n)\}, \quad (2.18)$$

де кожний перехід між станами обумовлений обробкою відповідної події. Така траєкторія є відображенням еволюції локального процесу та повинна відповідати строгій монотонності за порядковим номером подій, що формалізує вимогу детермінованості виконання.

З огляду на те, що система функціонує у розподіленому середовищі, де відсутній глобальний синхронізований час, доцільно вводити частковий порядок

подій, який визначається відношенням причинно-наслідкової залежності. Формально це може бути задано бінарним відношенням  $<$ , де

$$e_i < e_j \quad (2.19)$$

означає, що подія  $e_i$  повинна бути оброблена раніше, ніж  $e_j$ . У межах одного ключа це відношення є повним і визначається порядковими номерами, тоді як між подіями різних ключів воно є частковим, що дозволяє реалізувати паралельну обробку без порушення локальної узгодженості.

З метою узгодження локального порядку з глобальним паралелізмом до моделі вводиться функція узгодження

$$\Omega: E \rightarrow \{0,1\}, \quad (2.20)$$

яка визначає, чи може подія бути оброблена у поточний момент часу без порушення встановлених обмежень. Значення  $\Omega(e) = 1$  відповідає допустимості виконання, тоді як  $\Omega(e) = 0$  означає необхідність відкладення обробки. Така функція фактично реалізує механізм синхронізації у системі, не вдаючись до централізованого керування, що є принципово важливим для забезпечення масштабованості.

Розглядаючи систему з точки зору теорії масового обслуговування, можна інтерпретувати кожен чергу  $Q_k$  як незалежний канал обслуговування із власною дисципліною доступу, тоді як загальна система являє собою мережу взаємопов'язаних черг із обмеженими ресурсами. При цьому ефективність функціонування системи визначається балансом між інтенсивністю надходження подій, швидкістю їх обробки та накладними витратами, пов'язаними із забезпеченням впорядкованості.

Особливого значення набуває питання узгодження часових характеристик різних компонентів системи. Нехай  $T_{proc}$  – середній час обробки події,  $T_{wait}$  – середній час очікування у черзі,  $T_{buf}$  – додатковий час, пов'язаний із буферизацією, а  $T_{rec}$  – час, необхідний для відновлення пропущених подій. Тоді загальний час перебування події у системі може бути представлений як

$$T_{total} = T_{wait} + T_{proc} + T_{buf} + T_{rec} \quad (2.21)$$

що дозволяє формалізувати вплив кожного з механізмів на латентність системи та визначити ключові фактори її оптимізації.

У контексті забезпечення відмовостійкості доцільно враховувати можливість існування альтернативних каналів доставки подій, які можуть бути активовані у випадку деградації основного транспортного середовища. Формально це може бути описано як наявність множини каналів  $C = \{c_1, c_2, \dots, c_m\}$ , для яких визначається функція доступності

$$A: C \rightarrow \{0,1\} \quad (2.22)$$

що відображає поточний стан кожного каналу. Використання декількох каналів передачі дозволяє підвищити ймовірність успішної доставки подій та зменшити ризик втрати інформації, що є критичним для забезпечення властивості живучості системи.

Загалом, запропонована формалізація дозволяє розглядати подієво-орієнтовану систему як складну стохастичну динамічну систему з дискретним часом, у якій поєднуються процеси генерації, передачі, впорядкування та обробки подій. Такий підхід створює можливість для подальшого переходу до синтезу алгоритмічних рішень, що забезпечують гарантоване дотримання порядку обробки подій при збереженні високого рівня паралелізму.

Важливим елементом формалізації подієво-орієнтованої моделі є визначення інваріантів, які повинні зберігатися під час функціонування системи незалежно від конкретної реалізації механізмів доставки та обробки подій. До таких інваріантів, насамперед, належить умова збереження порядку обробки подій у межах одного ключа, яка може бути формалізована у вигляді вимоги:

$$\forall k \in K : \text{якщо } e_{k,i} \text{ оброблено раніше } e_{k,j}, \text{ то } i < j. \quad (2.23)$$

Даний інваріант визначає властивість локальної детермінованості, яка є необхідною умовою коректного функціонування системи, оскільки гарантує узгодженість змін стану в межах одного логічного процесу. Порушення цієї властивості призводить до виникнення неузгоджених станів, які не можуть бути виправлені без повторного виконання послідовності операцій.

Другим ключовим інваріантом є забезпечення ідемпотентності обробки подій, що може бути представлено у вигляді умови:

$$\forall e \in E : f(f(S, e), e) = f(S, e), \quad (2.24)$$

де  $f$  – функція переходу стану. Це означає, що повторна обробка однієї й тієї самої події не повинна змінювати результат виконання, що є критично важливим у системах із семантикою *at-least-once*, де дублювання подій є невід’ємною характеристикою транспортного рівня.

Окрім цього, необхідно враховувати інваріант живучості системи, який полягає у гарантії того, що кожна подія, яка була згенерована, у кінцевому підсумку буде або оброблена, або коректно відхилена. Формально це може бути представлено як:

$$\forall e \in E : P(\text{eventually processed}(e)) = 1, \quad (2.25)$$

за умови наявності хоча б одного доступного каналу доставки та можливості відновлення пропущених подій. Така властивість є особливо важливою для розподілених систем, у яких часткові відмови є типовим явищем.

Узагальнюючи введені інваріанти, можна сформулювати множину вимог до функціонування системи доставки подій:

- забезпечення строгого порядку обробки подій у межах одного ключа;
- гарантування відсутності побічних ефектів при повторній доставці;
- забезпечення завершення обробки подій навіть за умов часткових збоїв;
- підтримка високого рівня паралелізму для подій із різними ключами.

Водночас, аналіз сформованої моделі дозволяє виявити фундаментальне протиріччя, притаманне подієво-орієнтованим мікросервісним системам: з одного боку, підвищення рівня паралелізму є необхідною умовою забезпечення високої продуктивності та масштабованості, а з іншого – зростання паралельності призводить до підвищення ймовірності порушення порядку обробки подій та збільшення накладних витрат на його відновлення.

Формально це протиріччя може бути представлене у вигляді залежності:

$$P(\text{order violation}) \sim f(c, \sigma_t), \quad (2.26)$$

де  $c$  – рівень паралелізму, а  $\sigma_\tau$  – дисперсія затримок доставки. Зі збільшенням цих параметрів зростає ймовірність виникнення розривів у послідовності, що безпосередньо впливає на ефективність функціонування системи.

Узагальнюючи проведену формалізацію, доцільно розглядати подієво-орієнтовану мікросервісну систему як багаторівневу динамічну структуру, функціонування якої визначається взаємодією потоків подій, механізмів їх обробки та обмежень, що накладаються як транспортним середовищем, так і прикладною логікою. При цьому кожна подія інтерпретується не ізольовано, а як елемент впорядкованої послідовності, що визначає еволюцію локального стану відповідного логічного процесу.

З формальної точки зору подія може бути представлена у вигляді кортежу

$$e = (k, s, t, p), \quad (2.27)$$

де ключ  $k$  визначає належність події до певного логічного контексту, порядковий номер  $s$  задає її позицію в межах цього контексту, часовий параметр  $t$  характеризує момент надходження, а  $p$  – інформаційне наповнення. Такий опис дозволяє одночасно враховувати як структурні, так і часові аспекти функціонування системи.

Виходячи з цього, для кожного ключа  $k$  вводиться функція стану  $S(k)$ , яка визначає номер останньої коректно обробленої події. Умова коректності обробки, яка є базовим інваріантом системи, формалізується співвідношенням

$$s = S(k) + 1, \quad (2.28)$$

що означає, що перехід до наступного стану можливий лише за умови отримання саме тієї події, яка логічно продовжує вже виконану послідовність. Усі інші випадки потребують додаткових дій, зокрема буферизації або ігнорування, що відображає необхідність введення механізмів контролю порядку на прикладному рівні.

З урахуванням асинхронної природи системи, де доставка подій здійснюється з використанням семантики *at-least-once*, до моделі інтегруються параметри, що характеризують стохастичні властивості транспортного



середовища. Зокрема, інтенсивність надходження подій для кожного ключа  $\lambda_k$  та швидкість їх обробки  $\mu_k$  визначають коефіцієнт завантаження

$$\rho_k = \frac{\lambda_k}{\mu_k}, \quad (2.29)$$

який, за умови  $\rho_k < 1$ , гарантує існування стаціонарного режиму функціонування системи, тоді як перевищення цього порогу призводить до накопичення подій у чергах і зростання затримок.

Для систематизації введених параметрів доцільно узагальнити їх у вигляді таблиці.

Таблиця 2.1

Основні параметри формалізованої моделі

Позначення	Характеристика	Інтерпретація
$k$	Ключ	логічний контекст обробки
$s$	номер події	позиція у послідовності
$\lambda_k$	інтенсивність	швидкість надходження
$\mu_k$	продуктивність	швидкість обробки
$\rho_k$	завантаження	ступінь використання ресурсу
$p_{loss}$	Втрати	імовірність недоставки
$p_s$	Відновлення	імовірність успішного replay

Важливим аспектом моделі є врахування можливості порушення порядку подій, що виникає внаслідок варіації затримок доставки. У цьому випадку вводиться поняття розриву послідовності, який виникає тоді, коли до системи надходить подія з номером, що перевищує очікуване значення. Для оцінки ефективності механізму відновлення таких ситуацій використовується залежність ймовірності успішного відновлення після  $R_{спроб}$ :

$$P_{\text{success}} = 1 - (1 - p_s)^R, \quad (2.30)$$

що дозволяє формалізувати вплив параметрів системи на її здатність до самовідновлення в умовах часткових втрат або затримок.

У свою чергу, часові характеристики функціонування системи визначаються не лише швидкістю обробки подій, але й додатковими витратами, пов'язаними з очікуванням, буферизацією та відновленням. З урахуванням цього загальний час перебування події у системі може бути представлений у вигляді:

$$T_{total} = T_{wait} + T_{proc} + T_{buf} + T_{rec} \quad (2.31)$$

де кожна складова відображає окремий аспект функціонування, а їх сумарний вплив визначає латентність системи в цілому.

З метою узагальнення режимів обробки подій, що виникають у процесі функціонування системи, доцільно представити їх у табличній формі.

Таблиця 2.2

Режими обробки подій

Умова	Інтерпретація	Дія системи
$s = S(k) + 1$	очікувана подія	обробка
$s > S(k) + 1$	розрив послідовності	буферизація/очікування
$s \leq S(k)$	дублювання	ігнорування

Подальше узагальнення моделі дозволяє ввести інтегральний показник накладних витрат  $\varepsilon$ , який відображає сумарний вплив механізмів забезпечення впорядкованості на ефективність функціонування системи. Даний показник є функцією параметрів середовища передачі, рівня паралелізму та характеристик механізмів відновлення, тобто

$$\varepsilon = f(p_{loss}, p_s, T_{gap}, R, c) \quad (2.32)$$

де  $c$  – рівень паралельної обробки. Аналіз цієї залежності свідчить про те, що збільшення паралелізму, з одного боку, підвищує пропускну здатність системи, а з іншого – призводить до зростання накладних витрат, пов'язаних із необхідністю підтримання впорядкованості.

Таким чином, узагальнена формалізована модель дозволяє представити систему доставки подій як стохастичну динамічну систему, у якій поєднуються процеси генерації, передачі та обробки подій, а також механізми контролю їх

послідовності. Водночас проведений аналіз показує, що забезпечення строгого порядку обробки при збереженні високого рівня паралелізму є нетривіальним завданням, оскільки ці вимоги мають взаємно суперечливий характер.

Узагальнюючи отримані результати, подієво-орієнтовану систему доцільно представити у вигляді інтегрованої формалізованої структури, яка відображає взаємозв'язок між потоками подій, механізмами їх обробки та параметрами середовища функціонування:

$$M = (K, E, Q, W, S, \lambda_k, \mu_k, p_{loss}, p_s, R, T_{gap}, c, \Phi), \quad (2.33)$$

за умови виконання наступних співвідношень:

$$s = S(k) + 1, \quad \rho_k = \frac{\lambda_k}{\mu_k} < 1, \quad (2.34)$$

$$P_{success} = 1 - (1 - p_s)^R, \quad (2.35)$$

$$T_{total} = T_{wait} + T_{proc} + T_{buf} + T_{rec} \quad (2.36)$$

$$\varepsilon = f(p_{loss}, p_s, T_{gap}, R, c) \quad (2.37)$$

де  $K$  – множина ключів впорядкування, що визначають незалежні логічні контексти обробки подій (наприклад, окремі процеси або транзакції);  $E$  – множина подій, кожна з яких описується параметрами ключа, порядкового номера, часу та змісту;  $Q = \{Q_k\}$  – множина внутрішніх черг, що забезпечують локальну буферизацію та впорядкування подій для кожного ключа;  $W$  – множина обробників, які реалізують паралельну обробку подій у системі;  $S(k)$  – функція стану, що визначає номер останньої коректно обробленої події для ключа  $k$ ;  $\lambda_k$  – інтенсивність надходження подій для ключа  $k$ ;  $\mu_k$  – швидкість обробки подій, яка визначається обчислювальними ресурсами системи;  $\rho_k$  – коефіцієнт завантаження, що характеризує ступінь використання ресурсів і визначає умову стабільності системи;  $p_{loss}$  – імовірність втрати або недоставки події у транспортному середовищі;  $p_s$  – імовірність успішного відновлення події при використанні механізму повторної доставки;  $R$  – максимальна кількість спроб відновлення пропущеної події;  $T_{gap}$  – час очікування перед ініціацією процедури відновлення;  $c$  – рівень паралелізму, що визначає кількість одночасно виконуваних процесів обробки;  $\Phi$  – функція переходу стану, яка визначає зміну

системного стану під впливом обробки подій;  $P_{success}$  – ймовірність успішного відновлення пропущених подій;  $T_{total}$  – сумарний час перебування події у системі, який включає всі складові затримок;  $\varepsilon$  – узагальнений показник накладних витрат, що відображає вплив механізмів забезпечення впорядкованості на продуктивність системи.

Запропонована модель дозволяє розглядати систему доставки подій як стохастичну динамічну систему, у якій забезпечення строгої впорядкованості обробки подій у межах кожного ключа поєднується з необхідністю підтримання високого рівня паралелізму. Водночас залежності між параметрами моделі демонструють, що зростання рівня паралельної обробки супроводжується збільшенням накладних витрат та ускладненням механізмів контролю порядку, що обґрунтовує доцільність розробки спеціалізованого методу упорядкованого паралелізму, спрямованого на узгодження зазначених вимог.

Формалізована модель

$$M = (K, E, Q, W, S, \lambda_k, \mu_k, p_{loss}, p_s, R, T_{gap}, c, \Phi), \quad (2.38)$$

є достатньо повною для опису умов функціонування методу упорядкованого паралелізму, оскільки вона враховує як структурні компоненти системи, так і стохастичні характеристики асинхронного середовища передачі повідомлень.

Зокрема, наявність ключа впорядкування  $k \in K$  та функції стану  $S(k)$  дозволяє формалізувати локальну послідовність обробки подій, що є базовою умовою реалізації впорядкованого паралелізму. Введене співвідношення

$$s = S(k) + 1, \quad (2.39)$$

визначає необхідну і достатню умову коректності виконання, яка забезпечує строгий порядок обробки подій у межах кожного логічного контексту незалежно від порядку їх фактичного надходження.

Водночас модель явно враховує асинхронну природу доставки через параметри  $p_{loss}$ ,  $p_s$ ,  $R$  та  $T_{gap}$ , що дозволяє описати механізми відновлення пропущених подій і тим самим забезпечити властивість живучості системи. Залежність

$$P_{success} = 1 - (1 - p_s)^R, \quad (2.40)$$

показує, що за умови ненульової ймовірності успішного відновлення та скінченної кількості спроб система здатна відновити порушену послідовність, що є критично важливим для роботи методу в умовах *at-least-once* семантики.

Крім того, введення множини внутрішніх черг  $Q = \{Q_k\}$  та обмеженої множини обробників  $W$  створює передумови для реалізації паралельної обробки незалежних потоків подій. У цьому випадку рівень паралелізму свизначає ступінь одночасного виконання, тоді як умова стабільності

$$\rho_k = \frac{\lambda_k}{\mu_k} < 1 \quad (2.41)$$

гарантує, що система не переходить у перевантажений режим, у якому впорядкування стає неефективним або неможливим.

Разом із тим, модель також відображає обмеження, у межах яких метод може функціонувати коректно. Зокрема, збільшення рівня паралелізму призводить до зростання ймовірності порушення порядку надходження подій та збільшення накладних витрат

$$\varepsilon = f(p_{loss}, p_s, T_{gap}, R, c) \quad (2.42)$$

що обумовлює необхідність використання додаткових механізмів, таких як буферизація, ідемпотентність та повторна доставка.

Таким чином, можна стверджувати, що сформульована модель не лише сумісна з методом упорядкованого паралелізму, але й визначає умови його коректного застосування, зокрема:

- наявність ключів впорядкування;
- забезпечення ідемпотентності обробки;
- підтримка механізму відновлення пропущених подій;
- виконання умов стабільності за навантаженням.

Узагальнюючи, модель створює необхідне теоретичне підґрунтя для реалізації методу упорядкованого паралелізму в асинхронних мікросервісних системах і підтверджує можливість поєднання строгого порядку обробки подій із високим рівнем паралельної обробки за умови дотримання визначених обмежень.

У зв'язку з цим подальше дослідження спрямовується на розробку методу, який дозволяє узгодити зазначені вимоги шляхом перенесення механізмів впорядкування на прикладний рівень і використання додаткових структур керування потоками подій. Формалізації такого методу та його алгоритмічній реалізації присвячено підрозділ 2.2.

## **2.2. Побудова методу упорядкованого паралелізму в умовах асинхронної доставки**

Сформована у попередньому підрозділі формалізована подієво-орієнтована модель дозволяє описати структуру та закономірності функціонування мікросервісної системи доставки подій, однак сама по собі вона не визначає конкретних механізмів забезпечення впорядкованості обробки у реальних умовах асинхронної взаємодії. Зокрема, введені залежності демонструють, що у системі одночасно діють дві протилежні тенденції: необхідність підтримання високого рівня паралелізму для забезпечення продуктивності та вимога строгого порядку обробки подій у межах кожного логічного контексту.

Як було встановлено, у рамках моделі умова коректності обробки подій формалізується співвідношенням

$$s = S(k) + 1, \quad (2.43)$$

що забезпечує детерміновану еволюцію стану системи. Водночас асинхронна доставка подій, яка характеризується наявністю затримок, повторної доставки та потенційних втрат, призводить до ситуацій, коли події надходять у порядку, що не відповідає їх логічній послідовності. У такому випадку безпосереднє застосування цієї умови у традиційних механізмах обробки призводить або до блокування виконання (у разі очікування відсутніх подій), або до порушення узгодженості станів (у разі ігнорування порядку).

Таким чином, виникає необхідність у побудові спеціалізованого методу, який дозволяє реалізувати вимогу

$$s = S(k) + 1, \quad (2.44)$$

не на рівні транспортного середовища, де гарантії впорядкованості є обмеженими, а на рівні прикладної логіки системи, використовуючи додаткові механізми керування потоками подій.

Ідея методу упорядкованого паралелізму полягає у розділенні глобального потоку подій на множину незалежних підпотоків, кожен з яких відповідає окремому ключу  $k$ , із забезпеченням строгої послідовності обробки всередині кожного такого підпотoku при збереженні можливості паралельного виконання між різними ключами. Такий підхід дозволяє поєднати дві, на перший погляд, взаємовиключні вимоги – локальну впорядкованість і глобальний паралелізм.

З формальної точки зору, це означає перехід від глобальної функції обробки подій

$$\Phi : E \rightarrow S \quad (2.45)$$

до сімейства локальних функцій

$$\Phi_k : E_k \rightarrow S_k, \quad (2.46)$$

кожна з яких визначає обробку подій для відповідного ключа. При цьому виконання функцій  $\Phi_k$  може здійснюватися паралельно для різних значень  $k$ , що забезпечує масштабованість системи, тоді як всередині кожного підпроцесу зберігається строгий порядок виконання.

Практична реалізація такого підходу передбачає використання структур типу внутрішніх черг  $Q_k$ , які формуються динамічно відповідно до появи нових ключів у системі. Кожна така черга виконує роль буфера впорядкування, у якому події накопичуються до моменту, коли стає можливим їх коректне виконання відповідно до умови порядку. Саме на цьому рівні реалізується механізм синхронізації, який у традиційних підходах покладається на транспортний рівень.

Важливо підкреслити, що метод не змінює властивостей середовища доставки, а адаптується до них, враховуючи параметри  $p_{loss}$ ,  $p_s$ ,  $T_{gap}$  та інші характеристики, введені у моделі. Зокрема, механізм відновлення пропущених подій дозволяє компенсувати недоліки асинхронної доставки, тоді як ідемпотентність забезпечує коректність обробки у випадку повторної доставки.

Методу упорядкованого паралелізму передбачає перехід від загального принципу декомпозиції потоків подій до чітко визначеної системи структурних компонентів, кожен із яких виконує окрему функцію у забезпеченні впорядкованості та паралельності обробки. На відміну від узагальненої моделі, яка описує систему в термінах змінних і залежностей, метод конкретизує, яким чином ці залежності реалізуються у вигляді алгоритмічних та організаційних механізмів.

У межах запропонованого підходу система обробки подій може бути представлена як композиція взаємодіючих компонентів:

$$M_{method} = (P, D, Q, C, R, J), \quad (2.47)$$

де  $P$  – компонент генерації та публікації подій,  $D$  – транспортне середовище доставки,  $Q$  – підсистема впорядкування (внутрішні черги),  $C$  – компонент обробки (виконавці),  $R$  – механізм відновлення (replay),  $J$  – механізм забезпечення ідемпотентності.

Кожен із зазначених компонентів виконує строго визначену функцію, і лише їх узгоджена взаємодія забезпечує виконання базових інваріантів, сформульованих у попередньому підрозділі.

Зокрема, компонент  $P$  відповідає за формування подій і присвоєння їм порядкових номерів  $s$ , які є критично важливими для подальшого контролю послідовності. Саме на цьому етапі реалізується гарантія монотонного зростання значень  $s$  для кожного ключа  $k$ , що створює основу для впорядкування.

Компонент  $D$ , який зазвичай представлений брокером повідомлень або набором каналів передачі, виконує функцію транспортування подій, проте не гарантує глобальної впорядкованості. Його поведінка описується стохастичними параметрами, введеними раніше, що обумовлює необхідність компенсаційних механізмів на рівні методу.

Центральним елементом методу є підсистема  $Q$ , яка реалізує розділення потоків подій за ключами та організовує їх у вигляді множини внутрішніх черг  $Q_k$ . Кожна така черга відповідає окремому логічному процесу та функціонує як буфер, що забезпечує можливість відкладеної обробки подій у випадку



порушення їх послідовності. Саме тут реалізується практичне застосування умови

$$s = S(k) + 1, \quad (2.48)$$

оскільки обробка події дозволяється лише за її відповідності очікуваному номеру.

Компонент  $\mathcal{S}$  відповідає за безпосереднє виконання обробки подій і реалізується у вигляді множини паралельних обробників. Його функціонування визначається політикою розподілу ресурсів, яка повинна забезпечувати ефективне використання обчислювальних потужностей при одночасному дотриманні обмежень впорядкованості.

Механізм  $\mathcal{R}$  реалізує відновлення пропущених подій у випадку виявлення розривів у послідовності. Його функціонування базується на повторному запиті відсутніх подій та використанні збережених станів системи, що дозволяє гарантувати властивість живучості навіть у випадку втрат або затримок у транспортному середовищі.

Компонент  $\mathcal{I}$ , у свою чергу, забезпечує ідемпотентність обробки, що є необхідною умовою коректного функціонування методу в умовах повторної доставки подій. Його роль полягає у фільтрації дублювань та запобіганні повторному виконанню вже оброблених операцій.

Для узагальнення функціонального призначення компонентів методу доцільно представити їх у табличній формі.

Таблиця 2.3

Структурні компоненти методу упорядкованого паралелізму

Компонент	Призначення	Основна функція
$\mathcal{P}$	генерація подій	присвоєння порядкових номерів
$\mathcal{D}$	доставка	передача подій
$\mathcal{Q}$	впорядкування	буферизація та синхронізація
$\mathcal{C}$	обробка	виконання подій
$\mathcal{R}$	відновлення	replay пропущених подій
$\mathcal{I}$	ідемпотентність	усунення дублювань

Формалізація структурних компонентів методу дозволяє перейти від абстрактного опису системи до конкретної архітектурної організації процесу обробки подій, у якій кожен елемент виконує визначену роль у забезпеченні впорядкованості та паралельності. Запропонована структура є основою для подальшого визначення алгоритму функціонування методу, який описує послідовність дій системи при надходженні та обробці подій в умовах асинхронної доставки.

Після визначення структурних компонентів методу доцільно перейти до формалізації його алгоритмічної реалізації, яка відображає послідовність дій системи при надходженні, впорядкуванні та обробці подій. На відміну від попередніх етапів, де розглядалися загальні принципи та архітектурні елементи, у даному випадку увага зосереджується на процедурному аспекті функціонування методу, що забезпечує практичну реалізацію встановлених інваріантів.

Алгоритмічна основа методу базується на обробці кожної вхідної події

$$e = (k, s, t, p), \quad (2.49)$$

у контексті поточного стану  $S(k)$  та відповідної внутрішньої черги  $Q_k$ . При цьому ключовим є забезпечення умови коректності

$$s = S(k) + 1, \quad (2.50)$$

яка визначає допустимість виконання операції.

Загальна логіка алгоритму може бути представлена як послідовність взаємопов'язаних етапів.

На початковому етапі відбувається прийом події із транспортного середовища та її ідентифікація за ключем  $k$ . У разі відсутності відповідної внутрішньої черги здійснюється її динамічне створення, що забезпечує адаптивність системи до змін структури навантаження. Таким чином, формується множина активних черг  $Q_k$ , яка змінюється у часі залежно від кількості оброблюваних логічних процесів.

Далі подія поміщається до відповідної черги, після чого здійснюється перевірка її порядкового номера відносно поточного стану системи. У випадку, коли виконується умова

$$s = S(k) + 1, \quad (2.51)$$

подія негайно передається на виконання обробнику, що відповідає за даний ключ, і після успішного завершення обробки відбувається оновлення стану

$$S(k) \leftarrow s. \quad (2.52)$$

У ситуації, коли

$$s > S(k) + 1, \quad (2.53)$$

виявляється розрив у послідовності, що означає відсутність однієї або кількох попередніх подій. У цьому випадку подія зберігається у буфері черги  $Q_k$ , а система переходить у режим очікування протягом інтервалу  $T_{gap}$ . Якщо протягом цього часу необхідна подія не надходить, ініціюється процедура відновлення, яка передбачає виконання запитів на повторну доставку відсутніх елементів із використанням механізму replay.

Якщо ж виконується умова

$$s < S(k) \quad (2.54)$$

подія інтерпретується як дубль, що виник унаслідок повторної доставки, і не підлягає повторній обробці, що забезпечується за рахунок ідемпотентності операцій.

Важливою складовою алгоритму є організація паралельної обробки подій, яка реалізується через розподіл внутрішніх черг між обробниками. Формально це може бути описано функцією відображення

$$П : Q_k \rightarrow W, \quad (2.55)$$

яка визначає, який саме обробник виконує обробку подій для конкретного ключа. При цьому різні черги можуть обслуговуватися незалежно, що забезпечує масштабованість системи, тоді як обробка всередині кожної черги залишається послідовною.

З урахуванням зазначеного, алгоритм функціонування методу може бути узагальнено представлений у табличній формі як послідовність логічних умов і відповідних дій.

Узагальнений алгоритм обробки подій

Умова	Стан системи	Дія
надходження $e(k, s)$	черга відсутня	створення $Q_k$
$s = S(k) + 1$	коректна подія	обробка, оновлення $S(k)$
$s > S(k) + 1$	розрив	буферизація, очікування
timeout $T_{gap}$	подія відсутня	запуск replay
$s \leq S(k)$	дубль	ігнорування

Алгоритмічна реалізація методу забезпечує узгодження асинхронної доставки подій із вимогами до строгої послідовності їх обробки шляхом введення механізмів буферизації, контролю стану та відновлення пропущених елементів. Важливо підкреслити, що реалізація даного алгоритму не потребує модифікації транспортного середовища, а здійснюється виключно на рівні прикладної логіки, що забезпечує універсальність і сумісність із різними брокерами повідомлень.

Основна умова коректної обробки:

$$s > S(k) + 1, \quad (2.56)$$

Умова дублювання:

$$s < S(k) \quad (2.57)$$

Умова розриву послідовності:

$$s > S(k) + 1, \quad (2.58)$$

Ймовірність успішного відновлення пропущеної події:

$$P_{\text{success}} = 1 - (1 - p_s)^R, \quad (2.59)$$

Умова стабільності обробки для ключа:

$$\rho_k = \frac{\lambda_k}{\mu_k} < 1 \quad (2.60)$$

Блок-схема відображає логіку функціонування методу упорядкованого паралелізму, згідно з якою кожна подія після надходження ідентифікується за ключем  $k$ , після чого для відповідного логічного потоку створюється або використовується внутрішня черга  $Q_k$ .

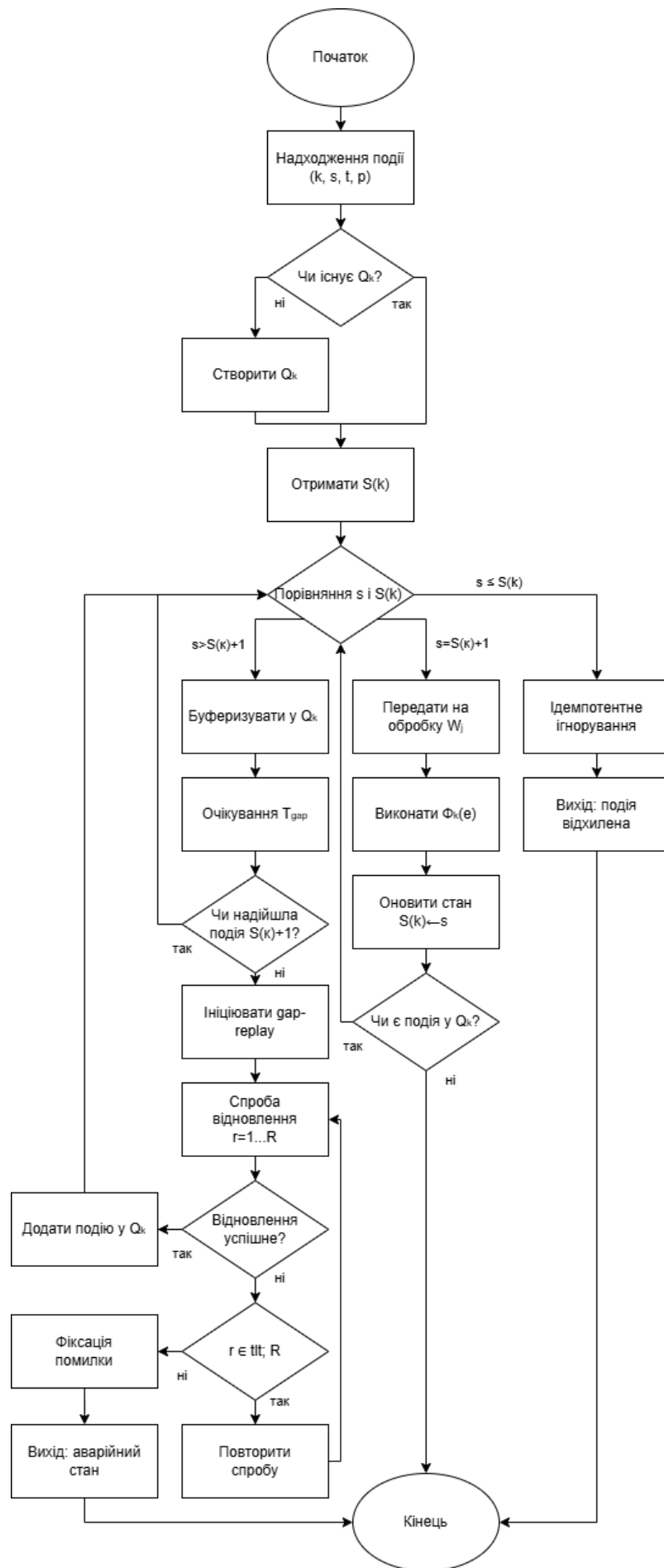


Рис. 2.1. Блок-схема алгоритму методу упорядкованого паралелізму

Подальша обробка події залежить від співвідношення її порядкового номера  $s$  із поточним станом  $S(k)$ , що дозволяє відрізнити коректну подію, дубль або подію з розривом послідовності.

У разі коректної послідовності подія передається на обробку, після чого стан системи оновлюється. Якщо подія надходить передчасно, вона буферизується, а система очікує відсутню подію протягом  $T_{gap}$ ; у разі її ненадходження ініціюється механізм *gap-replay*, який виконується не більше ніж  $R$  разів. Алгоритм дозволяє забезпечити строгий порядок обробки подій у межах кожного ключа при збереженні паралельної обробки незалежних потоків.

У результаті проведеного дослідження побудовано метод упорядкованого паралелізму в умовах асинхронної доставки подій, який базується на декомпозиції глобального потоку подій на множину незалежних підпотоків за ключами впорядкування  $k$ , введенні функції стану  $S(k)$  та використанні умови коректності

$$s = S(k) + 1, \quad (2.61)$$

як базового інваріанту обробки. На відміну від традиційних підходів, у яких впорядкованість забезпечується засобами транспортного рівня, запропонований метод реалізує контроль послідовності на рівні прикладної логіки, що дозволяє компенсувати стохастичний характер доставки подій, зумовлений наявністю затримок, повторної доставки та можливих втрат.

Формалізація структурних компонентів методу та побудова алгоритму його функціонування дозволили визначити узагальнену схему обробки подій, яка включає операції ідентифікації, буферизації, перевірки порядку, відновлення пропущених елементів та ідемпотентної обробки. Зокрема, показано, що використання внутрішніх черг  $Q_k$ , механізму *gap-replay* та функції розподілу обробників забезпечує можливість одночасної реалізації двох ключових властивостей: строгої локальної впорядкованості та глобального паралелізму.

Разом із тим, проведена побудова методу має узагальнений характер і визначає, передусім, логіку функціонування системи та взаємодію її компонентів. Реалізація зазначених принципів у реальних мікросервісних середовищах

потребує визначення конкретних механізмів прикладного рівня, які забезпечують збереження стану  $S(k)$ , організацію внутрішніх черг, обробку конкурентних доступів, а також інтеграцію з транспортними протоколами передачі повідомлень.

Завершуючи побудову методу упорядкованого паралелізму, можна стверджувати, що отримані результати створюють концептуальну та алгоритмічну основу для забезпечення впорядкованої обробки подій у асинхронних системах, однак їх практичне застосування вимагає подальшої деталізації на рівні програмної реалізації.

У зв'язку з цим у наступному підрозділі 2.3 доцільно розглянути механізми прикладного рівня впорядкування подій, які забезпечують реалізацію запропонованого методу в умовах реальних мікросервісних систем.

### **2.3. Реалізація механізмів прикладного рівня впорядкування подій**

Реалізація запропонованого методу на прикладному рівні передбачає трансформацію формалізованих залежностей та алгоритмічних конструкцій у конкретні програмні механізми, здатні функціонувати в умовах розподіленого середовища з асинхронною доставкою повідомлень та конкурентним доступом до ресурсів. На відміну від теоретичної моделі, де функція стану  $S(k)$  та множина черг  $Q_k$  розглядаються як абстрактні сутності, у реальній системі вони повинні бути реалізовані у вигляді конкретних структур даних і процедур доступу, що забезпечують коректність, узгодженість і відмовостійкість.

Ключовим елементом прикладної реалізації є механізм керування станом для кожного ключа впорядкування  $k$ , який відповідає за збереження значення  $S(k)$  та його оновлення відповідно до умови

$$s = S(k) + 1, \quad (2.62)$$

З огляду на те, що обробка подій здійснюється у середовищі з потенційно високим рівнем паралелізму, зазначений механізм повинен забезпечувати атомарність операцій читання та запису, а також узгодженість стану при

одночасному доступі з боку декількох обробників. Це досягається шляхом використання транзакційних або квазітранзакційних підходів, зокрема через застосування оптимістичних або песимістичних механізмів синхронізації.

Паралельно з цим реалізується підсистема внутрішніх черг  $Q_k$ , яка виконує функцію буферизації подій у випадку порушення їх послідовності. З практичної точки зору така підсистема може бути реалізована у вигляді асоціативної структури, що відображає ключі  $k$  у відповідні черги або пріоритетні набори подій, впорядковані за значенням  $s$ . Використання впорядкованих структур даних дозволяє ефективно визначати наявність події з номером  $S(k) + 1$  та ініціювати її обробку без додаткових витрат на пошук.

Важливим компонентом прикладного рівня є механізм виявлення розривів у послідовності, який реалізується шляхом порівняння порядкового номера події з поточним станом системи. У разі виконання нерівності

$$s = S(k) + 1, \quad (2.63)$$

подія не може бути оброблена негайно і підлягає буферизації, після чого запускається процедура контролю часу очікування  $T_{gap}$ . Якщо протягом заданого інтервалу відсутня подія з очікуваним номером, ініціюється механізм відновлення, який може бути реалізований як повторний запит до джерела подій або як звернення до резервного каналу передачі.

Окрему роль у прикладній реалізації відіграє механізм ідемпотентності, який забезпечує коректність обробки у випадку повторної доставки подій. Практично це досягається шляхом збереження інформації про вже оброблені події, наприклад, через ведення журналу виконаних операцій або використання унікальних ідентифікаторів подій, що дозволяють ідентифікувати дублікати та запобігти їх повторному виконанню.

Подальша деталізація прикладної реалізації передбачає формалізацію окремих механізмів у вигляді узгодженої системи структур даних і операцій доступу, що забезпечують виконання інваріантів методу в умовах конкурентного середовища та асинхронної доставки повідомлень. При цьому центральним



елементом виступає механізм керування станом, який повинен гарантувати атомарність переходу

$$S(k) \xrightarrow{e=(k,s,\cdot)} S(k) = s \text{ лише за умови } s = S(k) + 1, \quad (2.64)$$

що на практиці реалізується як умовне оновлення (compare-and-set):

$$S(k) \leftarrow s \Leftrightarrow S(k)_{\text{old}} = s - 1. \quad (2.65)$$

Такий підхід унеможлиблює некоректні переходи стану за конкурентного доступу та виключає «перестрибування» через відсутні події.

Таблиця 2.5

Відображення елементів моделі у механізми прикладного рівня

Формальний елемент	Прикладна реалізація	Основна операція
$S(k)$	сховище стану (in-memory/БД)	CAS-оновлення
$Q_k$	впорядкована черга / map	вставка, вибір мінімуму
$e = (k, s, t, p)$	повідомлення з X-Sequence-ID	Десеріалізація
перевірка $s = S(k) + 1$	умовний оператор	Валідація
replay	повторний запит	fetch/HTTP
ідемпотентність	журнал оброблених подій	Lookup

Паралельно із збереженням стану формується підсистема впорядкування, у якій для кожного ключа  $k$  підтримується буфер подій  $Q_k$ , впорядкований за значенням  $s$ . З метою мінімізації часу доступу до «наступної» події доцільно використовувати структури типу впорядкованого відображення або пріоритетної черги, для яких операції вставки та вибору мінімального елемента мають складність  $O(\log_b n)$ , що обумовлено використанням двійкових деревоподібних структур, у яких висота визначається логарифмом за основою 2 від кількості елементів. У цьому випадку наявність події, що задовольняє умову  $s = S(k) + 1$ , перевіряється за сталий або логарифмічний час.

Важливою складовою реалізації є формалізація операції обробки події як атомарної транзакції, що поєднує перевірку порядку, виконання бізнес-логіки та оновлення стану. Узагальнено така операція може бути представлена як:

$$Process(e) = \begin{cases} \text{apply}(p), & S(k) \leftarrow s, \text{ якщо } s = S(k) + 1, \\ \text{buffer}(e), & \text{якщо } s > S(k) + 1, \\ \text{ignore}(e), & \text{якщо } s \leq S(k). \end{cases} \quad (2.66)$$

З урахуванням асинхронної доставки, де можливі втрати або затримки, вводиться механізм керування очікуванням та відновленням. Нехай  $\Delta_k = s - S(k) - 1$  – величина розриву послідовності. Якщо  $\Delta_k > 0$ , система переходить у режим очікування протягом інтервалу  $T_{gap}$ , після чого ініціюється процедура відновлення. Інтенсивність повторних запитів може бути описана як:

$$\lambda_{\text{replay}} = \frac{1}{T_{\text{gap}}} \cdot I(\Delta_k > 0), \quad (2.67)$$

де  $I(\cdot)$  – індикаторна функція. Це дозволяє пов'язати параметри механізму відновлення з навантаженням на систему.

Сукупність описаних компонентів формує узгоджену прикладну архітектуру, яку доцільно представити у вигляді узагальненої схеми взаємодії.



Рис. 2.2. Узагальнена схема реалізації механізмів прикладного рівня впорядкування подій

У наведеній схемі відображено узагальнену архітектуру реалізації механізмів прикладного рівня впорядкування подій, яка забезпечує виконання інваріантів методу в умовах асинхронної доставки.

Генератор подій формує послідовності подій із присвоєнням їм порядкових номерів, після чого вони передаються до брокера повідомлень, який виконує



забезпечення коректної обробки подій у мікросервісних системах за умов асинхронної доставки повідомлень. Її основною метою є трансформація формалізованих залежностей, отриманих у попередніх підрозділах, у конкретні програмні механізми, які гарантують дотримання порядку обробки подій при збереженні високого рівня паралелізму.

У функціональному відношенні схема реалізує повний життєвий цикл події від моменту її генерації до завершення обробки та фіксації результату. Генератор подій формує повідомлення у вигляді кортежу  $e = (k, s, t, p)$ , де ключ  $k$  визначає логічний контекст, а порядковий номер  $s$  – позицію події у відповідній послідовності. Подальша передача здійснюється через брокер повідомлень, який виступає транспортним середовищем і, як правило, не гарантує глобальної впорядкованості доставки.

Центральним елементом схеми є диспетчер подій, який виконує декомпозицію вхідного потоку за ключами  $k$  та координує взаємодію між іншими компонентами системи. На цьому етапі здійснюється направлення подій до буфера впорядкування  $Q_k$  та ініціалізація взаємодії зі сховищем стану  $S(k)$ , яке зберігає інформацію про останню коректно оброблену подію.

Буфер впорядкування  $Q_k$  реалізує функцію тимчасового зберігання подій і забезпечує їх упорядкування за значенням  $s$ . Завдяки використанню впорядкованих структур даних операції вставки та вибору наступної події виконуються зі складністю

$$O(\log_2 |Q_k|), \quad (2.69)$$

що дозволяє ефективно працювати навіть при значних обсягах даних.

Подальша обробка визначається перевіркою умови коректності

$$s = S(k) + 1, \quad (2.70)$$

яка реалізується у вигляді логічного блоку. У разі виконання цієї умови подія передається до відповідного потоку обробки  $W_j$ , де виконується бізнес-логіка  $\Phi_k(e)$ , після чого здійснюється атомарне оновлення стану системи. Використання операцій типу compare-and-set забезпечує узгодженість даних при конкурентному доступі.

Якщо ж подія не задовольняє умову порядку, здійснюється додаткова перевірка на наявність розриву послідовності. У випадку, коли виконується нерівність

$$s > S(k) + 1, \quad (2.71)$$

подія буферизується, а система переходить у режим очікування протягом інтервалу  $T_{gap}$ . Якщо протягом цього часу відсутня подія з номером  $S(k) + 1$ , активується механізм відновлення (*gap-replay*), який ініціює повторну доставку пропущених елементів. Інтенсивність таких запитів визначається як

$$\lambda_{\text{replay}} = \frac{1}{T_{\text{gap}}}, \quad (2.72)$$

що дозволяє контролювати навантаження на систему.

У випадку, коли виконується умова

$$s < S(k) \quad (2.73)$$

подія інтерпретується як дубль і обробляється ідемпотентно, тобто без повторного виконання операцій, що забезпечує стабільність системи у середовищах із семантикою *at-least-once*.

Паралельність обробки забезпечується за рахунок використання пулу потоків  $W = \{W_1, \dots, W_c\}$ , де свизначає рівень паралелізму. При цьому події з різними ключами можуть оброблятися одночасно, тоді як для кожного окремого ключа зберігається послідовність виконання.

Завершальним етапом є формування результату обробки, який відповідає коректно впорядкованій послідовності подій і може бути використаний у подальших бізнес-процесах або переданий до інших сервісів.

Розглянута схема призначена для реалізації систем, у яких необхідно забезпечити строгий порядок обробки подій у межах окремих логічних процесів за відсутності гарантій впорядкованої доставки на транспортному рівні. До таких систем належать:

- мікросервісні архітектури з асинхронною взаємодією;
- системи електронної комерції (обробка замовлень);
- фінансові системи (послідовність транзакцій);

- системи обробки потоків даних (event streaming);
- розподілені корпоративні інформаційні системи.

Запропонована схема відображає практичну реалізацію методу упорядкованого паралелізму та демонструє, яким чином формальні залежності і алгоритмічні конструкції трансформуються у конкретні механізми прикладного рівня. Її використання дозволяє забезпечити баланс між вимогами до впорядкованості, продуктивності та відмовостійкості, що є критично важливим для сучасних розподілених систем.

Таким чином, у підрозділі 2.3 було сформовано систему механізмів прикладного рівня, яка забезпечує реалізацію методу упорядкованого паралелізму через інтеграцію компонентів керування станом  $S(k)$ , буферизації  $Q_k$ , відновлення пропущених подій та ідемпотентної обробки. Запропоновані механізми дозволяють відобразити формалізовані залежності методу у конкретні програмні конструкції та забезпечити виконання інваріанту впорядкованості

$$s > S(k) + 1, \quad (2.74)$$

в умовах асинхронної доставки повідомлень.

Водночас наведені рішення визначають, передусім, структурно-організаційні аспекти реалізації методу та описують окремі функціональні компоненти системи, тоді як питання їх узгодженої взаємодії в рамках єдиного процесу обробки подій потребує додаткової формалізації. Зокрема, необхідно визначити послідовність виконання операцій, правила переходу між станами системи та умови активації окремих механізмів з урахуванням як вимог до впорядкованості, так і необхідності забезпечення паралельної обробки.

У цьому контексті доцільним є виділення узагальненого алгоритму обробки подій, який інтегрує розглянуті механізми у цілісну процедуру функціонування системи та забезпечує одночасне виконання вимог до послідовності та паралелізму. Такий алгоритм повинен формалізувати взаємодію між компонентами прикладного рівня, визначити умови запуску обробки, буферизації та відновлення, а також встановити правила розподілу подій між паралельними обробниками.

Завершуючи розгляд механізмів прикладного рівня впорядкування подій, доцільно побудувати алгоритм обробки подій із забезпеченням послідовності та паралелізму, що узагальнює отримані результати та визначає процедурну основу функціонування запропонованого методу.

#### **2.4. Алгоритм обробки подій із забезпеченням послідовності та паралелізму**

У даному підрозділі здійснюється формалізація алгоритму обробки подій, який узагальнює розроблений метод упорядкованого паралелізму та механізми його прикладної реалізації, розглянуті у підрозділах 2.2–2.3. На відміну від попередніх підрозділів, де основна увага приділялася структурним компонентам системи та принципам їх взаємодії, у даному підрозділі розглядається цілісна алгоритмічна процедура функціонування системи як послідовність формалізованих операцій над подіями та станами.

Необхідність побудови такого алгоритму обумовлена тим, що в умовах асинхронної доставки повідомлень із семантикою *at-least-once* забезпечення строгої послідовності обробки подій не може бути досягнуте виключно засобами транспортного рівня, а потребує визначення чітких правил переходу між станами системи з урахуванням можливих розривів послідовності, повторної доставки та конкурентної обробки. У зв'язку з цим алгоритм повинен одночасно задовольняти дві ключові вимоги: збереження інваріанту впорядкованості

$$s > S(k) + 1, \quad (2.75)$$

для кожного ключа  $k$  та забезпечення можливості паралельної обробки незалежних потоків подій.

З урахуванням зазначеного, алгоритм обробки подій доцільно розглядати як відображення, що визначає перехід системи зі стану  $S(k)$  та буферу  $Q_k$  у новий стан під впливом вхідної події  $e = (k, s, t, p)$ . Такий підхід дозволяє перейти від описового представлення процесу до його строгої формалізації у вигляді функції переходу станів, яка визначає всі можливі сценарії обробки подій у системі.

З урахуванням викладених положень алгоритм обробки подій доцільно формалізувати як функцію переходу станів, яка визначає зміну стану системи та вмісту буфера впорядкування під впливом кожної вхідної події. Нехай стан системи для ключа  $k$  задається парою

$$\Sigma_k = (S(k), Q_k), \quad (2.76)$$

де  $S(k)$  – поточний стан (номер останньої коректно обробленої події), а  $Q_k$  – множина буферизованих подій, впорядкованих за значенням  $s$ .

Тоді алгоритм можна визначити як відображення

$$A: (e, \Sigma_k) \rightarrow \Sigma'_k, \quad (2.77)$$

де  $e = (k, s, t, p)$  – вхідна подія, а  $\Sigma'_k = (S'(k), Q'_k)$  – новий стан системи після її обробки.

Залежно від співвідношення між значеннями  $s$  та  $S(k)$ , функція переходу набуває різних форм, що відповідають можливим сценаріям обробки подій:

1. Випадок коректної послідовності.

Якщо виконується умова

$$s = S(k) + 1, \quad (2.78)$$

подія вважається наступною у послідовності та підлягає негайній обробці. У цьому випадку відбувається виконання функції обробки  $\Phi_k(e)$  та оновлення стану:

$$S'(k) = s. \quad (2.79)$$

Одночасно здійснюється перевірка наявності у буфері  $Q_k$  подій, які можуть бути оброблені після оновлення стану. Формально це може бути представлено як ітеративне застосування:

$$\text{while } \exists e' \in Q_k : s' = S(k) + 1 \Rightarrow S(k) \leftarrow s', Q_k \leftarrow Q_k \setminus \{e'\}. \quad (2.80)$$

Таким чином, у даному випадку функція переходу має вигляд:

$$\Sigma_{k'} = (S(k) \leftarrow s^*, Q_k \setminus E^*), \quad (2.81)$$

де  $E^* \subseteq Q_k$  – множина послідовно оброблених буферизованих подій.

2. Випадок розриву послідовності.

Якщо виконується нерівність



$$s > S(k) + 1, \quad (2.82)$$

подія не може бути оброблена негайно, оскільки відсутні попередні елементи послідовності. У цьому випадку вона додається до буфера:

$$Q_{k'} = Q_k \cup \{e\}, S'(k) = S(k). \quad (2.83)$$

Додатково фіксується величина розриву:

$$\Delta_k = s - S(k) - 1, \quad (2.84)$$

яка використовується для ініціації механізму відновлення. Якщо протягом часу  $T_{gap}$  подія з номером  $S(k) + 1$  не надходить, запускається процедура *gap-replay*.

### 3. Випадок дублювання події.

Якщо виконується умова

$$s < S(k) \quad (2.85)$$

подія вважається дубльованою і не підлягає повторній обробці. У цьому випадку:

$$S'(k) = S(k), Q_{k'} = Q_k. \quad (2.86)$$

Коректність такого переходу забезпечується властивістю ідемпотентності, яка гарантує, що повторне надходження вже обробленої події не змінює стан системи.

Об'єднуючи наведені випадки, функцію переходу можна записати у вигляді:

$$A(e, \Sigma_k) = \begin{cases} (S(k), Q_k \cup \{e\}), & s > S(k) + 1, \\ (S(k), Q_k \setminus E^*), & s = S(k) + 1, \\ (S(k), Q_k), & s < S(k). \end{cases} \quad (2.87)$$

Отримана формалізація визначає алгоритм як систему локальних правил переходу станів для кожного ключа  $k$ , що дозволяє розглядати обробку подій як сукупність незалежних процесів. Це, у свою чергу, створює передумови для паралельного виконання алгоритму, оскільки переходи для різних значень  $k$  не впливають один на одного.

Оскільки запропонований алгоритм задається локальними правилами переходу станів для кожного ключа впорядкування  $k$ , його паралельне виконання ґрунтується на тому, що події, які належать до різних логічних контекстів, можуть оброблятися незалежно одна від одної. При цьому послідовність виконання

зберігається лише в межах одного ключа, тоді як між різними ключами не встановлюється обмеження тотального порядку. Саме ця властивість дозволяє поєднати локальну детермінованість із глобальним паралелізмом.

Для формалізації розподілу обробки введемо множину обробників:

$$W = \{W_1, W_2, \dots, W_c\}, \quad (2.88)$$

де  $c$  визначає рівень паралелізму системи, тобто кількість одночасно доступних потоків або виконавців. Тоді розподіл ключів між обробниками може бути представлений відображенням:

$$\Pi: K \rightarrow W, \quad (2.89)$$

яке кожному ключу  $k \in K$  ставить у відповідність певний обробник  $W_j$ . Це означає, що події одного ключа передаються в межах одного послідовного логічного потоку обробки, тоді як події з різними ключами можуть виконуватися різними обробниками паралельно.

У такому випадку для двох подій  $e_i = (k_i, s_i, t_i, p_i)$  та  $e_j = (k_j, s_j, t_j, p_j)$  умова їх незалежної паралельної обробки може бути записана як:

$$k_i \neq k_j \Rightarrow \mathcal{A}(e_i, \Sigma_{k_i}) \parallel \mathcal{A}(e_j, \Sigma_{k_j}), \quad (2.90)$$

де символ  $\parallel$  означає можливість одночасного виконання відповідних операцій без порушення локальної послідовності станів.

Водночас для подій, що належать одному ключу, зберігається обмеження:

$$k_i = k_j, s_i < s_j \Rightarrow e_i < e_j, \quad (2.91)$$

де  $<$  позначає відношення передування, яке визначає необхідність обробки події  $e_i$  раніше за подію  $e_j$ . Таким чином, алгоритм не формує глобального порядку для всіх подій системи, оскільки це призвело б до втрати продуктивності, а забезпечує локальний порядок у межах кожного ключа.

З практичної точки зору такий підхід дозволяє уникнути обмеження, характерного для послідовної обробки всього потоку повідомлень, коли система змушена використовувати один канал виконання для збереження порядку. Натомість запропонований алгоритм забезпечує паралельність на рівні незалежних логічних потоків, що дозволяє масштабувати обробку відповідно до кількості активних ключів та доступних обчислювальних ресурсів.

Якщо кількість активних ключів у певний момент часу позначити як  $|K_a|$ , то максимально можливий рівень фактичного паралелізму визначається співвідношенням:

$$c_{\text{eff}} = \min(|K_a|, c), \quad (2.92)$$

де  $c_{\text{eff}}$  – ефективний рівень паралельного виконання,  $|K_a|$  – кількість активних ключів, а  $c$  – кількість доступних обробників. Це співвідношення показує, що навіть за великої кількості потоків обробки реальний паралелізм обмежується кількістю незалежних логічних контекстів.

Для узагальнення принципів паралельної обробки доцільно подати їх у вигляді таблиці 2.8.

Таблиця 2.8

Принципи забезпечення послідовності та паралелізму в алгоритмі

Умова	Характер обробки	Результат
$k_i = k_j, s_i < s_j$	послідовна обробка	збереження порядку в межах ключа
$k_i \neq k_j$	паралельна обробка	незалежне виконання різних потоків

Отже, паралельність у запропонованому алгоритмі не суперечить вимозі впорядкованості, оскільки вона реалізується не всередині одного потоку подій, а між різними потоками, які не мають причинно-наслідкової залежності між собою. Саме така декомпозиція дозволяє забезпечити масштабованість системи без порушення коректності обробки подій.

Побудована блок-схема відображає повну алгоритмічну процедуру функціонування системи обробки подій у межах запропонованого методу упорядкованого паралелізму та формалізує всі можливі сценарії переходу станів системи залежно від параметрів вхідної події. Схема побудована відповідно до стандартів алгоритмічного моделювання та включає базові елементи: початок і завершення процесу, блоки введення/виведення, операційні блоки та блоки прийняття рішень із формалізованими умовами.

Алгоритм ініціюється з блоку початку, після чого здійснюється введення події у вигляді кортежу

$$e = (k, s, t, p), \quad (2.93)$$

де  $k$  визначає ключ впорядкування, а  $s$  – порядковий номер події. На наступному етапі виконується ідентифікація ключа та визначення відповідного логічного підпроцесу  $\Sigma_k$ , що дозволяє локалізувати обробку в межах конкретного контексту.

Далі здійснюється зчитування поточного стану системи  $S(k)$  та буфера впорядкування  $Q_k$ , після чого перевіряється наявність структури буферизації для відповідного ключа. У разі її відсутності відбувається ініціалізація  $Q_k$  та початкового стану  $S(k)$ , що забезпечує коректний старт обробки нових потоків подій.

Після цього виконується розподіл події між обробниками за допомогою відображення

$$W_j = \Pi(k), \quad (2.94)$$

що визначає, який саме потік обробки відповідатиме за обробку подій із даним ключем. Цей етап закладає основу для паралельного виконання алгоритму.

Центральним елементом блок-схеми є перевірка умови впорядкованості

$$s = S(k) + 1, \quad (2.95)$$

яка визначає три можливі гілки виконання алгоритму.

### **1. Гілка коректної послідовності.**

Якщо умова виконується, подія передається до обробника, де виконується функція

$$\Phi_k(e), \quad (2.96)$$

після чого відбувається атомарне оновлення стану

$$S(k) \leftarrow s. \quad (2.97)$$

Після цього алгоритм переходить до перевірки буфера  $Q_k$  на наявність подій, які можуть бути оброблені наступними. Якщо така подія існує (тобто  $s' = S(k) + 1$ ), вона вилучається з буфера та обробляється аналогічно, що реалізує механізм каскадної обробки. У разі відсутності таких подій формуються результати обробки та алгоритм переходить до завершення.

### **2. Гілка розриву послідовності**

Якщо виконується умова

$$s > S(k) + 1, \quad (2.98)$$

алгоритм переходить у режим буферизації: подія додається до  $Q_k$ , після чого обчислюється величина розриву

$$\Delta_k = s - S(k) - 1. \quad (2.99)$$

Далі система очікує надходження відсутньої події протягом інтервалу  $T_{gap}$ . Якщо необхідна подія надходить, алгоритм повертається до гілки коректної обробки.

У випадку, якщо подія не надходить, ініціюється механізм відновлення (*gap-replay*), який виконує серію спроб повторної доставки пропущених подій. Якщо відновлення успішне, відновлена подія додається до буфера і алгоритм повертається до перевірки порядку. Якщо ж після  $R_{спроб}$  відновлення не відбулося, фіксується помилка та формується відповідний вихідний стан системи.

### 3. Гілка дубльованих подій

Якщо виконується умова

$$s \leq S(k) \quad (2.100)$$

подія ідентифікується як дубльована. У цьому випадку виконується ідемпотентне ігнорування події без зміни стану системи, після чого формується відповідний вихідний результат.

Незалежно від гілки виконання, після завершення відповідних операцій алгоритм переходить до блоку завершення, що означає завершення обробки поточної події.

Побудована блок-схема демонструє, що алгоритм:

- забезпечує строге дотримання порядку обробки подій через інваріант

$$s = S(k) + 1; \quad (2.101)$$

- підтримує паралельну обробку через розподіл ключів між обробниками

$$\Pi : k \rightarrow W_j; \quad (2.102)$$

- є стійким до дублювання завдяки ідемпотентності;
- забезпечує відновлення послідовності через механізм *gap-replay*;
- адаптується до асинхронного середовища за рахунок буферизації та контролю часу очікування.

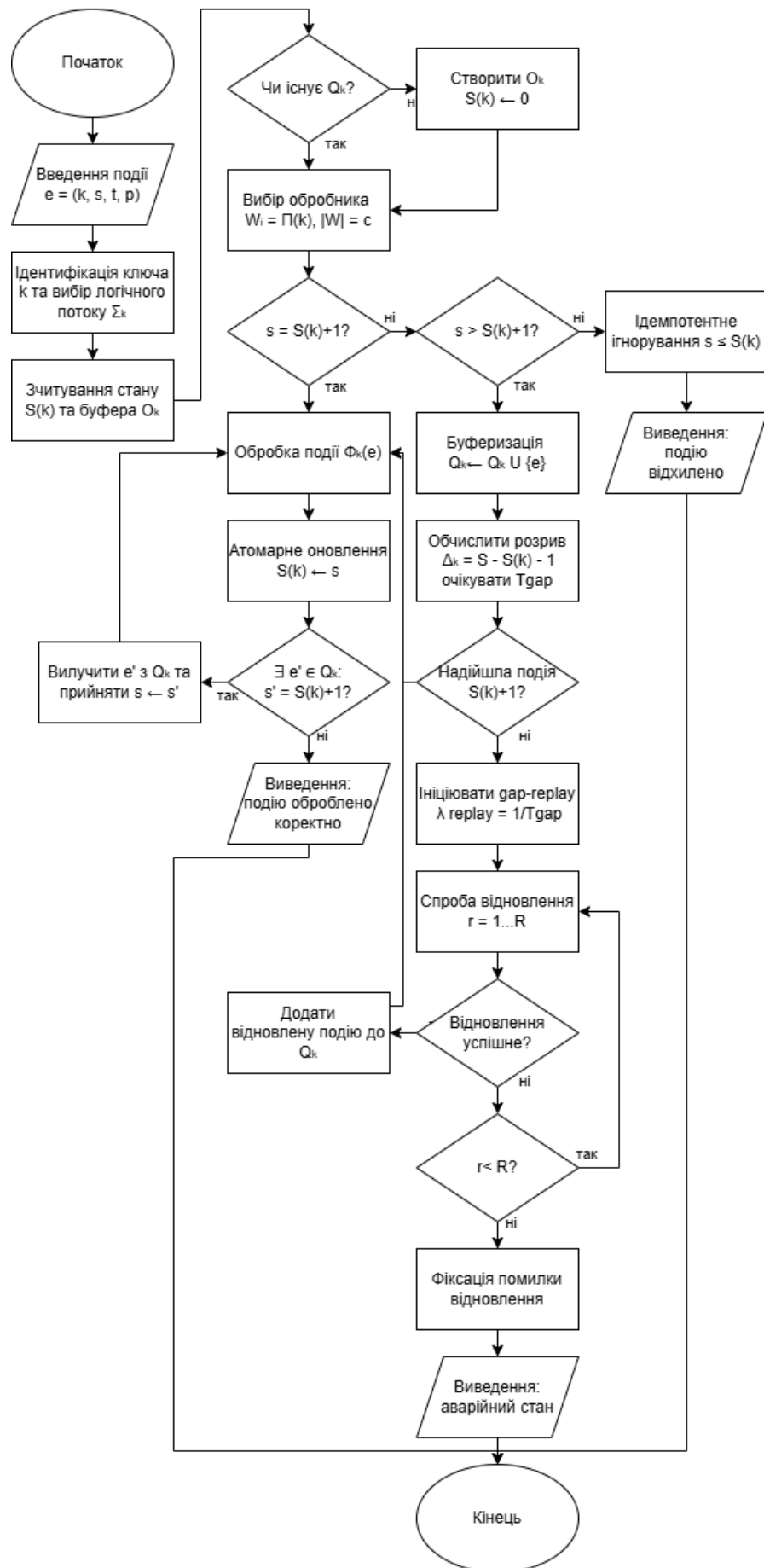


Рис. 2.4. Детальна блок-схема алгоритму обробки події із забезпечення послідовності та паралелізму

Побудований алгоритм забезпечує узгоджену обробку подій і поєднує вимоги до послідовності та паралелізму, однак його ефективне функціонування в умовах асинхронної доставки значною мірою залежить від здатності системи коректно реагувати на порушення послідовності та повторну доставку повідомлень. У зв'язку з цим виникає необхідність детального розгляду механізмів, що забезпечують відновлення пропущених станів і коректну обробку дубльованих подій.

## **2.5. Механізми відновлення пропущених станів та забезпечення ідемпотентності**

У даному підрозділі розглядаються механізми, що забезпечують коректне функціонування алгоритму обробки подій в умовах асинхронної доставки повідомлень, зокрема у випадках порушення послідовності та повторної доставки подій. Необхідність їх впровадження обумовлена тим, що в реальних розподілених системах доставка повідомлень має недетермінований характер, що призводить до появи затримок, втрат або дублювання подій, які без відповідної обробки можуть порушити інваріанти алгоритму.

Одним із ключових механізмів є відновлення пропущених станів, яке активується у випадку виявлення розриву у послідовності подій для певного ключа. Така ситуація виникає, коли система отримує подію з порядковим номером, що перевищує очікуваний, що свідчить про відсутність одного або кількох попередніх елементів послідовності. У цьому випадку подія тимчасово буферизується, а система переходить у режим очікування, протягом якого здійснюється моніторинг надходження відсутніх подій.

Якщо протягом визначеного інтервалу часу відсутні події не надходять, ініціюється процедура відновлення, яка передбачає повторний запит або повторну доставку відповідних елементів послідовності. Реалізація такого механізму може базуватися на використанні резервного каналу передачі, повторному зверненні до джерела подій або застосуванні журналів подій, що дозволяють реконструювати пропущені стани. Важливою характеристикою

цього процесу є обмеження кількості спроб відновлення, що запобігає нескінченним циклам очікування та дозволяє своєчасно виявляти критичні збої.

Паралельно з цим значну роль відіграє механізм забезпечення ідемпотентності, який гарантує, що повторна доставка однієї і тієї ж події не призводить до некоректної зміни стану системи. У практичній реалізації це досягається шляхом фіксації інформації про вже оброблені події та перевірки їх унікальності перед виконанням бізнес-логіки. Зокрема, можуть використовуватися ідентифікатори подій, журнали виконаних операцій або контрольні значення стану, що дозволяють виявляти дублікати та запобігати їх повторній обробці.

Важливо підкреслити, що механізми відновлення та ідемпотентності тісно взаємодіють між собою. З одного боку, відновлення пропущених станів може призводити до повторної доставки вже оброблених подій, що вимагає наявності ідемпотентної обробки. З іншого боку, ідемпотентність дозволяє безпечно виконувати повторні запити, не побоюючись порушення коректності стану системи. Така взаємодія забезпечує властивість живучості алгоритму, тобто здатність системи завершити обробку всіх подій навіть у разі часткових збоїв.

Важливим аспектом реалізації механізмів відновлення пропущених станів є вибір стратегії керування затримками та повторними запитами. Зокрема, фіксований інтервал очікування може бути неефективним в умовах змінного навантаження, тому доцільним є використання адаптивних підходів, за яких параметри очікування та частота повторних запитів змінюються залежно від поточного стану системи, інтенсивності надходження подій та характеристик мережевої взаємодії. Такий підхід дозволяє зменшити накладні витрати на відновлення та уникнути перевантаження системи зайвими запитами.

Окремої уваги заслуговує питання узгодженості стану при відновленні, оскільки в умовах розподіленої системи можливі ситуації, коли події надходять із різних джерел або каналів доставки. У таких випадках важливо забезпечити, щоб відновлені події інтегрувалися в систему без порушення вже сформованого стану, що досягається шляхом використання єдиної моделі ідентифікації подій та централізованого або узгодженого механізму перевірки їх актуальності.



Крім того, забезпечення ідемпотентності може бути реалізоване на різних рівнях системи, зокрема на рівні бізнес-логіки, рівні доступу до даних або рівні інфраструктурних компонентів. Найбільш ефективним є комбінований підхід, за якого ідемпотентність забезпечується як через контроль ідентифікаторів подій, так і через побудову операцій, що не змінюють стан системи при повторному виконанні. Це дозволяє підвищити стійкість до помилок навіть у випадках часткової втрати інформації про попередні обробки.

Додатково слід зазначити, що запропоновані механізми впливають не лише на надійність, але й на продуктивність системи, оскільки буферизація, повторні запити та перевірка ідемпотентності створюють додаткові обчислювальні витрати. У зв'язку з цим виникає задача оптимального вибору параметрів цих механізмів, що потребує подальшого аналітичного дослідження та буде розглянута у наступних розділах роботи.



Рис. 2.5. Схема механізму відновлення пропущених станів

Схема демонструє, що у випадку порушення послідовності система переходить у режим буферизації та очікування, після чого ініціює процедуру відновлення. Такий підхід дозволяє уникнути передчасної обробки подій і забезпечити коректне відновлення стану.



Рис. 2.6. Схема механізму ідемпотентності

Механізм ідемпотентності гарантує, що навіть при повторній доставці події стан системи не буде змінено некоректно. Це досягається за рахунок перевірки унікальності події перед виконанням обробки.

Таблиця 2.10

Порівняльна характеристика механізмів забезпечення надійності

Механізм	Призначення	Основні операції	Вплив на систему
Відновлення станів	Усунення розривів	буферизація, replay	підвищення живучості
Ідемпотентність	Усунення дублювання	перевірка, журналювання	стабільність стану

Таблиця 2.11

Рівні забезпечення ідемпотентності

Рівень	Реалізація	Особливості
Логічний	перевірка ID	проста реалізація
Дані	контроль стану	висока надійність
Інфраструктура	брокер/БД	залежність від технологій

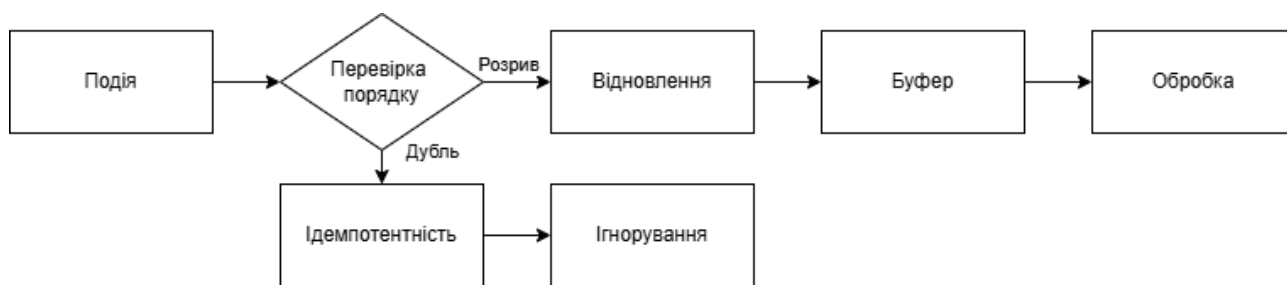


Рис. 2.7. Схема взаємодії механізмів

Подані механізми не функціонують ізольовано, а утворюють єдину систему забезпечення надійності обробки подій. Зокрема, відновлення пропущених станів може призводити до повторної доставки повідомлень, що, у свою чергу, вимагає застосування ідемпотентної обробки. Така взаємодія

дозволяє забезпечити стабільність системи навіть у випадках часткових збоїв або втрати повідомлень.

Таким чином, у підрозділах 2.4–2.5 було сформовано алгоритмічну основу методу та розглянуто механізми, що забезпечують його стійкість до порушень послідовності та повторної доставки подій. Запропоновані рішення дозволяють гарантувати коректну обробку подій у розподіленому середовищі, однак їх ефективність потребує теоретичного обґрунтування з точки зору фундаментальних властивостей функціонування системи.

У цьому контексті доцільним є дослідження властивостей коректності та живучості методу, які визначають здатність системи забезпечувати правильність обробки подій та завершення всіх процесів навіть за наявності збоїв і асинхронності. Саме аналіз цих властивостей дозволяє формально підтвердити надійність і придатність запропонованого підходу для використання в практичних мікросервісних системах.

## **2.6. Дослідження властивостей коректності та живучості методу**

У даному підрозділі здійснюється дослідження властивостей коректності та живучості запропонованого методу упорядкованого паралелізму, що реалізує обробку подій в умовах асинхронної доставки повідомлень. Необхідність такого дослідження обумовлена тим, що побудова алгоритму та відповідних механізмів його реалізації сама по собі не гарантує правильності функціонування системи, а потребує формального обґрунтування виконання ключових вимог до обробки подій у розподіленому середовищі.

Під коректністю у даній роботі розуміється здатність системи забезпечувати узгоджений та детермінований результат обробки подій для кожного ключа впорядкування, незалежно від порядку їх надходження, затримок або повторної доставки. Це означає, що для будь-якої послідовності подій, що належать одному логічному контексту, система повинна відтворювати той самий результат, який був би отриманий у випадку їх ідеально впорядкованої доставки.

Водночас під живучістю розуміється властивість системи гарантувати завершення обробки всіх подій за скінченний час, навіть у випадках часткових збоїв, втрати повідомлень або тимчасової недоступності окремих компонентів. Ця властивість є критичною для асинхронних систем, оскільки відсутність механізмів відновлення може призвести до блокування обробки або накопичення необроблених подій.

З урахуванням зазначеного, дослідження властивостей методу доцільно проводити шляхом формалізації відповідних критеріїв та доведення виконання інваріантів, що визначають правильність переходів між станами системи. Такий підхід дозволяє перейти від описового представлення алгоритму до його строгого математичного обґрунтування та встановити умови, за яких запропонований метод забезпечує необхідний рівень надійності.

Після формалізації інваріантів коректності доцільно перейти до дослідження властивості живучості, оскільки саме вона визначає здатність запропонованого методу не лише зберігати правильний порядок обробки подій, але й забезпечувати поступальний рух системи до завершення обробки навіть за умов асинхронної доставки, затримок, повторних повідомлень або часткових втрат. Якщо коректність методу відповідає на питання, чи не буде порушено порядок виконання, то живучість характеризує, чи не залишиться система у стані нескінченного очікування пропущеної події.

Для формального аналізу живучості розглянемо логічний потік подій, що відповідає певному ключу впорядкування  $k$ . Нехай для цього ключа поточний стан системи визначається значенням  $S(k)$ , а наступною очікуваною подією є подія з порядковим номером  $S(k) + 1$ . Система вважається живучою для ключа  $k$ , якщо за умови існування джерела відновлення та доступності хоча б одного каналу доставки вона здатна за скінченний час перейти від поточного стану до наступного коректного стану.

У формалізованому вигляді цю властивість можна подати так:

$$\forall k \in K, \forall s > S(k) : P(S(k) \rightarrow s) \rightarrow 1, \quad (2.103)$$

за умови, що пропущені події зберігаються у джерелі відновлення, а механізм повторної доставки має ненульову ймовірність успішного завершення. Це означає, що для кожного ключа система не повинна залишатися заблокованою на очікуванні відсутнього стану, якщо цей стан може бути відновлений за допомогою передбачених механізмів.

У межах запропонованого методу живучість забезпечується трьома взаємопов'язаними механізмами: по-перше, буферизацією подій, що надійшли раніше за очікувану подію; по-друге, очікуванням протягом інтервалу  $T_{gap}$ , який дозволяє врахувати природні затримки асинхронного середовища; по-третє, ініціацією процедури відновлення пропущених подій у разі, якщо очікувана подія не надійшла у визначений проміжок часу.

Нехай ймовірність успішного відновлення пропущеної події за одну спробу дорівнює  $p_s$ , а максимальна кількість спроб відновлення становить  $R$ . Тоді ймовірність успішного відновлення пропущеної події після не більше ніж  $R$  спроб визначається співвідношенням:

$$P_{\text{success}} = 1 - (1 - p_s)^R. \quad (2.104)$$

З наведеного співвідношення випливає, що за умови  $p_s > 0$  збільшення кількості спроб  $R$  приводить до зростання ймовірності успішного відновлення, а отже, зменшує ризик тривалого блокування відповідного потоку подій. При цьому кількість спроб не може збільшуватися необмежено, оскільки кожна спроба створює додаткове навантаження на систему, тому параметр  $R$  має визначатися з урахуванням допустимого рівня затримки та накладних витрат.

Для оцінювання граничної ситуації, у якій пропущена подія не була відновлена після всіх спроб, доцільно ввести ймовірність невдалого відновлення:

$$P_{\text{fail}} = (1 - p_s)^R. \quad (2.105)$$

Цей показник характеризує ризик переходу системи до стану аварійної обробки або компенсаційного сценарію. Зменшення  $P_{\text{fail}}$  до прийнятного рівня є однією з умов практичного забезпечення живучості, оскільки дозволяє

мінімізувати кількість випадків, у яких логічний потік залишається незавершеним.

Узагальнення умов забезпечення живучості подано в таблиці 2.13.

Таблиця 2.13

Умови забезпечення живучості методу упорядкованого паралелізму

Умова	Зміст	Вплив на живучість
Наявність буфера $Q_k$	Події, що надійшли передчасно, не втрачаються	Забезпечує можливість подальшої обробки після відновлення порядку
Визначення $T_{gap}$	Система очікує природне надходження пропущеної події	Запобігає передчасному запуску відновлення
Наявність механізму replay	Пропущена подія може бути повторно отримана	Усуває блокування потоку
Виконання умови $p_s > 0$	Існує ненульова ймовірність успішного відновлення	Забезпечує можливість прогресу
Обмеження кількості спроб $R$	Відновлення не переходить у нескінченний цикл	Забезпечує контрольований вихід зі збою
Ідемпотентність обробки	Повторно доставлені події не змінюють стан повторно	Дозволяє безпечно виконувати replay

З таблиці видно, що живучість методу не забезпечується окремим компонентом, а формується як результат взаємодії механізмів буферизації, відновлення та ідемпотентності. Буферизація запобігає втраті передчасно отриманих подій, replay усуває розриви у послідовності, а ідемпотентність гарантує, що повторне надходження подій під час відновлення не призведе до некоректної зміни стану.

Логіку забезпечення живучості методу можна подати у вигляді послідовності станів, що відображає реакцію системи на виявлений розрив у потоці подій.

На рис. 2.8 показано, що система не переходить у стан безмежного очікування, оскільки після завершення інтервалу  $T_{gap}$  активується механізм

відновлення, а кількість повторних спроб обмежується параметром  $R$ . Саме наявність контрольованого переходу від очікування до відновлення, а після вичерпання спроб – до аварійного або компенсаційного сценарію, дозволяє розглядати метод як такий, що забезпечує живучість у практичному сенсі.

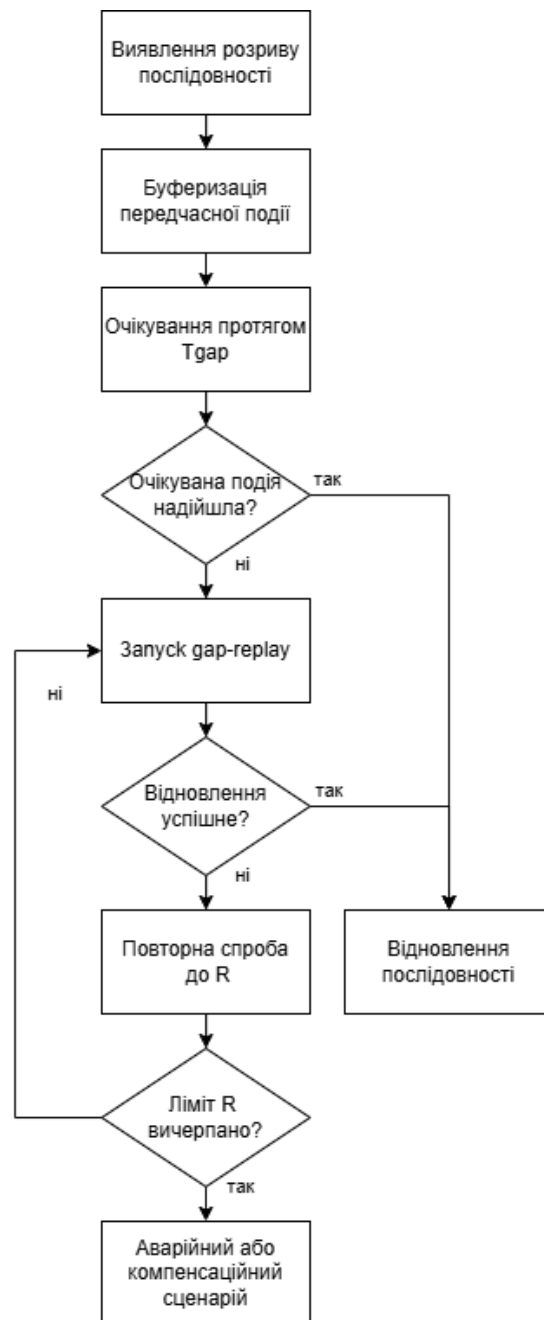


Рис. 2.8. Узагальнена схема забезпечення живучості методу при виявленні розриву послідовності подій

З теоретичної точки зору властивість живучості може бути сформульована у вигляді такого твердження.

**Твердження 2.1.** Якщо для кожного ключа впорядкування  $k$  усі події зберігаються у джерелі відновлення протягом часу, достатнього для виконання процедури *replay*, імовірність успішного відновлення однієї події є додатною, а кількість спроб відновлення обмежена, то запропонований метод забезпечує завершення обробки послідовності подій або контрольований перехід до аварійного сценарію без нескінченного блокування системи.

Обґрунтування цього твердження ґрунтується на тому, що будь-який розрив у послідовності переводить відповідний потік не у пасивний стан невизначеного очікування, а у стан керованого відновлення. Оскільки кожна спроба *replay* має ймовірність успішного завершення  $p_s > 0$ , ймовірність того, що всі  $R$  спроб завершаться невдало, визначається як  $P_{fail}$ . Отже, система або відновлює пропущену подію і продовжує обробку, або після вичерпання допустимої кількості спроб фіксує збій і передає процес до компенсаційного сценарію, що виключає нескінченне блокування.

Таким чином, живучість запропонованого методу забезпечується не припущенням про ідеальну доставку повідомлень, а наявністю формалізованого механізму реагування на втрату або затримку подій. Це є принципово важливим для мікросервісних систем, у яких асинхронність, повторна доставка та часткові збої є не винятковими, а типовими умовами функціонування.

З урахуванням сформульованих інваріантів коректності та встановлених умов забезпечення живучості доцільно розглянути узагальнені властивості функціонування методу як цілісної системи, що поєднує механізми впорядкування, буферизації, відновлення та ідемпотентної обробки. Такий підхід дозволяє перейти від аналізу окремих компонентів до оцінювання їх узгодженої взаємодії у межах єдиного алгоритмічного процесу.

Передусім слід зазначити, що коректність і живучість методу не є незалежними характеристиками, а формують взаємодоповнюючу систему вимог. Зокрема, забезпечення строгого порядку обробки подій призводить до необхідності очікування відсутніх елементів послідовності, що потенційно може знижувати живучість системи. Водночас застосування механізмів відновлення дозволяє усунути такі затримки, однак створює ризик повторної доставки подій,



що, у свою чергу, потребує забезпечення ідемпотентності. Таким чином, ефективність методу визначається балансом між вимогами до коректності та живучості.

Для формалізації цього взаємозв'язку доцільно ввести узагальнену характеристику прогресу системи. Нехай для ключа  $k$  функція прогресу визначається як:

$$\Pi_k(t) = S(k, t), \quad (2.106)$$

де  $S(k, t)$  – значення стану системи у момент часу  $t$ . Тоді система вважається такою, що забезпечує прогрес, якщо для будь-якого ключа виконується умова монотонності:

$$S(k, t_2) \geq S(k, t_1), \forall t_2 > t_1. \quad (2.107)$$

Це означає, що стан системи не зменшується з часом, а отже, відсутні зворотні переходи, які могли б порушити вже досягнуту послідовність обробки. Водночас для забезпечення живучості необхідно, щоб для будь-якого допустимого стану існувала можливість переходу до наступного стану:

$$\exists t' : S(k, t') > S(k, t). \quad (2.108)$$

Поєднання цих умов дозволяє сформулювати узагальнену властивість функціонування методу як умову монотонного прогресу, яка одночасно враховує вимоги до коректності та живучості.

Для систематизації отриманих результатів доцільно подати їх у вигляді узагальнюючої таблиці.

Таблиця 2.14

Узагальнені властивості функціонування методу

Властивість	Формалізація	Зміст
Впорядкованість	збереження відношення передування	правильний порядок обробки
Узгодженість	відповідність стану виконаним подіям	відсутність суперечностей
Ідемпотентність	інваріант повторної обробки	стійкість до дублювання
Живучість	наявність переходу до наступного стану	відсутність блокування
Прогрес	монотонність $S(k, t)$	гарантоване просування

Аналіз наведених властивостей показує, що запропонований метод забезпечує узгоджене виконання всіх необхідних вимог до обробки подій у розподіленому середовищі. Зокрема, коректність гарантує правильність результату, живучість – можливість завершення обробки, а прогрес – відсутність деградації системи з часом.

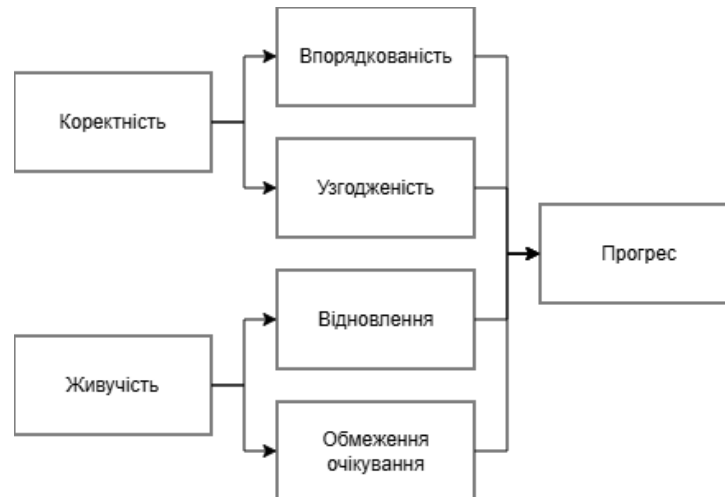


Рис. 2.9. Взаємозв'язок властивостей коректності, живучості та прогресу

Для наочного відображення взаємозв'язку між властивостями доцільно подати їх у вигляді структурної схеми.

Як показано на рис. 2.9, властивість прогресу формується як результат взаємодії компонентів, що відповідають за коректність та живучість. Це свідчить про те, що відсутність хоча б одного з цих компонентів може призвести до порушення загальної стабільності системи.

З практичної точки зору важливим є також питання впливу параметрів системи на забезпечення розглянутих властивостей. Зокрема, вибір інтервалу очікування, кількості спроб відновлення та розміру буфера безпосередньо впливає на баланс між швидкістю обробки та ймовірністю успішного відновлення. Надто малий інтервал очікування може призвести до надмірної кількості повторних запитів, тоді як надто великий – до затримок у обробці. Аналогічно, недостатня кількість спроб відновлення підвищує ризик втрати прогресу, тоді як надмірна – збільшує навантаження на систему.

Узагальнюючи, можна зробити висновок, що запропонований метод упорядкованого паралелізму забезпечує виконання властивостей коректності та

живучості за умов дотримання визначених інваріантів та параметричних обмежень. При цьому його ефективність визначається не лише структурою алгоритму, але й вибором параметрів, що регулюють процес відновлення та буферизації.

Для підтвердження властивості коректності методу доцільно розглянути зміну стану системи у часі, яка подана на рисунку 2.10.

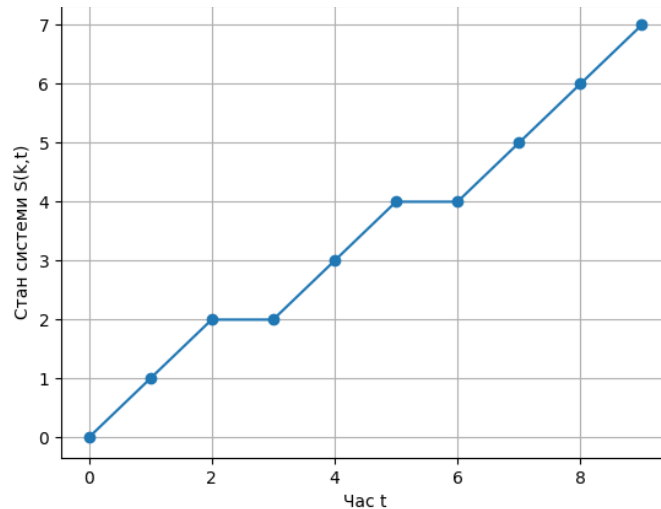


Рис. 2.10. Залежність стану системи від часу (властивість коректності)

Як видно з рис. 2.10, значення стану системи є монотонно неспадною функцією часу, що свідчить про відсутність зворотних переходів та підтверджує виконання інваріанту узгодженості. Таким чином, система не порушує порядок обробки подій і зберігає детермінованість результату.

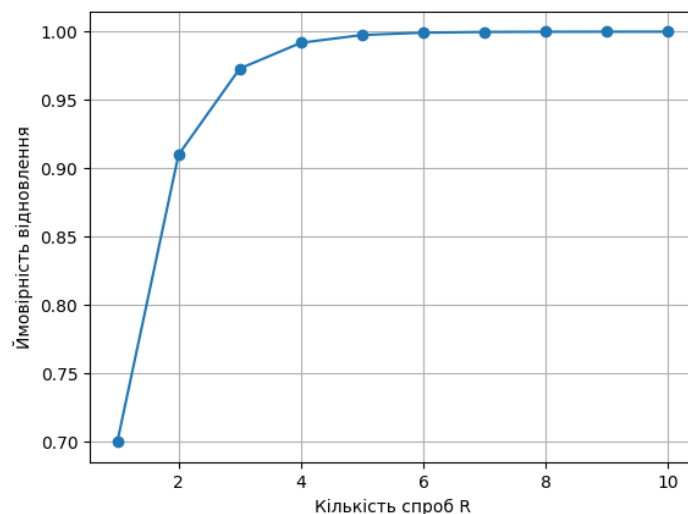


Рис. 2.11. Залежність ймовірності відновлення від кількості спроб

Для ілюстрації властивості живучості розглянемо залежність ймовірності успішного відновлення пропущених подій від кількості спроб replay, яка наведена на рисунку 2.11.

Як видно з рисунка 2.11, зі збільшенням кількості спроб ймовірність успішного відновлення асимптотично наближається до одиниці, що підтверджує здатність системи забезпечувати завершення обробки подій за скінченний час. Це дозволяє зробити висновок про виконання властивості живучості методу.

**Наслідок 2.1.** За умови виконання інваріантів коректності та забезпечення ненульової ймовірності відновлення пропущених подій, система забезпечує монотонний прогрес стану для кожного ключа впорядкування.

Проведене дослідження показало, що запропонований метод упорядкованого паралелізму забезпечує виконання ключових властивостей функціонування розподілених систем обробки подій, зокрема коректності та живучості. Формалізація інваріантів та аналіз умов відновлення дозволили встановити, що система не лише зберігає правильний порядок обробки подій, але й гарантує завершення обробки за скінченний час навіть у випадках асинхронності та часткових збоїв. Це, у свою чергу, забезпечує монотонний прогрес стану системи та підтверджує придатність запропонованого методу для практичного застосування в мікросервісних архітектурах.

Таким чином, проведене дослідження підтверджує, що метод є теоретично обґрунтованим та придатним для застосування у системах асинхронної обробки подій, що створює основу для подальшого кількісного аналізу його продуктивності та масштабованості у наступному розділі роботи.

## **Висновки до розділу 2**

У другому розділі проведено формалізацію подієво-орієнтованої мікросервісної системи як стохастичної динамічної системи з дискретним часом, функціонування якої визначається процесами генерації, доставки, впорядкування та обробки подій у розподіленому середовищі. Запропонована формалізована модель враховує асинхронний характер взаємодії сервісів,

можливість повторної доставки повідомлень, виникнення розривів у послідовності подій та обмеження обчислювальних ресурсів. Введено систему формальних залежностей, що описують локальні послідовності подій, функцію стану логічного контексту, умови коректності обробки, параметри навантаження, часові характеристики та показники відновлення порушеного порядку, що дозволило сформуванню цілісного математичного представлення процесів функціонування подієво-орієнтованих систем доставки повідомлень.

На основі проведеної формалізації розроблено метод упорядкованого паралелізму в межах логічного ключа, який забезпечує поєднання локальної впорядкованості та глобальної паралельності обробки подій. На відміну від традиційних підходів, запропонований метод реалізує контроль послідовності на прикладному рівні поверх брокера повідомлень за рахунок використання маркерів послідовності, внутрішніх черг впорядкування, механізмів буферизації та відновлення пропущених станів. Формалізовано правила прийняття рішень щодо обробки, буферизації або ігнорування подій залежно від поточного стану логічного контексту, що дозволило забезпечити детерміновану еволюцію станів системи навіть за умов асинхронної доставки повідомлень та семантики *at-least-once*.

Вперше розроблено аналітичну модель функціонування системи впорядкованої доставки подій, яка встановлює взаємозв'язок між інтенсивністю надходження повідомлень, швидкістю їх обробки, рівнем паралелізму, накладними витратами на підтримання впорядкованості та часовими характеристиками системи. Отримані співвідношення дозволили формалізувати вплив порушень порядку доставки, процедур *replay*-відновлення та механізмів буферизації на пропускну здатність і латентність системи, а також визначити умови стабільного функціонування за різних режимів навантаження. Проведений аналіз показав існування суперечності між підвищенням рівня паралелізму та зростанням накладних витрат на підтримання коректного порядку виконання подій, що обґрунтовує доцільність застосування запропонованого методу упорядкованого паралелізму.

Сформовані у розділі математичні моделі, алгоритмічні залежності та структурні рішення створюють теоретичне підґрунтя для подальшого експериментального дослідження запропонованого методу, оцінювання його ефективності в умовах різного рівня навантаження та порівняння з існуючими підходами до організації доставки подій у мікросервісних системах.

## **РОЗДІЛ 3. АНАЛІЗ ТА АНАЛІТИЧНЕ МОДЕЛЮВАННЯ СИСТЕМ УПОРЯДКОВАНОЇ ДОСТАВКИ ПОДІЙ**

### **3.1. Огляд сучасних підходів до аналізу продуктивності подієво-орієнтованих систем**

Сучасні подієво-орієнтовані системи (Event-Driven Architecture, EDA) займають провідне місце серед архітектурних підходів до побудови розподілених програмних систем. Їх широке застосування обумовлене здатністю забезпечувати слабке зв'язування компонентів, горизонтальну масштабованість і підвищену відмовостійкість – властивості, що є критично важливими для сучасних корпоративних і хмарних застосувань, систем реального часу, фінансових платформ та систем Інтернету речей. Взаємодія між сервісами у таких архітектурах здійснюється через асинхронну передачу потоків подій за допомогою брокерів повідомлень або платформ потокової обробки даних.

Незважаючи на значну кількість наукових публікацій, присвячених окремим аспектам функціонування подієво-орієнтованих систем, питання комплексного аналітичного моделювання їх продуктивності з урахуванням специфічних механізмів залишається недостатньо вивченим. Аналіз сучасних підходів дозволяє виділити кілька ключових напрямів досліджень, кожен з яких охоплює певний аспект проблематики.

Одним з базових напрямів досліджень є формування детермінованого порядку подій у паралельних системах. Зокрема, у роботах McGlohon та Carothers розглянуто проблему неупередженого детермінованого тотального впорядкування подій у паралельному дискретно-подієвому моделюванні. Автори показують, що навіть за одночасного настання подій питання коректного визначення порядку їх виконання є критичним для відтворюваності результатів симуляції. Систематизацію підходів до детермінованого впорядкування подій у паралельних системах моделювання здійснено в подальших роботах тих самих авторів. Ці дослідження підтверджують фундаментальну важливість проблеми

впорядкованості, однак їх основний акцент – детермінізм симуляційних систем, а не аналітичне оцінювання продуктивності потокових мікросервісних архітектур у реальних умовах.

Ключовим обмеженням зазначених підходів є те, що вони не враховують ефектів, характерних саме для виробничих систем: нестабільності мережевого середовища, повторної доставки повідомлень внаслідок збоїв та необхідності ідемпотентної обробки. Крім того, в задачах дискретно-подієвого моделювання порядок подій є строго детермінованим вхідним параметром, тоді як у мікросервісних системах він є результатом конкурентної взаємодії компонентів і принципово не може бути заздалегідь заданим.

Інший напрям представлений роботами, присвяченими алгоритмічній оптимізації паралельного виконання подій. Andelfinger та інші запропонували алгоритм Window Racer для паралельного дискретно-подієвого моделювання, орієнтований на підвищення продуктивності через ефективне керування вікном обробки подій та мінімізацію часу простою обробників. Eker та інші дослідили динамічну синхронізацію логічного часу з урахуванням поточного навантаження системи, що дозволяє адаптивно налаштовувати рівень паралелізму залежно від інтенсивності вхідного потоку.

Bachan та інші представили фреймворк Devastator для сучасного паралельного дискретно-подієвого моделювання, орієнтований на максимальну продуктивність і масштабованість на багатоядерних архітектурах, підтверджує, що збільшення кількості паралельних виконавців саме по собі не гарантує лінійного зростання продуктивності, оскільки з'являються витрати на координацію, планування та підтримку коректного порядку подій. Однак, як і в інших роботах цього напрямку, головний акцент зроблено на архітектурних і системних аспектах реалізації платформи, а не на побудові узагальненої аналітичної моделі.

Спільним обмеженням цього напрямку є орієнтація на системи паралельного дискретно-подієвого моделювання, де обробники конкурують за спільний глобальний стан симуляції. У мікросервісних системах з ключами впорядкування конкуренція є локальною – в межах одного ключа – що



кардинально змінює характер накладних витрат і вимагає окремого аналітичного апарату.

Важливий клас досліджень стосується впорядкованого виконання транзакцій у розподілених транзакційних середовищах. Poudel та інші розглянули впорядковане планування, розподілена транзакційна пам'ять з керуванням потоком, де впорядкування операцій використовується для забезпечення коректності виконання паралельних транзакцій. У розвитку цього напрямку ті самі автори дослідили обробку розподілених транзакцій у заздалегідь визначеному порядку, що є концептуально близьким до задачі збереження послідовності подій у системах доставки повідомлень.

Перечислені роботи мають важливе методологічне значення, оскільки демонструють, що впорядкування операцій є необхідною умовою коректного функціонування розподілених систем. Зокрема, встановлено, що механізми впорядкування транзакцій неминуче супроводжуються додатковими накладними витратами на синхронізацію та підтримку узгодженості стану. Разом з тим дані підходи орієнтовані переважно на транзакційні механізми з властивостями ACID, а не на аналітичне моделювання пропускну здатності систем подієвої взаємодії з повторною доставкою повідомлень і семантикою *eventual consistency*.

Batista та інші запропонували протокол FlexCast для ефективної атомарної multicast-доставки у розподілених системах. Хоча ця робота стосується іншого класу систем, вона демонструє принципово важливий висновок: підтримка узгодженості та впорядкованості повідомлень у розподіленому середовищі неминуче супроводжується додатковими накладними витратами на координацію. Цей висновок є принципово важливим для побудови моделі продуктивності систем *ordered delivery*.

Окремий напрям охоплює дослідження відмовостійкості та безперервності функціонування подієво-орієнтованих систем. Bhupatiraju розглянув подієво-орієнтований підхід в контексті резервування і відмовостійкості фінансових платформ, підкресливши роль резервних каналів доставки та механізмів *failover* у забезпеченні безперервності обробки транзакційних подій. Автор підкреслює, що затримки, пов'язані з перемиканням на резервний канал, та імовірність

використання такого каналу безпосередньо впливають на середній час відповіді системи. Проте в роботі відсутній формальний аналітичний опис впливу цих факторів на пропускну здатність системи залежно від кількості паралельних обробників.

Chejarla дослідив проблеми впорядкованої доставки подій у хмарно-орієнтованих системах управління замовленнями, спроектованих для обробки подій у мікросервісній архітектурі. Ці роботи демонструють практичну значущість проблеми впорядкованої доставки в реальних промислових системах та підкреслюють необхідність механізмів, що забезпечують коректну послідовність обробки при масштабуванні.

Ramisetty проаналізував можливості подієво-орієнтованих мікросервісів для досягнення надмалих затримок у хмарних потокових системах. Автор встановив, що продуктивність таких систем суттєво залежить від конфігурації обробників та характеристик потоку подій, однак запропонований підхід носить переважно практичний характер і не надає формальної аналітичної основи для прогнозування продуктивності.

Kassetty та Jain дослідили масштабовані подієво-орієнтовані мікросервісні архітектури для обробки платежів у реальному часі. Їх аналіз підкреслює, що при підвищеному навантаженні ефективність паралельної обробки знижується через витрати на координацію, але кількісна оцінка цього ефекту в роботі не наведена.

Значну увагу приділено також дослідженням, пов'язаним з патерном Saga як механізмом управління розподіленими транзакціями. Daraghmi та інші запропонували вдосконалення патерну Saga для розподілених транзакцій у мікросервісній архітектурі, продемонструвавши, що використання компенсаційних транзакцій дозволяє забезпечити eventual consistency навіть при часткових збоях. Bugaeva розглянула реалізацію патерну Saga із застосуванням шаблону Outbox, орієнтовану на забезпечення надійної доставки повідомлень та мінімізацію ризику дублювання.

Ці дослідження підтверджують, що механізми управління розподіленими транзакціями є невіддільною частиною систем впорядкованої доставки подій і безпосередньо впливають на їх продуктивність через витрати на підтримку стану,

перевірку умов завершення та виконання компенсаційних дій. Проте аналітична кількісна оцінка цих витрат у контексті загальної продуктивності системи залишається поза увагою зазначених досліджень.

Узагальнення результатів аналізу дозволяє зробити наступний підсумок. Існуючі дослідження формують теоретичне та прикладне підґрунтя для вивчення впорядкованої обробки подій у паралельних і розподілених середовищах. Вони охоплюють питання детермінованого впорядкування подій, алгоритмів паралельного дискретно-подієвого моделювання, впорядкованого виконання транзакцій, відмовостійкої подієво-орієнтованої архітектури та управління розподіленими транзакціями через патерн Saga.

Разом з тим більшість цих робіт або не орієнтована на мікросервісні системи з семантикою *at-least-once*, або не враховує сукупний вплив усіх характерних факторів: повторної доставки повідомлень, ідемпотентної обробки, буферизації подій поза порядком, резервних каналів передачі та нерівномірності розподілу навантаження між ключами подій. При цьому кожен з цих факторів окремо досліджений у літературі, проте відсутні роботи, що розглядають їх комплексний вплив на продуктивність у єдиній формалізованій аналітичній моделі.

Для наочного підтвердження виявленої прогалини результати огляду систематизовано у вигляді порівняльного аналізу розглянутих підходів за критеріями охоплення ключових факторів, що визначають продуктивність систем упорядкованої доставки подій з семантикою *at-least-once*. Критеріями порівняння обрано п'ять факторів: врахування повторної доставки повідомлень, механізм ідемпотентної обробки, буферизація подій поза порядком, наявність резервних каналів передачі та нерівномірність розподілу навантаження між ключами подій.

З таблиці видно, що жоден з розглянутих підходів не охоплює одночасно всі п'ять факторів. Найповнішим за охопленням є підхід до упорядкованої доставки в хмарних системах управління замовленнями [12], що враховує три з п'яти факторів, проте навіть він не включає резервний канал та нерівномірність розподілу навантаження між ключами.

Таблиця 3.1

Порівняльний аналіз існуючих підходів за критеріями охоплення факторів  
продуктивності систем упорядкованої доставки подій

Підхід	Повторна доставка	Ідемпотентність	Буферизація поза порядком	Резервний канал	Нерівно- мірність ключів
Детерміноване впорядкування в паралельному моделюванні [1, 10]	Ні	Ні	Частково	Ні	Ні
Алгоритмічна оптимізація паралельного виконання [2, 5, 9]	Ні	Ні	Так	Ні	Частково
Впорядковане виконання транзакцій [4, 6, 8]	Ні	Частково	Ні	Ні	Ні
Атомарна multicast- доставка [7]	Частково	Ні	Ні	Ні	Ні
Відмовостійкість подієво-орієнтованих систем [3]	Так	Ні	Ні	Так	Ні
Упорядкована доставка в хмарних системах управління замовленнями [12]	Так	Частково	Частково	Ні	Ні
Мікросервіси з надмалими затримками [13]	Ні	Ні	Ні	Ні	Так
Масштабовані архітектури для обробки платежів [14]	Ні	Ні	Ні	Ні	Так
Патерн Saga та механізм Outbox [15, 16]	Так	Так	Ні	Ні	Ні

Фактор буферизації поза порядком досліджений виключно в системах паралельного дискретно-подієвого моделювання, де його природа принципово відрізняється від мікросервісних архітектур: у системах моделювання порядок подій є вхідним параметром і відомий заздалегідь, тоді як у мікросервісних

системах він є результатом конкурентної взаємодії обробників у реальному часі. Нерівномірність розподілу навантаження між ключами у жодній з розглянутих робіт не отримала кількісної оцінки її впливу на ефективний рівень паралелізму, а лише фіксується як практична проблема. Підходи, що розглядають ідемпотентність, роблять це виключно з позиції коректності виконання, не аналізуючи витрати на її підтримку як вимірюваний внесок до загальної латентності.

Принципова складність полягає не у відсутності окремих досліджень, а у тому, що всі п'ять факторів взаємодіють між собою, і розгляд кожного з них окремо не дає коректної картини. Повторні доставки збільшують фактичне навантаження на обробники та одночасно підвищують частоту звернень до механізму ідемпотентності. Порушення порядку, спричинене паралельною обробкою в поєднанні з повторними доставками, збільшує потребу в буферизації нелінійним чином. Нерівномірність ключів знецінює горизонтальне масштабування навіть при ідеальній роботі решти механізмів, оскільки перевантажений ключ серіалізує виконання незалежно від кількості доступних обробників. Модель, що розглядає ці фактори ізольовано, систематично недооцінює сукупні накладні витрати і не може слугувати надійним інструментом прогнозування.

З огляду на це, завданням даного розділу є побудова аналітичної моделі, що встановлює функціональну залежність між рівнем паралелізму та пропускнуою здатністю системи упорядкованої доставки подій з семантикою *at-least-once*, з одночасним урахуванням усіх п'яти зазначених факторів у єдиному виразі. Модель має забезпечувати можливість аналітичного визначення оптимальної кількості паралельних обробників, при якій досягається максимальна пропускна здатність, та ідентифікації абсолютної межі масштабованості системи, яка не може бути подолана збільшенням кількості обробників без зміни архітектурних параметрів. Практичним наслідком такої моделі є можливість на етапі проектування, без проведення повного циклу навантажувального тестування, визначити який саме з п'яти факторів є домінуючим обмеженням для конкретної системи і на усунення якого з них слід спрямовувати зусилля з оптимізації в

першу чергу. Це скорочує час і вартість підбору конфігурації у виробничих системах, де повторне тестування при кожній зміні параметрів є ресурсомістким і не завжди здійсненим в умовах виробничого навантаження.

### **3.2. Аналіз впливу паралелізму на латентність та пропускну здатність**

Паралельна обробка подій є одним із ключових засобів підвищення пропускну здатності мікросервісних систем. Проте взаємозв'язок між кількістю паралельних обробників та фактичною продуктивністю системи є нетривіальним і суттєво відрізняється від поведінки класичних паралельних обчислювальних систем. У системах впорядкованої доставки подій збільшення рівня паралелізму одночасно породжує кілька конкуруючих ефектів: з одного боку підвищує теоретичну пропускну здатність, з іншого - збільшує накладні витрати, пов'язані з буферизацією, синхронізацією та підтримкою коректного порядку виконання. Розуміння природи цих ефектів є необхідною передумовою для побудови адекватної аналітичної моделі.

У найпростішому випадку, коли відсутні будь-які накладні витрати і система масштабується ідеально, пропускну здатність при  $k$  обробниках лінійно зростає пропорційно до їх кількості. Кожен додатковий обробник вносить рівний внесок у загальну продуктивність системи, а загальний час обробки однієї події залишається незмінним незалежно від кількості паралельних потоків. Така ідеальна лінійна залежність слугує теоретичною верхньою межею продуктивності і використовується як базова лінія для оцінювання ефективності реальних систем.

На практиці пропускну здатність системи завжди виявляється нижчою за ідеальну через наявність різноманітних накладних витрат. Характер відхилення від ідеальної кривої залежить від конкретних механізмів, що діють у системі, і є індивідуальним для кожного класу архітектур. Для систем впорядкованої доставки подій з семантикою *at-least-once* цей характер визначається сукупністю п'яти основних факторів, детально розглянутих у наступних підрозділах.

Семантика доставки at-least-once гарантує, що кожна подія буде доставлена щонайменше один раз, але допускає її повторне надходження внаслідок мережевих збоїв, перезапусків компонентів або тайм-аутів підтвердження. Повторні доставки безпосередньо впливають на латентність і пропускну здатність системи через два механізми.

По-перше, в системах з семантикою at-least-once одна й та сама подія може доставлятися кілька разів. Середня кількість доставок однієї події визначається як:

$$E[N_d] = \frac{1}{1 - p_r} \quad (3.1)$$

де  $p_r$  – імовірність повторної доставки. Фактична інтенсивність потоку подій, що надходить до обробників, зростає і становить:

$$\lambda_{eff} = \frac{\lambda}{1 - p_r}, \quad (3.2)$$

При  $p_r = 0,1$  фактична інтенсивність перевищує номінальну на 11%, при  $p_r = 0,3$  – вже на 43%. Таким чином, навіть помірний рівень повторних доставок суттєво збільшує навантаження на обробники та зменшує ефективний запас пропускну здатності системи.

По-друге, обробка повторних доставок потребує виконання механізму дедуплікації для кожної події, незалежно від того, чи є вона дублікатом. Середній внесок повторної доставки до часу обробки:

$$T_r = \frac{\beta p_r}{1 - p_r}, \quad (3.3)$$

де  $\beta$  – коефіцієнт часових витрат на повторну обробку. Нелінійна залежність  $T_r$  від  $p_r$  означає, що зі зростанням частоти повторних доставок цей ефект посилюється непропорційно швидко і може стати домінуючим чинником зниження продуктивності при високому значенні  $p_r$ .

Ідемпотентна обробка передбачає перевірку кожної події на предмет того, чи вона вже була оброблена раніше. У запропонованому методі ця перевірка здійснюється шляхом порівняння порядкового номера події  $E.seq$  з останнім

успішно обробленим значенням  $last_{seq}[key]$  для відповідного ключа. Якщо  $E.seq \leq last_{seq}[key]$ , подія ідентифікується як дублікат і відхиляється без передачі до бізнес-логіки.

Середній час такої перевірки  $T_i$  є відносно невеликим для кожної окремої події, однак виконується для абсолютно кожної отриманої події - як унікальної, так і повторної. При великій інтенсивності потоку сукупні витрати на ідемпотентну обробку стають суттєвим внеском до загальної латентності системи. Крім того, перевірка ідемпотентності вимагає доступу до сховища стану  $last_{seq}[key]$ , що при паралельній обробці може ставати вузьким місцем через конкуренцію за спільний ресурс.

При паралельній обробці з кількома обробниками події, що належать до одного ключа впорядкування, можуть надходити на обробку не в порядку своїх порядкових номерів. Це відбувається тому, що різні обробники виконують свої завдання з різною швидкістю і різними затримками мережевої доставки. Для відновлення коректного порядку виконання кожна подія, що надійшла з порядковим номером, більшим за очікуване значення  $last_{seq}[key] + 1$ , повинна бути затримана у буфері до моменту отримання та обробки всіх попередніх подій.

Середня затримка буферизації описується залежністю:

$$T_b = \alpha \cdot k \cdot CV \quad (3.4)$$

де  $\alpha$  – коефіцієнт чутливості системи до порушення порядку,  $k$  – кількість обробників,  $CV = \sigma/T_p$  – коефіцієнт варіації часу обробки.

Залежність відображає два ключових аспекти. По-перше, зі збільшенням кількості обробників  $k$  ймовірність виходу подій з порядку зростає, оскільки зростає різноманітність шляхів, якими події можуть надходити до обробників. По-друге, чим більша мінливість часу обробки (тобто чим більший  $CV$ ), тим частіше виникають ситуації, коли пізніша за порядком подія виявляється обробленою раніше за попередню.

Важливо, що затримка буферизації є лінійною відносно  $k$ , тоді як теоретичне прискорення від паралелізму є сублінійним (у разі наявності послідовної компоненти) або лінійним (у разі ідеальної паралелізації). Це



означає, що при достатньо великому  $k$  витрати на буферизацію можуть повністю нівелювати виграш від паралельної обробки, визначаючи практичну верхню межу ефективного рівня паралелізму.

У системах з резервним каналом доставки - наприклад, при використанні HTTP як резервного транспорту при недоступності брокера повідомлень – частина подій доставляється через канал з іншими характеристиками затримки. Якщо  $p_h$  – ймовірність використання резервного каналу, а  $T_h$  – середня затримка передачі через нього, середній внесок цього механізму до загальної латентності:

$$T_{h,eff} = p_h \cdot T_h. \quad (3.5)$$

Хоча резервний канал у цілому має вищу затримку порівняно з основним, його вплив на загальну продуктивність системи є помірним за умови відносно низької ймовірності  $p_h$ . Разом з тим присутність резервного каналу принципово змінює характер надійності доставки: ймовірність успішної доставки при двоканальній архітектурі визначається як  $P_{deliver} = 1 - p_{loss} \cdot p_{loss}^{http}$ , що є значно вищою за аналогічний показник однока-нальної системи.

Зі збільшенням кількості паралельних обробників зростає інтенсивність конкуренції за спільні ресурси системи – сховище стану  $last\_seq[key]$ , структури даних буфера впорядкування та канали міжпроцесної взаємодії. Середні витрати часу на синхронізацію доступу до цих ресурсів:

$$T_c = \gamma \cdot k, \quad (3.6)$$

де  $\gamma$  – коефіцієнт конкуренції, що характеризує середні витрати на одиницю рівня паралелізму. Лінійна залежність  $T_c$  від  $k$  є спрощеним, але достатньо адекватним наближенням для помірного рівня паралелізму. При дуже великій кількості обробників реальна залежність може бути більш крутою через ефекти сильної конкуренції (contention), що виходить за межі лінійного режиму.

Практичні системи майже завжди характеризуються нерівномірним розподілом подій між ключами впорядкування: деякі ключі генерують значно більшу кількість подій, ніж інші. Такі "гарячі" ключі стають вузькими місцями, оскільки для кожного ключа гарантується серіалізоване виконання, і при дуже

активному ключі один обробник виявляється перевантаженим, тоді як інші простоюють.

Для врахування цього ефекту вводиться коефіцієнт нерівномірності  $s = \lambda_{max}/\lambda_{avg}$ , де  $\lambda_{max}$  – максимальна інтенсивність потоку для одного ключа,  $\lambda_{avg}$  – середня інтенсивність на ключ. При  $s = 1$  навантаження рівномірне і вся пропускна здатність системи задіяна ефективно. При  $s > 1$  ефективна кількість паралельних обробників знижується до  $k_{eff} = k/s$ , відображаючи реальний ступінь використання паралельних ресурсів.

Нерівномірність навантаження є особливо актуальною проблемою для систем, де ключі впорядкування визначаються за ідентифікатором транзакції, користувача або замовлення: активні користувачі або популярні товари можуть генерувати на порядок більше подій, ніж середній ключ, що призводить до суттєвого знецінення горизонтального масштабування.

Результати аналізу впливу окремих факторів на латентність і пропускну здатність системи узагальнено у таблиці 3.2.

Аналіз таблиці показує, що фактори можна розділити на два класи: константні відносно  $k$  (повторна доставка, ідемпотентність, резервний канал) та залежні від  $k$  (буферизація, конкуренція за ресурси). Саме фактори другого класу визначають межу ефективного масштабування системи і зумовлюють існування оптимального рівня паралелізму. При малих значеннях  $k$  домінує вигреш від паралелізму, при великих – накладні витрати, пов'язані з буферизацією і синхронізацією, нівелюють цей вигреш і можуть призводити до зниження загальної пропускної здатності.

Поділ факторів на два класи має безпосередній наслідок для розуміння природи масштабування системи. Константні фактори зміщують криву пропускної здатності вниз відносно ідеальної, але не змінюють її характер: система все одно масштабується, хоча й з нижчою початковою точкою. Натомість фактори, що лінійно зростають з  $k$ , змінюють сам характер кривої, перетворюючи її з монотонно зростаючої на таку, що після певного значення  $k$  дає дедалі менший приріст пропускної здатності на кожен доданий обробник. Саме це явище, а не просто наявність накладних витрат як таких, є принциповою

відмінністю систем впорядкованої доставки від класичних паралельних обчислювальних систем.

Таблиця 3.2.

Фактори впливу паралелізму на продуктивність системи впорядкованої доставки подій

Фактор	Позначення	Характер залежності від $k$	Вплив на латентність	Вплив на пропускну здатність
Повторна доставка (вплив на потік)	$\lambda_{eff}$	Не залежить від $k$	Середній	Знижує запас пропускну здатності
Повторна доставка (витрати обробки)	$T_r$	Не залежить від $k$	Середній	Знижує ефективний $\mu$
Ідемпотентна обробка	$T_i$	Не залежить від $k$	Малий	Знижує ефективний $\mu$
Буферизація для впорядкування	$T_b = \alpha k \cdot CV$	Лінійна	Зростає з $k$	Обмежує масштабованість
Резервний канал доставки	$T_{h,eff} = p_h T_h$	Не залежить від $k$	Малий-середній	Незначний при малому $p_h$
Конкуренція за ресурси	$T_c = \gamma k$	Лінійна	Зростає з $k$	Обмежує масштабованість
Нерівномірність ключів	$k_{eff} = k/s$	Знижує ефективне $k$	Опосередкований	Суттєво знижує при великому $s$

Для кількісної оцінки цього явища введемо коефіцієнт ефективності паралелізму  $E_p(k)$ , що визначає яка частка теоретичної пропускну здатності реалізується на практиці при заданому рівні паралелізму  $k$ . Оскільки теоретична пропускну здатність зростає лінійно з  $k$ , а фактична зростає повільніше через лінійно зростаючі накладні витрати, коефіцієнт  $E_p(k)$  є монотонно спадаючою

функцією від  $k$ . Це означає, що кожен наступний доданий обробник дає менший відносний приріст, ніж попередній, і починаючи з певного значення  $k$  приріст стає настільки малим, що подальше збільшення кількості обробників є економічно та технічно невиправданим.

Для ілюстрації цього ефекту розглянемо характерну поведінку системи при зміні  $k$  у трьох режимах. При малих значеннях  $k$ , коли константні накладні витрати домінують над лінійно зростаючими, система поводить себе близько до ідеальної: кожен новий обробник дає приріст пропускної здатності, близький до  $1/T_p$ . При помірних значеннях  $k$  лінійно зростаючі складові стають порівнянними з константною частиною, і ефективність кожного наступного обробника помітно знижується. При великих значеннях  $k$  лінійно зростаючі накладні витрати починають домінувати у знаменнику виразу для пропускної здатності, і крива фактично виходить на горизонтальну асимптоту: система більше не реагує на збільшення кількості обробників скільки-небудь значущим зростанням продуктивності.

Описана асимптотична поведінка є одним з ключових результатів аналізу і принципово відрізняє запропоновану модель від моделі Амдала. У ній асимптота визначається часткою послідовних обчислень і не залежить від кількості обробників у знаменнику. У системах впорядкованої доставки подій асимптота визначається коефіцієнтами буферизації та конкуренції, які є властивостями архітектури системи, а не вхідного алгоритму. Тобто, для підвищення межі масштабованості недостатньо зменшити послідовну частину обчислень, як у класичному випадку, а необхідно цілеспрямовано знижувати коефіцієнти  $\alpha$  та  $\gamma$  через архітектурні рішення: оптимізацію структур даних буфера впорядкування, зменшення конкуренції за сховище стану `lastseqkey` через партиціонування або локальне кешування, вибір схеми хешування ключів, що забезпечує рівномірний розподіл навантаження.

Окремої уваги потребує взаємодія між факторами при одночасному їх впливі. Розглянутий вище аналіз трактував кожен фактор незалежно, що є спрощенням. Насправді між факторами існують залежності, що підсилюють загальний ефект. Підвищений рівень повторних доставок  $p_r$  не лише збільшує

фактичну інтенсивність потоку  $\lambda_{eff}$ , але й опосередковано збільшує ймовірність порушення порядку подій: якщо одна й та сама подія доставляється повторно з певною затримкою, вона з більшою ймовірністю надходить до обробника після того, як наступна за порядком подія вже почала оброблятися, що безпосередньо збільшує середню тривалість очікування в буфері. Таким чином, при високому  $p_r$  фактичне значення затримки буферизації  $T_b$  є більшим, ніж дає формула  $T_b = \alpha * k * CV$  при номінальних параметрах, тобто формула дає нижню оцінку реальних витрат на буферизацію в умовах значного рівня повторних доставок.

Аналогічна взаємодія існує між нерівномірністю ключів та конкуренцією за ресурси. При нерівномірному розподілі навантаження "гарячий" ключ спричиняє непропорційно часті звернення до сховища стану `lastseqkey`, що локально підвищує конкуренцію за цей ресурс понад рівень, який дає коефіцієнт  $\gamma$  при рівномірному розподілі. Отже, при великому  $s$  фактичні витрати на синхронізацію для перевантаженого ключа є вищими за  $T_c = \gamma * k$ , а загальна деградація продуктивності є більшою, ніж можна було б очікувати, розглядаючи нерівномірність і конкуренцію ізольовано.

Ці спостереження мають важливий практичний висновок: у реальних системах з одночасно високим  $p_r$  і великим  $s$  погіршення продуктивності при збільшенні  $k$  є більш різким, ніж передбачає лінійне наближення для кожного фактора окремо. Запропонована модель використовує лінійні апроксимації для кожного фактора, що є обґрунтованим для помірних значень параметрів і помірного рівня паралелізму, однак при плануванні систем з екстремальними значеннями  $p_r$  або  $s$  необхідно враховувати, що модель дає консервативну, тобто оптимістичну оцінку реальної пропускної здатності. Це означає, що якщо модель вказує на нестабільність системи при певній конфігурації, система буде нестабільною з ще більшою впевненістю. Якщо ж модель вказує на стабільність із невеликим запасом, цей запас у реальній системі може виявитися меншим за розрахунковий через ефекти взаємного підсилення факторів.

### 3.3. Формалізація параметрів системи та потоків подій

Для побудови аналітичної моделі продуктивності системи впорядкованої доставки подій необхідно насамперед чітко визначити математичний апарат, що описує як характеристики вхідного потоку подій, так і параметри обробки. Формалізація параметрів є необхідним підґрунтям, що забезпечує однозначність подальших аналітичних виразів та можливість їх практичного застосування для оцінювання та оптимізації конфігурації системи. Вхідний потік подій характеризується номінальною інтенсивністю генерації  $\lambda$  – середньою кількістю нових унікальних подій, що надходять у систему за одиницю часу. Оскільки система функціонує з семантикою доставки at-least-once, частина подій доставляється повторно внаслідок мережових збоїв, тайм-аутів або перезапусків компонентів. Імовірність повторної доставки позначається  $p_r \in [0,1)$ , а фактична інтенсивність потоку подій, що надходять до обробників з урахуванням повторних доставок, визначається як:

$$\lambda_{eff} = \frac{\lambda}{1 - p_r} \quad (3.7)$$

Множина ключів впорядкування  $K$  визначає логічну партиціонованість потоку подій: події з різними ключами можуть оброблятися незалежно і паралельно, тоді як події з однаковим ключем мають виконуватися строго послідовно. Кількість ключів  $m = |K|$  визначає теоретичний рівень паралелізму системи. Для кожного ключа  $key \in K$  підтримується змінна стану  $last_{seq}[key]$ , що фіксує порядковий номер останньої успішно обробленої події в межах даного ключа. Параметри обробки включають середній базовий час обробки однієї події  $T_p$  – час виконання бізнес-логіки за відсутності будь-яких накладних витрат – та стандартне відхилення  $\sigma$ , що характеризує мінливість часу обробки. На основі цих величин визначається коефіцієнт варіації часу обробки:

$$CV = \frac{\sigma}{T_p}, \quad (3.8)$$

який є безрозмірним показником відносної мінливості й відіграє ключову роль у визначенні затримок буферизації при паралельній обробці. Базова швидкість обслуговування одного обробника визначається як  $\mu = 1/T_p$  і характеризує продуктивність за відсутності накладних витрат. Кількість паралельних обробників  $k$  є одним із найважливіших конфігураційних параметрів системи. Теоретична пропускна здатність при  $k$  обробниках без урахування накладних витрат:

$$X_{ideal}(k) = \frac{k}{T_p} \quad (3.9)$$

Ця величина є верхньою теоретичною межею продуктивності й слугує базовою лінією для оцінювання ефективності реальної системи. Параметри механізмів надійності описують часові витрати, пов'язані з підтримкою коректності обробки. Середній час перевірки ідемпотентності  $T_i$  відображає витрати на дедуплікацію – перевірку того, чи була подія вже оброблена раніше. Коефіцієнт  $\beta$  характеризує відносні витрати часу на повторну обробку, а середній внесок повторних доставок до часу обробки:

$$T_r = \frac{\beta \cdot p_r}{1 - p_r} \quad (3.10)$$

Ця величина нелінійно зростає зі збільшенням  $p_r$ : при значеннях  $p_r$ , що наближаються до одиниці,  $T_r$  необмежено зростає, що відображає катастрофічне навантаження від надмірної кількості повторних доставок. Середня затримка буферизації, необхідна для відновлення правильного порядку подій при паралельній обробці:

$$T_b = \alpha \cdot k \cdot CV \quad (3.11)$$

де  $\alpha$  – коефіцієнт чутливості системи до порушення порядку подій, що залежить від конкретної реалізації буферного механізму та топології системи.

Важливо, що  $T_b$  лінійно зростає з кількістю обробників  $k$  та коефіцієнтом варіації  $CV$ , відображаючи той факт, що при більшій кількості паралельних потоків і більшій мінливості часу обробки ймовірність порушення порядку зростає і потребує більш тривалої буферизації.

Параметри резервного каналу доставки включають імовірність використання резервного каналу  $p_h$  – частку подій, що доставляються через нього замість основного – та середню затримку передачі через резервний канал  $T_h$ . Середній внесок резервного каналу до загальної латентності:

$$T_{h,eff} = p_h \cdot T_h \quad (3.12)$$

Коефіцієнт конкуренції за спільні ресурси  $\gamma$  характеризує середні витрати часу на синхронізацію при одночасному доступі кількох обробників до спільних структур даних – зокрема, сховища стану  $last_{seq}[key]$  та буферних черг. Відповідні витрати:

$$T_c = \gamma \cdot k \quad (3.13)$$

Параметри нерівномірності навантаження описують розподіл подій між ключами. Максимальна інтенсивність потоку для одного ключа  $\lambda_{max}$  та середня інтенсивність на ключ

$$\lambda_{avg} = \frac{\lambda}{m}, \quad (3.14)$$

визначають коефіцієнт нерівномірності:

$$s = \frac{\lambda_{max}}{\lambda_{avg}}. \quad (3.15)$$

При  $s = 1$  навантаження рівномірне між усіма ключами, при  $s > 1$  окремі ключі є "гарячими" і генерують суттєво більше подій за середнє.

Ефективна кількість паралельних обробників з урахуванням нерівномірності:

$$k_{eff} = \frac{k}{s}. \quad (3.16)$$

Метрики ефективності системи включають коефіцієнт ефективності паралелізму

$$E_p(k) = \frac{X_g(k)}{X_{ideal}(k)} \quad (3.17)$$

де  $X_g(k)$  – фактична пропускна здатність системи, визначення якої наводиться у підрозділі 3.4.



Коефіцієнт ефективності показує, яка частина теоретичної пропускну здатності реалізується на практиці, і знаходиться у діапазоні  $(0,1]$ . Коефіцієнт впорядкованості:

$$O = \frac{N_{ordered}}{N_{total}}, \quad (3.18)$$

де  $N_{ordered}$  – кількість подій, оброблених без порушення порядку;  $N_{total}$  – загальна кількість подій, характеризує коректність роботи механізму впорядкування.

Для коректно налаштованої системи  $O \rightarrow 1$  навіть при наявності повторних доставок і часткових збоїв, завдяки механізму буферизації та ідемпотентної обробки.

Коефіцієнт завантаження системи

$$\rho = \frac{\lambda_{eff}}{X_g(k)}, \quad (3.19)$$

є фундаментальним показником стійкості: при  $\rho < 1$  система функціонує у стійкому режимі, при  $\rho \geq 1$  черга подій необмежено зростає і система стає нестійкою.

Для зручності використання при аналітичних розрахунках усі введені параметри систематизовано у таблиці 3.3.

Введена система позначень забезпечує формальну основу для побудови аналітичної моделі в наступному підрозділі.

Таблиця 3.3 містить усі параметри у їх формальному вигляді, однак для практичного застосування моделі важливо розуміти не лише що означає кожен параметр, але й яким чином він може бути визначений для конкретної системи. Параметри моделі поділяються на три категорії за способом їх отримання: параметри, що вимірюються безпосередньо, параметри, що оцінюються через профілювання, та параметри, що визначаються через спостереження за поведінкою системи в цілому.

Зведена таблиця параметрів аналітичної моделі

Позначення	Найменування	Одиниці виміру
$\lambda$	Номінальна інтенсивність генерації подій	подій/с
$p_r$	Імовірність повторної доставки	безрозмірна
$\lambda_{eff}$	Фактична інтенсивність потоку	подій/с
$m$	Кількість ключів впорядкування	шт.
$K$	Множина ключів впорядкування	-
$T_p$	Середній базовий час обробки	с
$\sigma$	Стандартне відхилення часу обробки	с
$CV$	Коефіцієнт варіації часу обробки	безрозмірна
$\mu$	Базова швидкість обслуговування	подій/с
$k$	Кількість паралельних обробників	шт.
$T_i$	Час перевірки ідемпотентності	с
$\beta$	Коефіцієнт витрат на повторну обробку	безрозмірна
$T_r$	Внесок повторних доставок до часу обробки	с
$\alpha$	Коефіцієнт чутливості до порушення порядку	безрозмірна
$T_b$	Затримка буферизації	с
$p_h$	Імовірність використання резервного каналу	безрозмірна
$T_h$	Затримка резервного каналу доставки	с
$T_{h,eff}$	Середній внесок резервного каналу	с
$\gamma$	Коефіцієнт конкуренції за ресурси	безрозмірна
$T_c$	Витрати на синхронізацію	с
$\lambda_{max}$	Максимальна інтенсивність потоку на ключ	подій/с
$\lambda_{avg}$	Середня інтенсивність потоку на ключ	подій/с
$s$	Коефіцієнт нерівномірності навантаження	безрозмірна
$k_{eff}$	Ефективна кількість паралельних обробників	шт.
$X_{ideal}(k)$	Ідеальна пропускна здатність	подій/с
$X_g(k)$	Фактична пропускна здатність	подій/с
$E_p(k)$	Коефіцієнт ефективності паралелізму	безрозмірна
$O$	Коефіцієнт впорядкованості	безрозмірна
$\rho$	Коефіцієнт завантаження системи	безрозмірна

До першої категорії належать параметри, значення яких може бути отримано прямим вимірюванням без будь-яких припущень про внутрішню структуру системи. Номінальна інтенсивність генерації подій  $\lambda$  вимірюється на рівні джерел подій як кількість унікальних подій за одиницю часу у стаціонарному режимі роботи. Імовірність повторної доставки  $p_r$  оцінюється як частка повторних доставок у загальній кількості отриманих подій за достатньо тривалий проміжок часу спостереження. Середній базовий час обробки  $T_p$  та стандартне відхилення  $\sigma$  вимірюються через профілювання бізнес-логіки обробника в ізольованих умовах, тобто без паралельного навантаження на спільні ресурси. Параметри резервного каналу  $p_h$  та  $T_h$  вимірюються відповідно як частота перемикавання на резервний канал та середня затримка передачі через нього при контрольованих тестових умовах.

До другої категорії належать коефіцієнти  $\alpha$ ,  $\beta$  та  $\gamma$ , що характеризують витрати конкретних механізмів реалізації. Коефіцієнт  $\beta$ , що визначає відносні витрати часу на повторну обробку, оцінюється як відношення середнього часу обробки повторно доставленої події до середнього базового часу обробки  $T_p$ . Коефіцієнт  $\gamma$ , що характеризує витрати на синхронізацію при конкуренції за ресурси, визначається як нахил залежності часу доступу до сховища стану `lastseqkey` від кількості паралельних потоків, що одночасно здійснюють до нього звернення. Коефіцієнт  $\alpha$ , що відображає чутливість системи до порушення порядку подій, є найскладнішим для прямого вимірювання, оскільки залежить від топології системи, дисципліни планування обробників та характеристик мережевого середовища. Його практична оцінка здійснюється через вимірювання середньої затримки буферизації при відомих значеннях  $k$  та  $CV$  з подальшим обчисленням  $\alpha$ .

До третьої категорії належать параметри нерівномірності навантаження. Максимальна інтенсивність потоку на один ключ  $\lambda_{max}$  та середня інтенсивність  $\lambda_{avg}$  визначаються через статистичний аналіз розподілу подій за ключами у виробничому або тестовому навантаженні. Коефіцієнт нерівномірності  $s$  є

похідним від цих двох величин і відображає ступінь концентрації навантаження на окремих ключах. У системах з рівномірним розподілом  $s$  наближається до одиниці, тоді як у системах з явно вираженими "гарячими" ключами  $s$  може перевищувати 5 і навіть 10, що суттєво знижує ефективну кількість паралельних обробників  $k_{eff} = k/s$  та робить горизонтальне масштабування малоефективним без одночасного вирішення проблеми нерівномірності.

Між параметрами існують функціональні обмеження, що є необхідними умовами коректності моделі. Імовірність повторної доставки  $p_r$  має знаходитись у діапазоні  $[0, 1)$ , оскільки значення  $p_r = 1$  означало б нескінченну кількість повторних доставок. Коефіцієнт варіації  $CV$  є невід'ємним за визначенням; при  $CV = 0$  система має строго детермінований час обробки і затримка буферизації  $T_b$  дорівнює нулю незалежно від  $k$ , що є граничним випадком повністю передбачуваного навантаження. Коефіцієнт нерівномірності  $s$  задовольняє умові  $s \geq 1$  за визначенням, оскільки максимальна інтенсивність на ключ не може бути меншою за середню. Кількість ключів  $m$  має перевищувати або дорівнювати кількості обробників  $k$  для того, щоб паралельна обробка мала сенс: при  $m < k$  частина обробників завжди простоюватиме через відсутність ключів для обробки.

Описані обмеження формують область допустимих значень параметрів моделі та визначають умови, при яких отримані аналітичні результати є фізично змістовними. Вихід за межі цієї області не означає математичну некоректність виразів, але означає, що відповідна конфігурація не реалізовна на практиці або є граничним виродженим випадком.

### **3.4. Побудова аналітичної моделі системи доставки подій**

На основі формалізованих у попередньому підрозділі параметрів побудуємо узагальнену аналітичну модель продуктивності системи впорядкованої доставки подій з семантикою *at-least-once*. Модель враховує комплекс факторів, що одночасно діють у реальній системі, і дозволяє отримати

аналітичний вираз для фактичної пропускної здатності як функції від рівня паралелізму та конфігураційних параметрів.

Відправною точкою побудови є визначення базової швидкості обслуговування одного обробника та ідеальної пропускної здатності системи. Базова швидкість обслуговування - тобто продуктивність одного обробника за відсутності будь-яких накладних витрат - визначається формулою  $\mu = 1/T_p$ . Відповідно, теоретична пропускна здатність при  $k$  паралельних обробниках без урахування накладних витрат становить:

$$X_{ideal}(k) = k \cdot \mu = \frac{k}{T_p}. \quad (3.20)$$

Ця величина відповідає ідеалізованому випадку повністю лінійного масштабування і слугує верхньою теоретичною межею продуктивності системи.

Наступним кроком є врахування семантики доставки at-least-once та механізму ідемпотентної обробки. Оскільки подія може бути доставлена кілька разів, фактична інтенсивність потоку, що надходить до обробників, перевищує номінальну і визначається як

$$\lambda_{eff} = \frac{\lambda}{1 - p_r}. \quad (3.21)$$

Крім того, кожна повторно доставлена подія потребує витрат часу на перевірку ідемпотентності та, у випадку ідентифікації як дублікату, на повторну обробку. Середній часовий внесок цього механізму до часу обробки однієї події описується формулою:

$$T_r = \frac{\beta p_r}{1 - p_r} \quad (3.22)$$

Паралельна обробка подій із кількома обробниками неминуче призводить до порушення початкового порядку надходження подій у межах одного ключа. Для відновлення коректного порядку виконання застосовується механізм буферизації, що затримує обробку поточної події до моменту отримання всіх попередніх. Середня затримка буферизації лінійно зростає із кількістю обробників і мінливістю часу обробки:

$$T_b = \alpha \cdot k \cdot CV \quad (3.23)$$

Цей член є одним із ключових факторів, що обмежують масштабованість системи, оскільки при збільшенні  $k$  витрати на буферизацію зростають, тоді як вииграш від паралелізму для кожного конкретного ключа залишається обмеженим.

У системах з резервним каналом доставки частина подій надходить через канал з підвищеною затримкою – наприклад, через НТТР при недоступності брокера повідомлень. Середній внесок цього механізму до загальної латентності:

$$T_{h,eff} = p_h \cdot T_h \quad (3.24)$$

При паралельній обробці кілька обробників одночасно звертаються до спільних структур даних – сховища стану  $last_{seq}[key]$  та буферних черг - що породжує конкуренцію за ресурси. Відповідні витрати на синхронізацію також лінійно зростають із кількістю обробників:

$$T_c = \gamma \cdot k \quad (3.25)$$

Сумарні накладні витрати системи утворюються як сума усіх перелічених компонентів:

$$T_{overhead}(k) = T_i + T_r + T_b + T_{h,eff} + T_c \quad (3.26)$$

Підставляючи розгорнуті вирази для кожного члена, отримуємо:

$$T_{overhead}(k) = T_i + \frac{\beta p_r}{1 - p_r} + \alpha k \cdot CV + p_h T_h + \gamma k \quad (3.27)$$

Структура цього виразу є принципово важливою: перші три члени  $T_i$ ,  $T_r$  та  $T_{h,eff}$  не залежать від кількості обробників  $k$ , тоді як члени  $T_b$  та  $T_c$  лінійно зростають із  $k$ . Саме ця лінійно зростаюча компонента накладних витрат є причиною існування практичної верхньої межі ефективного рівня паралелізму.

Ефективний середній час обробки однієї події з урахуванням усіх накладних витрат:

$$T_{eff}(k) = T_p + T_{overhead}(k) \quad (3.28)$$

Із підстановкою розгорнутого виразу для  $T_{overhead}(k)$

$$T_{eff}(k) = T_p + T_i + \frac{\beta p_r}{1 - p_r} + \alpha k \cdot CV + p_h T_h + \gamma k \quad (3.29)$$

Загальний алгоритм оцінювання продуктивності системи та пошуку оптимального рівня паралелізму наочно представлено на рис. 3.1 у вигляді блок-схеми, що відображає послідовність обчислювальних кроків від задання вхідних параметрів до перевірки стійкості та визначення оптимальної конфігурації.

Для врахування нерівномірності навантаження між ключами вводиться ефективна кількість обробників  $k_{eff} = k/s$ , де  $s$  – коефіцієнт нерівномірності. Фактична пропускна здатність системи визначається через відношення ефективної кількості обробників до ефективного часу обробки:

$$X_g(k) = \frac{k_{eff}}{T_{eff}(k)} = \frac{k/s}{T_p + T_i + \frac{\beta p_r}{1-p_r} + \alpha k \cdot CV + p_h T_h + \gamma k} \quad (3.30)$$

Отриманий вираз є основним результатом аналітичної моделі і встановлює функціональну залежність між рівнем паралелізму  $k$  та фактичною пропускною здатністю системи з урахуванням усіх характерних факторів. Коефіцієнт ефективності паралелізму, що показує, яка частина теоретичної пропускної здатності реалізується на практиці:

$$E_p(k) = \frac{X_g(k)}{X_{ideal}(k)} = \frac{T_p}{s \cdot T_{eff}(k)} \quad (3.31)$$

Оцінювання стійкості системи здійснюється через коефіцієнт завантаження:

$$\rho = \frac{\lambda_{eff}}{X_g(k)} \quad (3.32)$$

При  $\rho < 1$  система функціонує у стійкому режимі. В іншому випадку система є перевантаженою, черга подій необмежено зростає, і необхідно змінити параметри конфігурації - збільшити кількість обробників або знизити інтенсивність вхідного потоку. Оптимальний рівень паралелізму визначається як значення  $k$ , що максимізує фактичну пропускну здатність:

$$k^* = \arg \max k, X_g(k) \quad (3.33)$$

Отриманий вираз для фактичної пропускної здатності системи є центральним аналітичним результатом розділу. Для глибшого розуміння його

структури доцільно виділити у знаменнику константну та залежну від  $k$  компоненти. Позначимо константну частину:

$$A = T_p + T_i + \beta * p_r / (1 - p_r) + p_h * T_h \quad (3.34)$$

а коефіцієнт при  $k$  у лінійно зростаючій частині:

$$B = \alpha * CV + \gamma. \quad (3.35)$$

Тоді вираз для фактичної пропускної здатності набуває компактного вигляду:

$$X_g(k) = k / (s * (A + B * k)) \quad (3.36)$$

Ця форма запису безпосередньо розкриває характер поведінки системи. При малих  $k$ , коли виконується умова  $B * k \ll A$ , знаменник фактично дорівнює  $s * A$  і пропускна здатність зростає майже лінійно з  $k$ :  $Xg(k) \approx k / (s * A)$ . Саме в цьому діапазоні додавання кожного нового обробника дає відчутний приріст продуктивності, порівнянний з приростом в ідеальній системі. При великих  $k$ , коли виконується умова  $B * k \gg A$ , знаменник зростає пропорційно  $k$  і пропускна здатність виходить на горизонтальну асимптоту:

$$Xg_{max} = \lim Xg(k), \quad (3.37)$$

при

$$k \rightarrow \infty = 1 / (s * B) = 1 / (s * (\alpha * CV + \gamma)). \quad (3.38)$$

Цей граничний вираз є одним з ключових результатів моделі. Він показує, що максимальна досяжна пропускна здатність системи при необмеженому збільшенні числа обробників визначається виключно коефіцієнтом нерівномірності навантаження  $s$  та параметрами буферизації  $\alpha$  і конкуренції  $\gamma$ , зваженими на коефіцієнт варіації часу обробки  $CV$ . Базовий час обробки  $T_p$ , витрати на ідемпотентність  $T_i$ , внески від повторних доставок і резервного каналу входять лише до константної частини  $A$  і впливають лише на швидкість наближення до асимптоти, але не на саму асимптоту. Це означає, що оптимізація бізнес-логіки обробника або зниження частоти повторних доставок, хоча й корисні самі по собі, не здатні підвищити теоретичну стелю масштабованості



системи. Для підвищення  $Xg_{max}$  необхідно цілеспрямовано знижувати  $\alpha, \gamma$  або  $s$  через відповідні архітектурні рішення.

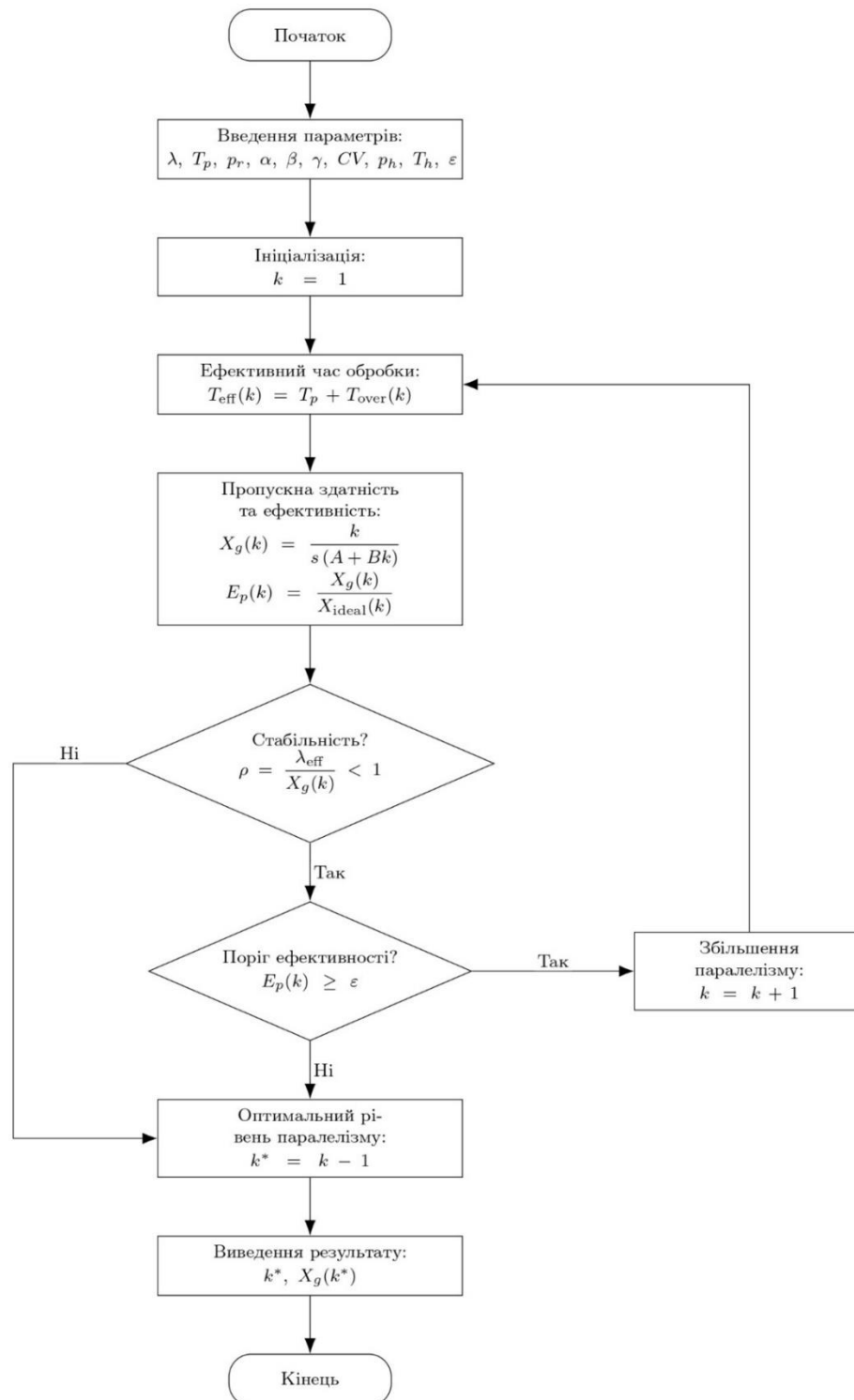


Рис. 3.1. Блок-схема алгоритму оцінювання продуктивності

Алгоритм оцінювання продуктивності, зображений на рис. 3.1, реалізує послідовність обчислень, описану в даному підрозділі.

На першому кроці задаються всі вхідні параметри системи відповідно до таблиці 3.3.

На другому кроці обчислюється фактична інтенсивність потоку

$$\lambda_{eff} = \lambda / 1 - pr. \quad (3.39)$$

На третьому кроці для заданого значення  $k$  обчислюються всі складові накладних витрат:  $T_i, T_r, T_b, Th_{eff}, T_c$ .

На четвертому кроці визначається ефективний час обробки

$$T_{eff}(k) = A + B * k \quad (3.40)$$

На п'ятому кроці обчислюється фактична пропускна здатність

$$Xg(k) = k / (s * T_{eff}(k)). \quad (3.41)$$

На шостому кроці перевіряється умова стійкості

$$\rho = \lambda_{eff} / Xg(k) < 1. \quad (3.42)$$

Якщо умова не виконується, система є нестійкою при даному  $k$  і необхідно збільшити  $k$  або переглянути архітектурні параметри. Якщо умова виконується, на сьомому кроці обчислюється коефіцієнт ефективності паралелізму  $Ep(k)$  і порівнюється із заданим порогом ефективності  $\epsilon$ . Якщо  $Ep(k) \geq \epsilon$ , поточне значення  $k$  є прийнятним. Якщо  $Ep(k) < \epsilon$ , значення  $k$  є надмірним з точки зору ефективності використання паралельних ресурсів і може бути знижено без суттєвої втрати пропускної здатності. Цикл повторюється для різних значень  $k$ , дозволяючи побудувати залежність  $Xg(k)$  та ідентифікувати оптимальну конфігурацію.

Практичний зміст знайденого  $k^*$  полягає в тому, що він визначає таку кількість паралельних обробників, при якій система одночасно забезпечує задовільну пропускну здатність і залишається стійкою з прийнятним запасом, при цьому не витрачаючи обчислювальні ресурси на обробники, внесок яких у загальну продуктивність є незначним через домінування накладних витрат. У наведеному числовому прикладі при порозі ефективності  $\epsilon = 0,5$  оптимальне  $k^*$  знаходиться в діапазоні 4-8 обробників, тоді як збільшення до 16 і вище дає відчутне зростання абсолютної пропускної здатності, але ціною різкого падіння

ефективності використання кожного окремого обробника. Вибір між цими конфігураціями є інженерним рішенням, що залежить від співвідношення вартості обчислювальних ресурсів та вимог до пропускної здатності конкретної системи, і запропонована модель дає аналітичний інструмент для обґрунтування цього вибору без необхідності проведення повного циклу навантажувального тестування для кожної розглянутої конфігурації.

### 3.5. Урахування повторної доставки, ідемпотентності та буферизації

Механізми повторної доставки, ідемпотентної обробки та буферизації є специфічними для систем з семантикою *at-least-once* і суттєво відрізняють їх поведінку від класичних моделей паралельних обчислень. Розуміння природи кожного з цих механізмів, їх взаємодії та сукупного впливу на продуктивність є необхідною передумовою для коректного застосування побудованої аналітичної моделі.

Семантика доставки *at-least-once* гарантує, що кожна подія буде доставлена споживачеві щонайменше один раз, але не виключає її повторного надходження. Причинами повторної доставки можуть бути мережеві збої, що перериваються до отримання підтвердження, перезапуски компонентів системи внаслідок збоїв або планового обслуговування, тайм-аути очікування підтвердження на стороні брокера повідомлень, а також явні механізми повторних спроб при виявленні помилок обробки. Імовірність повторної доставки  $p_r$  є агрегованим показником, що відображає сукупний вплив усіх цих причин і може суттєво варіюватися залежно від надійності мережевого середовища та конфігурації брокера.

Вплив повторних доставок на продуктивність системи є двостороннім. З одного боку, повторні доставки збільшують фактичну інтенсивність потоку подій, що надходять до обробників, відповідно до виразу  $\lambda_{eff} = \lambda / (1 - p_r)$ , – тим самим збільшуючи навантаження на обробники та зменшуючи ефективний запас пропускної здатності системи. З іншого боку, кожна повторно доставлена подія потребує часу на її ідентифікацію та відхилення, що збільшує середній час обробки відповідно до виразу  $T_r = \beta p_r / (1 - p_r)$ . Нелінійний характер обох

залежностей від  $p_r$  означає, що при значеннях  $p_r$ , що наближаються до одиниці, обидва ефекти посилюються катастрофічно швидко, тому контроль рівня повторних доставок є критично важливим для стабільності системи.

Для кількісного розуміння масштабу ефекту розглянемо декілька характерних значень параметра  $p_r$ . При  $p_r = 0,05$  фактична інтенсивність потоку перевищує номінальну лише на 5,3%, а внесок повторних доставок до часу обробки є незначним. При  $p_r = 0,20$  фактична інтенсивність вже на 25% вища за номінальну, що є суттєвим навантаженням для системи з невеликим запасом пропускної здатності. При  $p_r = 0,50$  фактична інтенсивність вдвічі перевищує номінальну – такий рівень повторних доставок практично унеможлиблює стійку роботу системи без значного збільшення кількості обробників.

Ідемпотентна обробка є механізмом, що дозволяє безпечно використовувати семантику *at-least-once* без ризику некоректного дублювання бізнес-операцій. У запропонованому методі ідемпотентність реалізується через підтримку змінної стану  $last_{seq}[key]$  для кожного ключа впорядкування, що фіксує порядковий номер останньої успішно обробленої події. Подія з порядковим номером  $E.seq \leq last_{seq}[key]$ , ідентифікується як дублікат і відхиляється без передачі до бізнес-логіки. Середній час перевірки ідемпотентності  $T_i$  є відносно невеликим для окремої події, однак він виконується для абсолютно кожної отриманої події – як унікальної, так і повторної – тому при великій інтенсивності потоку сукупні витрати на перевірку стають суттєвим внеском до загальної латентності.

Слід підкреслити, що механізм ідемпотентності також забезпечує захист від некоректного виконання при відновленні після збоїв: якщо обробник перезапускається у процесі виконання деякого кроку, повторне надходження тієї самої події не призведе до дублювання операції, оскільки її порядковий номер вже був зафіксований. Таким чином, витрати на підтримку ідемпотентності є обґрунтованою платою за коректність виконання у розподіленому середовищі.

Буферизація для відновлення порядку є третім ключовим механізмом, що безпосередньо пов'язаний із паралельною обробкою. При наявності  $k$  паралельних обробників події, що належать до одного ключа, можуть

оброблятися різними обробниками у різний час, тому їх порядок надходження до виконання може відрізнятися від порядку їх генерації. Механізм буферизації затримує поточну подію у черзі очікування доти, доки не будуть отримані та оброблені всі попередні події з тим самим ключем. Подія допускається до виконання лише тоді, коли виконується умова  $E.seq = last_{seq}[key] + 1$ .

Середня затримка буферизації  $T_b = \alpha k \cdot CV$  визначається двома факторами.

Перший – кількість паралельних обробників  $k$  – чим більше обробників, тим більша ймовірність того, що порядок виконання буде порушений, оскільки зростає різноманітність шляхів доставки та варіативність часу обробки.

Другий – коефіцієнт варіації часу обробки  $CV = \sigma/T_p$  – при більшій мінливості часу обробки ймовірність того, що пізніша за порядком подія виявиться обробленою раніше попередньої, є вищою.

При  $CV = 0$  – тобто коли час обробки є строго детермінованим – буферизація взагалі не потрібна, оскільки порядок виконання завжди збігається з порядком надходження. При  $CV > 0$  затримка буферизації є невід'ємною та зростає із  $k$ .

Важливою властивістю механізму буферизації є те, що він одночасно виконує дві функції: відновлює коректний порядок виконання при паралельній обробці та забезпечує стійкість до коротких збоїв доставки, затримуючи виконання поточних подій до надходження пропущених. Ця подвійна роль робить буферизацію незамінним компонентом архітектури, навіть незважаючи на пов'язані з нею витрати часу.

Сукупний вплив трьох розглянутих механізмів на продуктивність системи є адитивним у представленій моделі: кожен механізм додає свій внесок до ефективного часу обробки  $T_{eff}(k)$ . При цьому важливо розуміти, що ці механізми не є незалежними: повторні доставки збільшують навантаження на механізм ідемпотентності, а порушення порядку доставки, спричинене повторними доставками, збільшує потребу у буферизації. Таким чином, у системах з високим рівнем повторних доставок та значним паралелізмом усі три механізми підсилюють один одного, що може призводити до суттєвого зниження

ефективної пропускну здатності відносно теоретичної. Врахування цих взаємозв'язків є перевагою запропонованої моделі порівняно з класичними підходами, що розглядають кожен із зазначених факторів ізольовано.

Таблиця 3.4

Вплив окремих механізмів на ефективний час обробки при різних значеннях параметрів

Параметр	Низьке значення	Середнє значення	Високе значення	Залежність від $k$
$p_r$ (повторна доставка)	0,05 – незначний вплив	0,20 – помірний вплив	0,50 – суттєвий вплив	Відсутня
$CV$ (мінливість обробки)	0,1 – мала буферизація	0{,}5 – помірна буферизація	1,0 – значна буферизація	Посилює $T_b$
$k$ (обробники)	1 – без паралелізму	4 – помірний паралелізм	16 – високий паралелізм	Є аргументом
$s$ (нерівномірність)	1,0 – рівномірний розподіл	2,0 – помірна нерівномірність	5,0 – значна нерівномірність	Знижує $k_{eff}$

Для ілюстрації характеру взаємодії параметрів у таблиці 3.4 наведено порівняльний аналіз впливу окремих механізмів на загальний ефективний час обробки при різних значеннях ключових параметрів.

Таблиця 3.4 фіксує поведінку кожного параметра ізольовано, однак реальна система функціонує при одночасній дії всіх зазначених факторів. Для розуміння того, як їх спільна дія формує загальний ефективний час обробки, розглянемо конкретний числовий приклад з трьома сценаріями навантаження: сприятливим, типовим та несприятливим. У всіх трьох сценаріях базові параметри обробки залишаються незмінними:

$$T_p = 10\text{мс}, T_i = 0,5\text{мс}, \alpha = 0,3\text{мс}, \gamma = 0,1\text{мс}, \beta = 0,8, p_h = 0,02, T_h = 50\text{мс}.$$

Сприятливий сценарій відповідає системі з низьким рівнем повторних доставок, передбачуваним часом обробки та рівномірним розподілом навантаження:

$$p_r = 0,05, CV = 0,1, k = 4, s = 1,0.$$

Внески кожного механізму:

$$T_r = 0,8 * 0,05 / 0,95 = 0,042\text{мс},$$

$$T_b = 0,3 * 4 * 0,1 = 0,12\text{мс},$$

$$Th_{eff} = 0,02 * 50 = 1,0\text{мс},$$

$$T_c = 0,1 * 4 = 0,4\text{мс}.$$

$$T_{eff} = 10 + 0,5 + 0,042 + 0,12 + 1,0 + 0,4 = 12,062\text{мс}.$$

Фактична пропускна здатність

$$Xg(4) = 4 / (1,0 * 12,062 * 10^{(-3)}) \approx 332\text{подій/с}$$

порівняно з ідеальною

$$X_{ideal}(4) = 400\text{подій/с}.$$

Коефіцієнт ефективності  $E_p = 332/400 = 0,83$ , тобто система реалізує 83% теоретичної пропускної здатності.

Типовий сценарій відповідає виробничій системі з помірними значеннями всіх параметрів:

$$p_r = 0,15, CV = 0,5, k = 8, s = 1,5.$$

Внески:

$$T_r = 0,8 * 0,15 / 0,85 = 0,141\text{мс},$$

$$T_b = 0,3 * 8 * 0,5 = 1,2\text{мс},$$

$$Th_{eff} = 1,0\text{мс},$$

$$T_c = 0,1 * 8 = 0,8\text{мс}.$$

$$T_{eff} = 10 + 0,5 + 0,141 + 1,2 + 1,0 + 0,8 = 13,641\text{мс}.$$

$$Xg(8) = 8 / (1,5 * 13,641 * 10^{(-3)}) \approx 390\text{подій/с}.$$

$$X_{ideal}(8) = 800\text{подій/с}.$$

$$E_p = 390/800 = 0,49,$$

тобто система реалізує лише 49% теоретичної пропускної здатності, причому порівняно зі сприятливим сценарієм кількість обробників зросла вдвічі, а абсолютна пропускна здатність зросла лише на 17%.

Несприятливий сценарій відповідає системі під пікових навантажень з нестабільним мережевим середовищем та нерівномірним розподілом ключів:

$$p_r = 0,30, CV = 1,0, k = 16, s = 3,0.$$

Внески:

$$T_r = 0,8 * 0,30 / 0,70 = 0,343 \text{мс},$$

$$T_b = 0,3 * 16 * 1,0 = 4,8 \text{мс},$$

$$Th_{eff} = 1,0 \text{мс}, T_c = 0,1 * 16 = 1,6 \text{мс}.$$

$$T_{eff} = 10 + 0,5 + 0,343 + 4,8 + 1,0 + 1,6 = 18,243 \text{мс}.$$

$$Xg(16) = 16 / (3,0 * 18,243 * 10^{(-3)}) \approx 292 \text{подій/с}.$$

$$X_{ideal}(16) = 1600 \text{подій/с}.$$

$$E_p = 292 / 1600 = 0,18,$$

тобто система реалізує лише 18% теоретичної пропускної здатності.

Зіставлення трьох сценаріїв розкриває кілька практично важливих закономірностей. По-перше, абсолютна пропускна здатність у несприятливому сценарії з 16 обробниками виявляється нижчою, ніж у типовому з 8 обробниками: 292 проти 390 подій/с. При погіршенні умов навантаження збільшення кількості обробників не лише не допомагає, а й погіршує ситуацію, оскільки накладні витрати на буферизацію та конкуренцію зростають швидше, ніж теоретична продуктивність.

По-друге, найбільший одиничний внесок до деградації у несприятливому сценарії вносить буферизація: 4,8 мс з загального overhead у 8,243 мс, тобто 58% усіх накладних витрат. CV і k є ключовими параметрами з точки зору керованості продуктивністю: зниження мінливості часу обробки через стабілізацію бізнес-логіки або обмеження k до розумних меж є значно ефективнішим інструментом, ніж спроба компенсувати погані умови збільшенням числа обробників.

По-третє, внесок резервного каналу  $Th_{eff} = 1,0$  мс є однаковим у всіх трьох сценаріях і при малому  $p_h = 0,02$  не є визначальним. Проте якщо  $p_h$



зростає до 0,1 або вище, що може відбуватись при нестабільному брокері, внесок резервного каналу стає 5,0 мс і починає суттєво впливати на загальну латентність.

Ці спостереження підводять до важливого висновку щодо пріоритетності заходів з оптимізації продуктивності. Оскільки механізм буферизації є єдиним з трьох розглянутих, що залежить від  $k$ , і саме він є домінуючим джерелом накладних витрат при великому  $k$ , першим кроком при проектуванні або налаштуванні системи має бути мінімізація CV: стабілізація часу обробки через кешування залежностей, попереднє завантаження конфігурацій або виокремлення повільних операцій в асинхронні задачі. Другим кроком є вибір  $k$  у межах, де  $T_b$  ще не домінує у знаменнику виразу для  $Xg(k)$ , тобто де виконується умова  $B * k < A$ , що гарантує роботу системи в режимі ефективного масштабування, а не в режимі насичення. Третім кроком є контроль  $s$  через рівномірну схему хешування ключів: навіть при ідеальних значеннях усіх інших параметрів значне  $s$  знецінює горизонтальне масштабування пропорційно своєму значенню, і ніяка кількість додаткових обробників не здатна компенсувати цей ефект, якщо "гарячий" ключ серіалізує виконання.

Таким чином, три механізми, розглянуті в даному підрозділі, утворюють взаємопов'язану систему, в якій погіршення одного параметра посилює негативний вплив інших. Запропонована аналітична модель дозволяє кількісно оцінити цей сукупний вплив для конкретної конфігурації системи та визначити, який саме параметр є вузьким місцем у кожному конкретному випадку, що є необхідною умовою для прийняття обґрунтованих рішень при проектуванні та оптимізації мікросервісних систем упорядкованої доставки подій.

### **3.6. Оцінка накладних витрат та визначення оптимального рівня паралелізму**

Визначення оптимального рівня паралелізму є центральною практичною задачею аналітичного моделювання систем впорядкованої доставки подій. Як показано у попередніх підрозділах, збільшення кількості обробників  $k$  одночасно підвищує теоретичну пропускну здатність і збільшує накладні витрати, пов'язані

з буферизацією та конкуренцією за ресурси. Внаслідок цього залежність фактичної пропускної здатності  $X_g(k)$  від  $k$  є нелінійною і має характерну форму, що вимагає окремого аналізу.

Для розуміння структури накладних витрат доцільно розкласти сумарний ефективний час обробки  $T_{eff}(k)$  на дві компоненти - константну та залежну від  $k$ . Константна компонента об'єднує всі витрати, що не залежать від кількості обробників:

$$A = T_p + T_i + \frac{\beta, p_r}{1 - p_r} + p_h T_h \quad (3.43)$$

Залежна від  $k$  компонента включає витрати на буферизацію та конкуренцію за ресурси:

$$B = \alpha \cdot CV + \gamma \quad (3.44)$$

Тоді ефективний час обробки набуває компактного вигляду  $T_{eff}(k) = A + B \cdot k$ , а вираз для фактичної пропускної здатності спрощується до:

$$X_g(k) = \frac{k}{s(A + B \cdot k)} \quad (3.45)$$

Ця форма запису наочно демонструє характер залежності: при малих  $k$ , коли  $A \gg Bk$ , пропускна здатність зростає майже лінійно з  $k$ , оскільки константна частина знаменника домінує. При великих  $k$ , коли  $Bk \gg A$ , знаменник зростає пропорційно  $k$ , і пропускна здатність прагне до горизонтальної асимптоти:

$$\lim_{k \rightarrow \infty} X_g(k) = \frac{1}{s \cdot B} = \frac{1}{s(\alpha \cdot CV + \gamma)} \quad (3.46)$$

Цей граничний вираз є принципово важливим результатом: він показує, що максимальна досяжна пропускна здатність системи при необмеженому збільшенні числа обробників визначається виключно коефіцієнтом нерівномірності навантаження  $s$  та параметрами  $\alpha$  і  $\gamma$ , що характеризують механізми буферизації і синхронізації. Базовий час обробки  $T_p$ , витрати на ідемпотентність  $T_i$  та внески від повторних доставок і резервного каналу входять

лише до константної частини  $A$  і не впливають на асимптоту, – але вони визначають, наскільки швидко система наближається до цієї межі.

Для аналізу характеру зростання  $X_g(k)$  обчислимо першу похідну за  $k$ :

$$\frac{\partial X_g}{\partial k} = \frac{A}{s(A + B \cdot k)^2} \quad (3.47)$$

Оскільки  $A > 0$  і  $s > 0$  за визначенням, ця похідна є строго позитивною для будь-якого  $k > 0$ . Це означає, що функція  $X_g(k)$  є монотонно зростаючою і не має локального максимуму у звичайному математичному сенсі. Таким чином, теоретично оптимальним є максимально можливе  $k$ . Проте на практиці збільшення  $k$  обмежене кількома факторами: доступними обчислювальними ресурсами, вартістю утримання обробників, зниженням коефіцієнта ефективності паралелізму  $E_p(k)$  та необхідністю забезпечення стійкості системи.

Коефіцієнт ефективності паралелізму  $E_p(k) = T_p / (s \cdot T_{eff}(k))$  є монотонно спадаючою функцією від  $k$ , оскільки  $T_{eff}(k)$  зростає з  $k$ . Це означає, що кожен наступний доданий обробник дає менший приріст пропускної здатності, ніж попередній – класичний ефект спадної граничної продуктивності. Для практичного визначення оптимального  $k^*$  доцільно ввести поріг ефективності  $\varepsilon \in (0,1]$  і знаходити максимальне  $k$ , при якому  $E_p(k) \geq \varepsilon$ . З умови  $E_p(k) \geq \varepsilon$  отримуємо:

$$k^* \leq \frac{T_p / (s\varepsilon) - A}{B} \quad (3.48)$$

Цей вираз дозволяє аналітично оцінити граничну кількість обробників, при якій ефективність використання паралельних ресурсів залишається не нижчою за заданий поріг. Наприклад, при  $\varepsilon = 0,8$  (тобто ефективність не нижча за 80% від ідеальної) оптимальне число обробників визначається підстановкою відповідних параметрів системи.

Важливою складовою оцінки є також аналіз впливу нерівномірності навантаження на оптимальну конфігурацію. Коефіцієнт нерівномірності  $s$  входить до виразу для  $X_g(k)$  як мультиплікативний дільник, що означає

пропорційне зниження як поточної пропускної здатності, так і граничної асимптоти. Ефективна кількість обробників при нерівномірному навантаженні становить  $k_{eff} = k/s$ , а отже для досягнення тієї самої фактичної пропускної здатності, що й при рівномірному навантаженні, необхідно у  $s$  разів більше обробників. З практичної точки зору це підкреслює важливість рівномірного розподілу подій за ключами через відповідну схему хешування або маршрутизації.

Для наочного порівняння поведінки запропонованої моделі з класичними підходами побудуємо порівняльну таблицю, що відображає ключові характеристики трьох моделей масштабування при різних рівнях паралелізму.

Таблиця 3.5

Порівняльний аналіз моделей масштабування за ключовими характеристиками

Характеристика	Ідеальна модель	Модель Амдала	Запропонована модель	Характеристика
Залежність $X(k)$ від $k$	Лінійна	Сублінійна	Сублінійна з асимптотою	Залежність $X(k)$ від $k$
Наявність асимптоти	Відсутня	При $k \rightarrow \infty$	При $k \rightarrow \infty$	Наявність асимптоти
Врахування $p_r$	Ні	Ні	Так	Врахування $p_r$

Порівняльний аналіз підтверджує, що запропонована модель забезпечує більш реалістичний опис поведінки системи за рахунок урахування специфічних факторів подієво-орієнтованих архітектур. Відхилення від ідеальної моделі - так звана "втрата масштабованості" - зростає зі збільшенням  $k$  значно швидше, ніж передбачає модель Амдала, що пояснюється лінійним зростанням витрат на буферизацію та синхронізацію.

Для ілюстрації методики визначення оптимального рівня паралелізму наведемо блок-схему процесу прийняття рішення щодо конфігурації системи з урахуванням заданого порогу ефективності на рисунку 3.2.

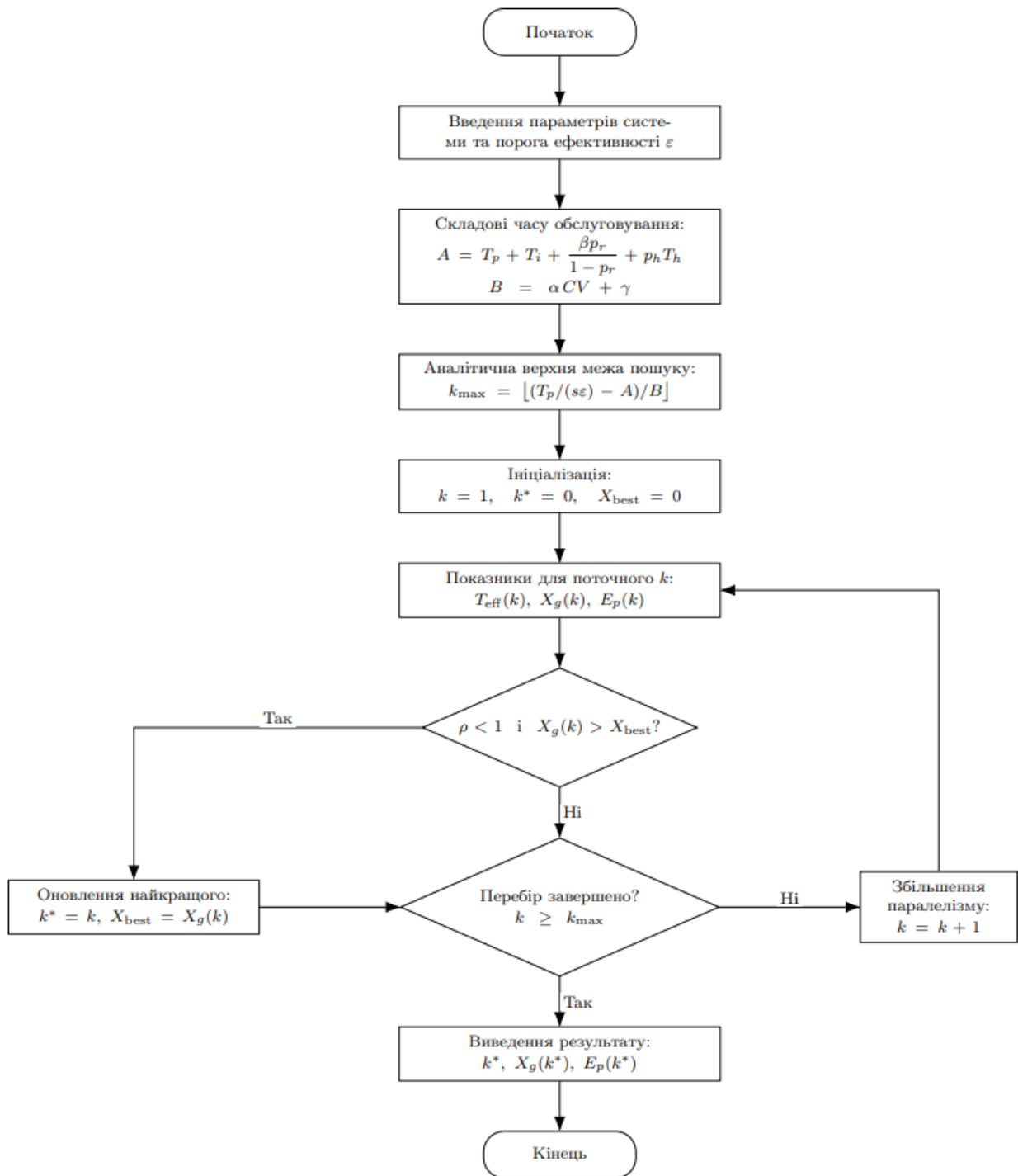


Рис. 3.2. Блок-схема визначення оптимального рівня паралелізму за критерієм ефективності

Алгоритм, зображений на рис. 3.2, реалізує таку послідовність кроків. На першому кроці задаються вхідні параметри системи та бажаний поріг ефективності  $\epsilon$ . На другому кроці обчислюються константна частина  $A$  і коефіцієнт  $B$  відповідно до виразів, наведених у даному підрозділі. На третьому кроці аналітично оцінюється верхня межа  $k^*$  за формулою  $k^* \leq (T_p / (s * \epsilon) - A) / B$ , що дає початкову точку пошуку. На четвертому кроці для кожного

цілочисельного значення  $k$  від 1 до  $k^*$  обчислюються  $T_{eff}(k)$ ,  $Xg(k)$  та  $Ep(k)$ . На п'ятому кроці перевіряється умова стійкості  $\rho < 1$ : якщо при деякому  $k$  умова не виконується, це  $k$  і всі більші значення є неприйнятними незалежно від ефективності. На шостому кроці серед усіх  $k$ , що задовольняють одночасно умову стійкості вибирається те, що забезпечує максимальну абсолютну пропускну здатність  $Xg(k)$ . Результатом алгоритму є оптимальна кількість обробників  $k^*$  та відповідні значення  $Xg(k^*)$  і  $Ep(k^*)$ , що однозначно характеризують очікувану продуктивність системи при обраній конфігурації.

Доповнимо порівняння Таблиці 3.5 кількісним прикладом, що демонструє різницю між передбаченнями трьох моделей для однакового набору параметрів. Використаємо параметри з числового прикладу підрозділу 3.4:  $T_p = 10$  мс, частка послідовних обчислень для моделі Амдала  $P = 0,9$  що відповідає 90% паралелізованих операцій,  $A = 11,589$  мс,  $B = 0,25$  мс,  $s = 1,5$ .

При  $k = 8$ : ідеальна модель дає

$$X_{ideal}(8) = 800 \text{ подій/с,}$$

модель Амдала дає

$$\begin{aligned} X_{amdahl}(8) &= 1 / (T_p * (1 - P + P/k)) = 1 / (10 * (0,1 + 0,9/8) * 10^{(-3)}) \\ &= 1 / (2,125 * 10^{(-3)}) \approx 471 \text{ подій/с,} \end{aligned}$$

запропонована модель дає

$$\begin{aligned} Xg(8) &= 8 / (1,5 * (11,589 + 0,25 * 8) * 10^{(-3)}) \\ &= 8 / (1,5 * 13,589 * 10^{(-3)}) \approx 392 \text{ подій/с.} \end{aligned}$$

Втрата масштабованості відносно ідеальної: ідеальна 0%, Амдала 41%, запропонована 51%.

При  $k = 32$ : ідеальна модель дає 3200 подій/с, модель Амдала дає

$$1 / (10 * (0,1 + 0,9/32) * 10^{(-3)}) = 1 / (1,281 * 10^{(-3)}) \approx 781 \text{ подій/с,}$$

запропонована модель дає

$$\begin{aligned} 32 / (1,5 * (11,589 + 0,25 * 32) * 10^{(-3)}) &= 32 / (1,5 * 19,589 * 10^{(-3)}) \\ &\approx 1089 \text{ подій/с.} \end{aligned}$$

Тут виявляється принципова відмінність: при  $k = 32$  запропонована модель дає вищу абсолютну пропускну здатність ніж модель Амдала, 1089 проти 781 подій/с, хоча її відносна втрата масштабованості є більшою: 66% проти 76% відносно ідеальної. Це пояснюється тим, що модель Амдала обмежена послідовною компонентою, яка при  $k = 32$  стає домінуючим обмеженням, тоді як запропонована модель не має фіксованої послідовної частини, а лише лінійно зростаючі накладні витрати, що при помірних значеннях  $\alpha$  і  $\gamma$  все ще дозволяють отримувати приріст від збільшення  $k$ .

При  $k = 128$ : ідеальна модель дає 12800 подій/с, модель Амдала дає  $\approx 909$  подій/с і наближається до своєї асимптоти  $1/(T_p * (1 - P)) = 1000$  подій/с, запропонована модель дає

$$128/(1,5 * (11,589 + 0,25 * 128) * 10^{(-3)}) = 128/(1,5 * 43,589 * 10^{(-3)}) \\ \approx 1956 \text{ подій/с}$$

і наближається до своєї асимптоти  $X_{g\_max} \approx 2667$  подій/с. Таким чином, асимптота запропонованої моделі у даному прикладі приблизно втричі вища за асимптоту моделі Амдала, що пояснюється різною природою обмежуючого фактора: у моделі Амдала це фіксована послідовна компонента алгоритму, у запропонованій моделі це параметри архітектурних механізмів  $\alpha$ ,  $\gamma$  та  $s$ , які можуть бути цілеспрямовано знижені.

Ця відмінність є принциповою з практичної точки зору. Якщо архітектор системи орієнтується на модель Амдала при плануванні масштабування, він може недооцінити реальну деградацію продуктивності при малих і середніх  $k$ , де запропонована модель дає нижчу пропускну здатність через витрати на буферизацію та конкуренцію, яких модель Амдала не враховує. Водночас при великих  $k$  він може переоцінити ступінь обмеженості системи, оскільки асимптота моделі Амдала є нижчою, ніж реальна межа масштабованості систем впорядкованої доставки з помірними значеннями  $\alpha$  і  $\gamma$ . Обидві помилки мають практичні наслідки: перша призводить до недостатнього резервування обробників у діапазоні  $k$ , де система ще масштабується ефективно, друга до

передчасної відмови від горизонтального масштабування на користь вертикального або архітектурних змін, які насправді ще не є необхідними.

Підсумовуючи результати підрозділу, зазначимо, що розроблена аналітична модель вперше дозволяє отримати кількісну оцінку оптимального рівня паралелізму  $k^*$  для систем упорядкованої доставки подій з at-least-once семантикою на основі параметрів конкретної реалізації без проведення повного циклу навантажувального тестування. Отримана аналітична формула  $k^* \leq (T_p / (s * \epsilon) - A) / B$  є прямим практичним результатом моделі: вона дозволяє на етапі проектування визначити розумну верхню межу кількості обробників для заданого порогу ефективності, а також виявити, які саме параметри системи є вузьким місцем з точки зору масштабованості, зниження яких дасть найбільший приріст  $Xg_{max}$ . Саме ця можливість аналітичного обґрунтування конфігурації, на відміну від виключно емпіричного підбору, є тією практичною цінністю, яку запропонована модель додає до існуючого арсеналу інструментів проектування мікросервісних систем.

### 3.7. Дослідження умов стабільності та масштабованості системи

Умови стабільності та масштабованості є фундаментальними характеристиками будь-якої системи обробки повідомлень і визначають практичні межі її застосовності при зміні навантаження та конфігурації. У контексті систем впорядкованої доставки подій ці характеристики набувають додаткової складності через наявність механізмів буферизації, повторних доставок та ідемпотентної обробки, що взаємодіють між собою нелінійним чином.

Умова стабільності системи у теорії масового обслуговування формулюється як вимога того, щоб інтенсивність надходження запитів не перевищувала інтенсивності їх обробки. У термінах запропонованої моделі ця умова записується через коефіцієнт завантаження:



$$\rho = \frac{\lambda_{eff}}{X_g(k)} < 1 \quad (3.49)$$

При виконанні цієї умови черга подій, що очікують обробки, залишається обмеженою, а система функціонує у стаціонарному режимі з кінцевими значеннями латентності та довжини черги. При  $\rho \geq 1$  черга необмежено зростає, латентність прямує до нескінченності і система фактично перестає виконувати свої функції у прийнятний час.

Підставляючи вирази для  $\lambda_{eff}$  та  $X_g(k)$  у умову стабільності, отримуємо розгорнутий вираз:

$$\rho = \frac{\lambda \cdot s \cdot T_{eff}(k)}{k(1 - p_r)} < 1 \quad (3.50)$$

Оскільки  $T_{eff}(k)$  сам залежить від  $k$  через лінійно зростаючі члени  $\alpha k \cdot CV$  та  $\gamma k$ , умова стабільності є нелінійною відносно  $k$  і не допускає простого аналітичного розв'язання у загальному випадку. Підставляючи  $T_{eff}(k) = A + Bk$ , де  $A$  та  $B$  визначені у підрозділі 3.6, умова стабільності набуває вигляду:

$$\frac{\lambda \cdot s \cdot (A + Bk)}{k(1 - p_r)} < 1 \quad (3.51)$$

Перетворення цього нерівняння дозволяє отримати умову відносно  $k$ :

$$k > \frac{\lambda \cdot s \cdot A}{(1 - p_r) - \lambda \cdot s \cdot B} \quad (3.52)$$

Ця умова має сенс лише тоді, коли знаменник правої частини є позитивним, тобто коли  $(1 - p_r) > \lambda \cdot s \cdot B$ . Якщо ця нерівність не виконується, система є нестабільною при будь-якій кількості обробників - ситуація, що виникає при надмірно великому навантаженні або надмірно великих коефіцієнтах накладних витрат  $\alpha$ ,  $\gamma$ . У такому випадку необхідна фундаментальна зміна архітектури – зменшення рівня нерівномірності навантаження  $s$ , оптимізація механізму буферизації (зменшення  $\alpha$ ) або зниження конкуренції за ресурси (зменшення  $\gamma$ ).

Аналіз меж масштабованості системи передбачає дослідження поведінки пропускної здатності  $X_g(k)$  при зростанні  $k$ . Як показано у підрозділі 3.6, ця функція є монотонно зростаючою і прагне до асимптотичного значення:

$$X_g^{max} = \lim_{k \rightarrow \infty} X_g(k) = \frac{1}{s \cdot B} = \frac{1}{s(\alpha \cdot CV + \gamma)} \quad (3.53)$$

Це значення визначає абсолютну верхню межу пропускної здатності системи, яка не може бути перевищена незалежно від кількості обробників. Для досягнення заданого рівня пропускної здатності  $X_{target}$  необхідно, щоб  $X_{target} < X_g^{max}$ , тобто:

$$X_{target} < \frac{1}{s(\alpha \cdot CV + \gamma)} \quad (3.54)$$

Якщо ця умова не виконується, бажаний рівень пропускної здатності принципово недосяжний у рамках поточної конфігурації і потребує структурних змін – зменшення  $s$ ,  $\alpha$  або  $\gamma$ .

Швидкість наближення до асимптоти характеризується відношенням поточної пропускної здатності до граничної. Із виразу для  $X_g(k)$  отримуємо:

$$\frac{X_g(k)}{X_g^{max}} = \frac{Bk}{A + Bk} = \frac{1}{1 + A/(Bk)} \quad (3.55)$$

Ця залежність показує, що при  $k = A/B$  досягається 50% від максимальної пропускної здатності, при  $k = 4A/B$  – 80%, при  $k = 9A/B$  – 90%. Таким чином, константна частина  $A$  відносно коефіцієнта  $B$  визначає "характерний масштаб" системи - кількість обробників, при якій починається режим насичення і подальше збільшення  $k$  дає все менший відносний приріст пропускної здатності.

Масштабованість системи можна кількісно охарактеризувати через показник відносної ефективності масштабування - відношення приросту пропускної здатності при збільшенні кількості обробників на одиницю до теоретичного приросту при ідеальному масштабуванні:

$$SE(k) = \frac{X_g(k+1) - X_g(k)}{X_{ideal}(k+1) - X_{ideal}(k)} = \frac{A \cdot T_p}{s(A + Bk)(A + B(k+1))} \quad (3.56)$$

Цей показник монотонно спадає з  $k$ , відображаючи поступове вичерпання ресурсу масштабування. При  $SE(k) < \varepsilon_{min}$  подальше збільшення кількості обробників є економічно недоцільним, навіть якщо технічно можливим.

Суттєвий вплив на масштабованість системи чинить коефіцієнт нерівномірності навантаження  $s$ . При рівномірному розподілі подій за ключами ( $s = 1$ ) система демонструє найкращу масштабованість і найвищу граничну пропускну здатність. При наявності "гарячих" ключів ( $s > 1$ ) масштабованість деградує пропорційно  $s$ : гранична пропускну здатність знижується у  $s$  разів, а для досягнення тієї самої пропускну здатності потрібно у  $s$  разів більше обробників. Це підкреслює критичну важливість рівномірного хешування при проектуванні систем впорядкованої доставки подій.

Вплив імовірності повторних доставок  $p_r$  на стабільність є неоднозначним. З одного боку, повторні доставки збільшують ефективну інтенсивність потоку  $\lambda_{eff}$ , підвищуючи коефіцієнт завантаження  $\rho$  і тим самим загрожуючи стабільності. З іншого боку,  $p_r$  не впливає безпосередньо на граничну пропускну здатність  $X_g^{max}$ , оскільки не входить до коефіцієнта  $B$ . Проте  $p_r$  входить до константної частини  $A$  через член  $T_r$ , збільшуючи ефективний час обробки і тим самим уповільнюючи наближення до асимптоти. Таким чином, системи з високим рівнем повторних доставок потребують більшої кількості обробників для досягнення того самого ступеня насичення пропускну здатності.

Таблиця 3.6

#### Граничні умови стабільності та масштабованості системи

Умова	Математичний вираз	Практична інтерпретація
Стійкість системи	$\rho < 1$	Черга подій обмежена, система функціонує нормально
Існування межі масштабованості	$(1 - p_r) > \lambda \cdot s \cdot B$	Система масштабується при будь-якому $k$
Досяжність цільової пропускну здатності	$X_{target} < 1/(sB)$	Цільова продуктивність принципово досяжна
Ефективне масштабування	$E_p(k) \geq$	Кожен обробник використовується ефективно
Режим насичення	$k \gg A/B$	Подальше збільшення $k$ дає малий приріст
Критична нерівномірність	$s \gg 1$	Масштабування практично неефективне

Для комплексного розуміння умов стабільності та меж масштабованості у таблиці 3.6 систематизовано ключові граничні умови та їх практичну інтерпретацію.

Умови, систематизовані в таблиці 3.6, утворюють ієрархічну послідовність перевірок, яку доцільно виконувати саме в тому порядку, в якому вони наведені.

Порушення першої умови, тобто  $\rho \geq 1$ , означає, що система вже нестабільна і жодна з наступних характеристик не має практичного сенсу, оскільки черга необмежено зростає незалежно від ефективності використання обробників.

Порушення другої умови, тобто невиконання  $1 - p_r > \lambda * s * B$ , означає, що нестабільність є принциповою і не може бути усунена збільшенням  $k$ : в цьому випадку зростання  $k$  збільшує як чисельник, так і знаменник виразу для  $Xg(k)$  з однаковою швидкістю, і система ніколи не досягне стійкого режиму незалежно від кількості обробників.

Порушення третьої умови означає, що навіть при досягненні асимптоти пропускну здатність є недостатньою для цільового навантаження і вимагає архітектурних змін. Лише після підтвердження виконання перших трьох умов має сенс переходити до четвертої і п'ятої, що характеризують якість масштабування, а не його принципову можливість.

Ця ієрархія перевірок є практичним алгоритмом діагностики системи при збільшенні навантаження або зміні конфігурації.

Якщо система демонструє деградацію продуктивності, першим кроком є перевірка коефіцієнта завантаження  $\rho$ : якщо  $\rho \geq 1$ , система перевантажена і потрібне негайне збільшення  $k$  або зниження  $\lambda$ .

Якщо  $\rho < 1$ , але близько до одиниці, система функціонує у стійкому режимі, але без достатнього запасу і є вразливою до пікових навантажень.

Якщо  $\rho$  задовільний, але ефективність  $Er(k)$  низька, це означає що система використовує обробники неефективно через надмірне  $k$  відносно оптимального  $k^*$  і частину обробників доцільно вивільнити без суттєвої втрати пропускну здатності.

Якщо ж показник відносної ефективності масштабування  $SE(k)$  опустився нижче мінімального порогу, подальше збільшення  $k$  є економічно невиправданим і ресурси краще спрямувати на зниження  $\alpha$ ,  $\gamma$  або  $s$ .

Блок-схема на рис. 3.3 реалізує описану вище ієрархічну послідовність перевірок у вигляді алгоритму, придатного для безпосереднього застосування при проектуванні та діагностиці системи.

На першому кроці алгоритму задаються всі параметри системи та обчислюються похідні величини  $A, B$  і  $\lambda_{eff}$ . На другому кроці перевіряється умова існування межі масштабованості: якщо  $1 - p_r \leq \lambda * s * B$ , алгоритм видає висновок про принципову нестабільність і вказує на необхідність зміни одного або кількох з параметрів  $s, \alpha, \gamma$  без розгляду конкретних значень  $k$ . На третьому кроці, якщо умова виконується, обчислюється мінімальне

$$k_{min} = \lceil \lambda \cdot s \cdot A / (1 - p_r) - \lambda \cdot s \cdot B \rceil \quad (3.57)$$

і перевіряється, чи є воно реалізовним з точки зору доступних ресурсів. На четвертому кроці для поточного значення  $k$ , що задовольняє  $k \geq k_{min}$ , обчислюється коефіцієнт завантаження  $\rho$  і перевіряється умова  $\rho < 1$  з урахуванням бажаного запасу надійності, наприклад  $\rho \leq 0,8$  для систем з вимогами до SLA.

На п'ятому кроці обчислюється коефіцієнт ефективності  $Ep(k)$  і порівнюється з порогом  $\epsilon$ , що визначає прийнятну ефективність використання ресурсів.

На шостому кроці перевіряється співвідношення  $Xg(k)$  з цільовою пропускною здатністю  $X_{target}$  і з асимптотою  $Xg_{max}$ : якщо  $X_{target} > Xg_{max}$ , цільова пропускна здатність принципово недосяжна і потрібні архітектурні зміни, якщо  $X_{target} \leq Xg(k)$  і виконані всі попередні умови, поточна конфігурація є прийнятною.

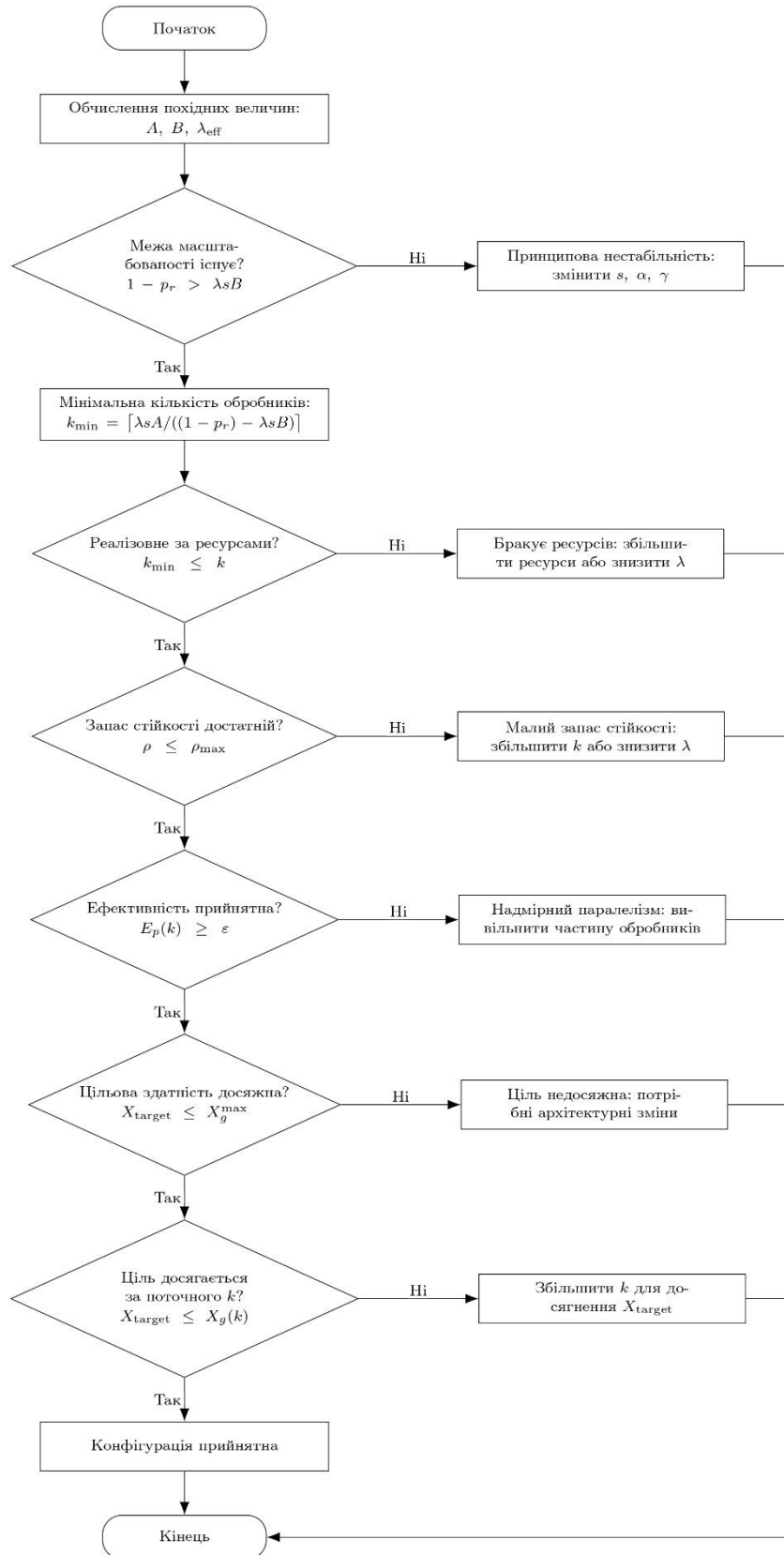


Рис. 3.3. Блок-схема аналізу стабільності та масштабованості системи

Важливо підкреслити, що алгоритм рис. 3.3 є не лише інструментом перевірки конкретної конфігурації, але й засобом чутливісного аналізу:

послідовно змінюючи один параметр при фіксованих інших і проходячи алгоритм повторно, можна кількісно оцінити вплив кожного параметра на стабільність і масштабованість системи. Зокрема, такий аналіз дозволяє відповісти на практично важливі питання: наскільки можна збільшити  $\lambda$  до втрати стабільності при поточному  $k$ , при якому значенні  $s$  система переходить до режиму принципової нестабільності, яке зниження  $\alpha$  або  $\gamma$  дозволить підвищити  $Xg_{max}$  до рівня  $X_{target}$  без збільшення  $k$ . Відповіді на ці питання, отримані аналітично за формулами моделі, дозволяють приймати обґрунтовані архітектурні рішення на основі кількісного аналізу, а не виключно на основі досвіду та інтуїції проектувальника.

Підсумовуючи результати розділу в цілому, зазначимо, що побудована аналітична модель вперше надає формальний інструмент для одночасної оцінки п'яти специфічних факторів деградації продуктивності систем упорядкованої паралельної доставки подій з at-least-once семантикою. Отриманий вираз  $Xg(k) = k / (s * (A + B * k))$ , де  $A$  і  $B$  є функціями параметрів системи, а також виведені на його основі умови стабільності, формула оптимального  $k^*$  та вираз для асимптотичної межі  $Xg_{max}$ , утворюють взаємопов'язану систему аналітичних результатів. Ця система дозволяє підвищити точність прогнозування продуктивності порівняно з класичними моделями масштабування за рахунок урахування механізмів, специфічних для даного класу архітектур, і тим самим скоротити розрив між теоретичними оцінками і реальною поведінкою системи у виробничому середовищі.

### **Висновки до розділу 3**

У розділі 3 вперше розроблено аналітичну модель продуктивності системи впорядкованої доставки подій з семантикою at-least-once, що є другим науковим результатом дисертаційної роботи. Модель враховує комплекс факторів, специфічних для мікросервісних подієво-орієнтованих систем, та забезпечує формальну основу для прогнозування продуктивності й оптимізації конфігурації таких систем.

Проведений огляд сучасних підходів до аналізу продуктивності подієво-орієнтованих систем показав, що існуючі дослідження охоплюють окремі аспекти проблематики – детерміноване впорядкування подій, алгоритмічну оптимізацію паралельного виконання, впорядковане виконання розподілених транзакцій, відмовостійкість та управління розподіленими транзакціями через патерн Saga. Разом з тим виявлено наукову прогалину: відсутність узагальненої аналітичної моделі, що одночасно враховує повторну доставку повідомлень, ідемпотентну обробку, буферизацію подій поза порядком, резервні канали передачі та нерівномірність розподілу навантаження між ключами.

Аналіз впливу паралелізму на латентність та пропускну здатність виявив, що ефекти, характерні для систем впорядкованої доставки подій, поділяються на дві групи за характером залежності від кількості обробників. Константні відносно кількості обробників ефекти – повторна доставка, ідемпотентна обробка та резервний канал – зменшують ефективну пропускну здатність незалежно від рівня паралелізму, тоді як залежні від кількості обробників ефекти – буферизація для відновлення порядку та конкуренція за спільні ресурси – лінійно зростають із кількістю обробників і визначають практичну межу ефективного масштабування.

У процесі формалізації параметрів системи введено повну систему позначень, що охоплює характеристики вхідного потоку подій, параметри обробки, коефіцієнти накладних витрат та метрики ефективності. Систематизація параметрів у єдину таблицю забезпечує однозначність аналітичних виразів та зручність їх практичного застосування.

Побудована аналітична модель встановлює функціональну залежність між рівнем паралелізму та фактичною пропускну здатністю системи. Структурний аналіз отриманого виразу показав, що ефективний час обробки розкладається на константну та лінійно зростаючу від кількості обробників компоненти, що зумовлює сублінійний характер зростання пропускну здатності та існування асимптотичної межі. Коефіцієнт ефективності паралелізму є монотонно спадаючою функцією від кількості обробників, що відображає ефект спадної граничної продуктивності паралельних ресурсів.



Детальний аналіз механізмів повторної доставки, ідемпотентності та буферизації показав, що ці механізми є взаємозалежними: повторні доставки збільшують навантаження на механізм ідемпотентності, а порушення порядку, спричинене повторними доставками, збільшує потребу у буферизації. У системах з одночасно високим рівнем повторних доставок та значним паралелізмом усі три механізми підсилюють один одного, що може призводити до суттєвого зниження ефективної пропускної здатності відносно теоретичної.

Оцінка накладних витрат та аналіз граничної пропускної здатності виявили, що максимально досяжна продуктивність системи при необмеженому збільшенні кількості обробників визначається виключно коефіцієнтом нерівномірності навантаження та параметрами механізмів буферизації і синхронізації. Для практичного визначення оптимального рівня паралелізму запропоновано використання порогу ефективності, що дозволяє аналітично оцінити граничну кількість обробників, при якій ефективність використання паралельних ресурсів залишається прийнятною.

Дослідження умов стабільності та масштабованості показало, що стійкість системи визначається умовою неперевищення коефіцієнта завантаження одиниці і залежить від усіх параметрів моделі у комплексі. Виявлено, що при надмірно великих коефіцієнтах накладних витрат або надмірно великому коефіцієнті нерівномірності система може бути нестабільною при будь-якій кількості обробників, що вимагає фундаментальних архітектурних змін, а не лише збільшення кількості обробників. Визначено аналітичні умови існування межі масштабованості та досяжності цільової пропускної здатності, що забезпечує практичний інструментарій для проектування систем.

Порівняльний аналіз із класичними моделями масштабування – ідеальною моделлю та моделлю Амдала – підтвердив, що запропонована модель забезпечує значно реалістичніший опис поведінки систем впорядкованої доставки подій. Відхилення від ідеальної моделі зростає зі збільшенням рівня паралелізму значно швидше, ніж передбачає модель Амдала, що пояснюється лінійним зростанням специфічних для даного класу систем витрат на буферизацію та синхронізацію.

Наукова новизна отриманого результату полягає в тому, що вперше для класу систем впорядкованої паралельної доставки подій з at-least-once семантикою отримано аналітичний вираз пропускну здатності у замкнутій формі  $Xg(k) = k / (s * (A + B * k))$ , де константна компонента  $A$  і лінійний коефіцієнт  $B$  є явними функціями вимірюваних параметрів системи. На відміну від моделі Амдала, де межа масштабованості визначається фіксованою послідовною часткою алгоритму, у запропонованій моделі межа масштабованості  $Xg_{max} = 1 / (s * B) = 1 / (s * (\alpha * CV + \gamma))$  визначається архітектурними параметрами конкретної реалізації, тобто коефіцієнтами буферизації та конкуренції за спільні ресурси, а також нерівномірністю розподілу навантаження між ключами. Це принципово відрізняє запропоновану модель від існуючих: вона не лише описує характер деградації продуктивності, але й вказує на конкретні параметри, зміна яких підвищує межу масштабованості, що є неможливим у рамках класичних підходів. Аналітично встановлено, що зниження  $\alpha$  на 50% або зниження  $s$  вдвічі дають однаковий ефект щодо підвищення  $Xg_{max}$ , що дозволяє кількісно порівнювати альтернативні напрями оптимізації ще на етапі проектування.

Додатковою складовою наукової новизни є аналітичне встановлення необхідної і достатньої умови принципової масштабованості системи у вигляді нерівності  $1 - p_r > \lambda * s * B$ , яка вперше формально розмежовує два якісно різні режими поведінки системи при збільшенні кількості обробників. При виконанні цієї умови система є принципово масштабованою і для будь-якого цільового рівня пропускну здатності, що не перевищує  $Xg_{max}$ , існує скінченна кількість обробників, достатня для його досягнення. При порушенні цієї умови жодна кількість обробників не забезпечує стабільної роботи системи і єдиним засобом є зміна архітектурних параметрів. Таке розмежування дозволяє на етапі проектування однозначно класифікувати конфігурацію системи як масштабовану або принципово обмежену без проведення повного циклу навантажувального тестування, що скорочує час і вартість прийняття архітектурних рішень у виробничих системах з жорсткими вимогами до продуктивності і доступності.

## **РОЗДІЛ 4. УДОСКОНАЛЕННЯ АРХІТЕКТУРИ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ ДОСТАВКИ ПОДІЙ**

### **4.1. Розробка архітектури забезпечення безперервності доставки подій**

Забезпечення безперервності доставки подій є однією з ключових вимог до мікросервісних систем, що функціонують в умовах високого навантаження та можливих часткових збоїв окремих компонентів. Під безперервністю доставки розуміється здатність системи гарантувати передачу та обробку кожної ініційованої події незалежно від стану транспортного рівня, відмов вузлів або тимчасової недоступності брокера повідомлень. Забезпечення цієї властивості потребує спеціальної архітектурної організації, що виходить за межі стандартних можливостей брокера повідомлень.

Запропонована архітектура системи доставки подій базується на інтеграції трьох взаємодоповнюючих механізмів: асинхронного обміну повідомленнями через брокер RabbitMQ як основного транспортного рівня, резервного HTTP-каналу як альтернативного транспорту при недоступності брокера, та координатора Saga як механізму управління розподіленими транзакціями і відстеження стану виконання. Саме поєднання цих трьох компонентів в єдиній архітектурі - а не їх ізольоване застосування - забезпечує системі властивість безперервності доставки.

Структурно система складається з таких основних компонентів: джерела подій, що генерують та публікують події у систему; брокера повідомлень, що забезпечує буферизацію, маршрутизацію та асинхронну передачу; набору мікросервісів-споживачів, що отримують та обробляють події; модуля моніторингу стану брокера; координатора Saga, що управляє розподіленими транзакціями; модуля контролю впорядкованості; модуля ідемпотентності; та резервного HTTP-маршруту доставки. Взаємодія між цими компонентами

організована за подієво-орієнтованим принципом, де кожна бізнес-операція представлена як послідовність подій.

Центральним елементом архітектури є механізм двоканальної доставки, що забезпечує безперервність транспортного рівня. Основним каналом є брокер повідомлень RabbitMQ, що забезпечує асинхронну передачу з гарантіями доставки, буферизацію при пікових навантаженнях та підтримку черг з пріоритетами. Резервний канал на основі HTTP активується автоматично при виявленні недоступності або суттєвої деградації продуктивності основного каналу. Вибір HTTP як резервного протоколу обумовлений його широкою підтримкою в мікросервісних середовищах, незалежністю від інфраструктури брокера та відносно низькою складністю інтеграції.

Нижче наведена структурна схема архітектури системи доставки подій наведена на Рисунку 4.1.

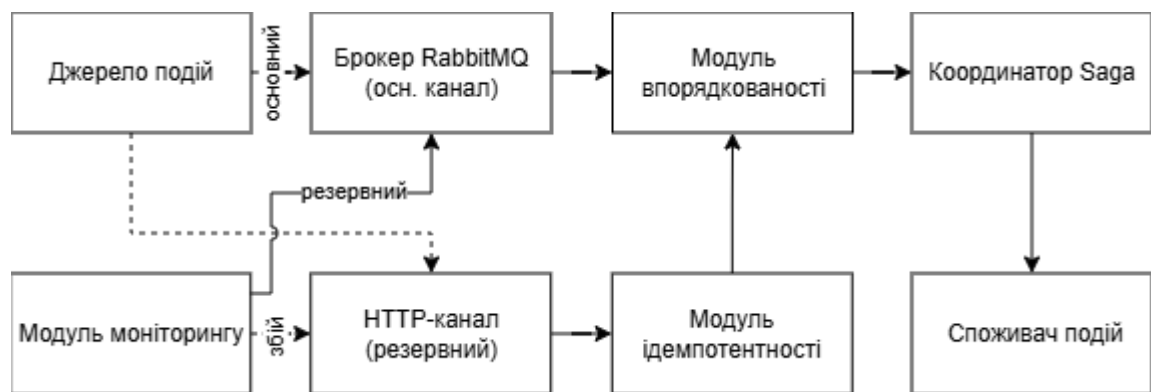


Рис. 4.1. Структурна схема архітектури системи доставки подій

Механізм перемикання між каналами реалізується модулем моніторингу стану брокера, що безперервно відстежує показники доступності та часу відповіді основного транспорту через протокол heartbeat. При перевищенні порогового значення часу відповіді або при відсутності підтвердження heartbeat фіксується факт збою і ініціюється процедура перемикання на резервний HTTP-канал. Критично важливою властивістю цього перемикання є його прозорість для вищих рівнів системи: стан змінних  $last_{seq}[key]$ .

$last_{seq}[key]$  для всіх активних ключів зберігається незмінним, тому координатор Saga та модуль впорядкованості продовжують функціонувати без будь-яких модифікацій незалежно від того, через який канал надходять події.

Процес перемикання між каналами доставки наочно представлено на Рисунку 4.2.

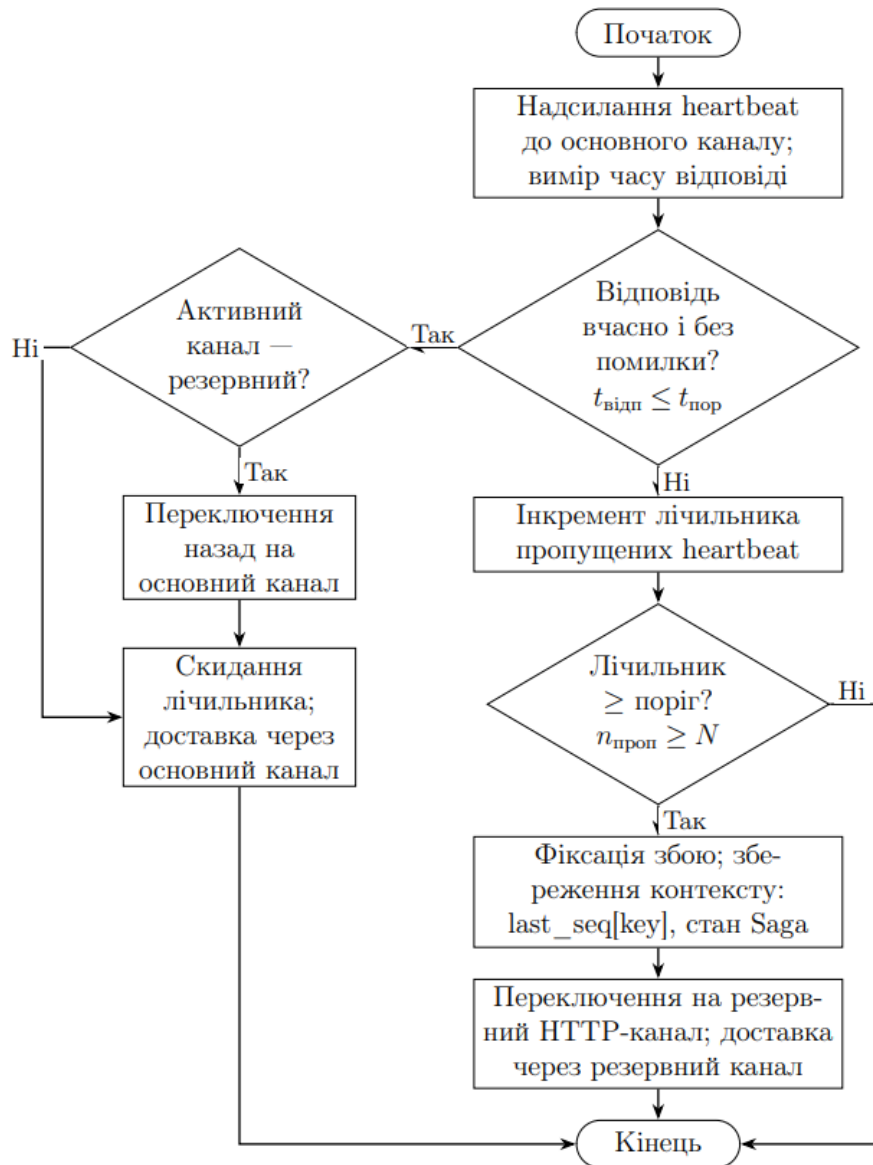


Рис. 4.2. Блок-схема алгоритму перемикання між каналами доставки

Модуль контролю впорядкованості забезпечує коректну послідовність обробки подій в умовах паралельної роботи кількох споживачів. Кожна подія формалізується у вигляді кортежу  $E = (id, key, seq, payload)$ , де  $id$  – глобально унікальний ідентифікатор,  $key$  – ключ впорядкування,  $seq$  – монотонно

зростаючий порядковий номер у межах ключа, *payload* – корисне навантаження. Для кожного ключа підтримується змінна стану  $last_{seq}[key]$ , і подія допускається до виконання лише за умови  $E.seq = last_{seq}[key] + 1$ . При надходженні події з більшим порядковим номером вона тимчасово буферизується до отримання всіх попередніх подій.

Схему обробки події модулем впорядкованості наведено на Рисунку 4.3.

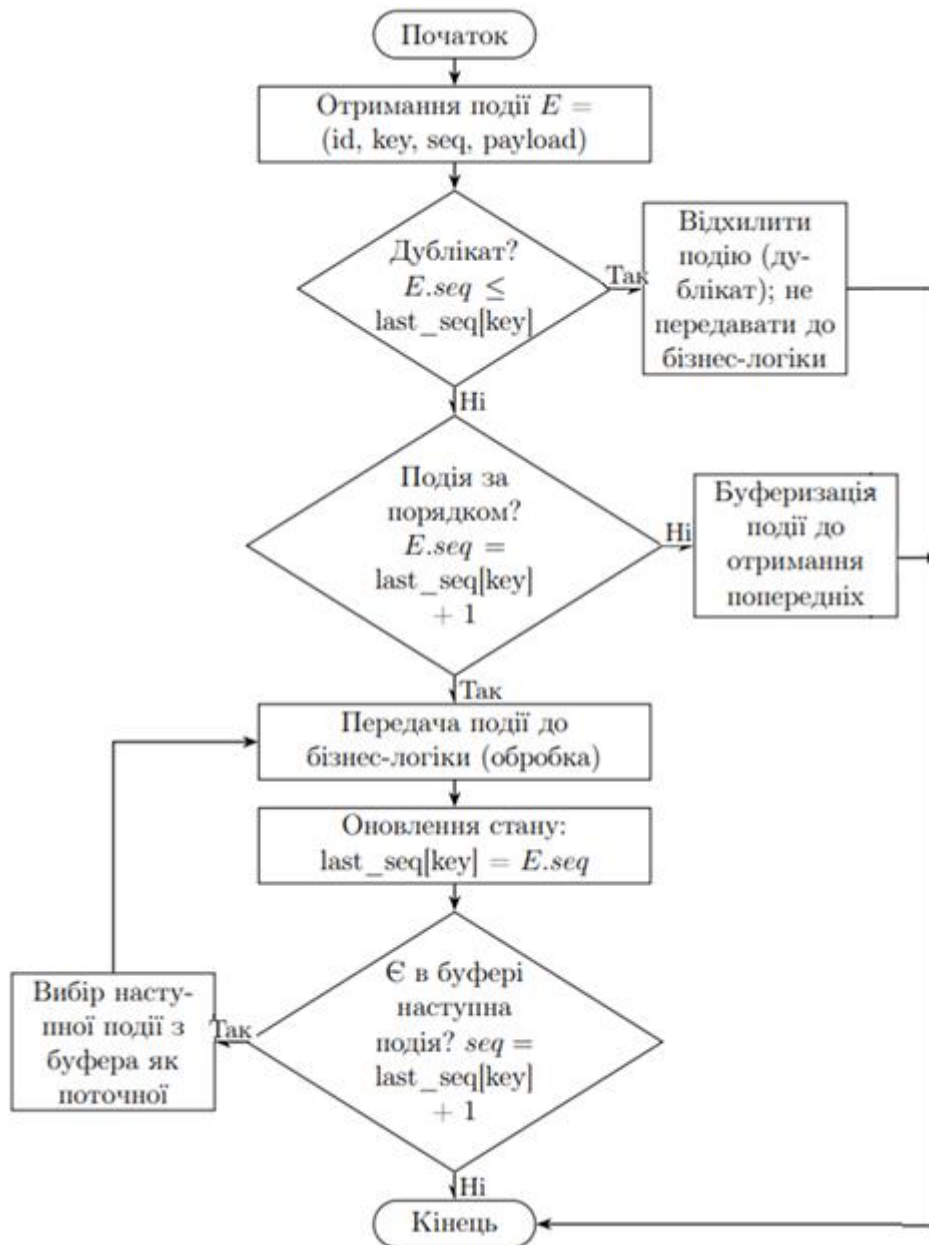


Рис. 4.3. Блок-схема алгоритму обробки події модулем контролю впорядкованості

Модуль ідемпотентності діє у тісній взаємодії з модулем впорядкованості і забезпечує захист від некоректного дублювання бізнес-операцій. Повторно доставлені події, для яких виконується умова  $E.seq \leq last_{seq}[key]$ , ідентифікуються як дублікати на рівні перевірки порядкового номера і не передаються до бізнес-логіки, що дозволяє безпечно використовувати механізми повторної доставки як на рівні основного брокера, так і при використанні резервного HTTP-каналу, без ризику некоректного дублювання операцій.

Координатор Saga відповідає за управління послідовністю виконання кроків розподіленої транзакції. Кожна подія асоціюється з конкретним кроком Saga через ключ впорядкування, і координатор відстежує стан виконання всього ланцюжка транзакції. Для кожного кроку визначено пару операцій – пряма операція та компенсаційна операція на випадок помилки – що дозволяє системі повертатися до узгодженого стану при будь-якому сценарії збою. Координатор Saga функціонує незалежно від транспортного рівня і отримує події однаково як від брокера, так і через HTTP-канал, що є принциповою властивістю запропонованої архітектури.

Запропонована архітектура відрізняється від відомих рішень насамперед тим, що забезпечує транспарентне резервування транспортного рівня без втрати контексту виконання розподілених транзакцій. У більшості існуючих підходів перемикання на резервний канал або вимагає повного перезапуску незавершених транзакцій, або не підтримується взагалі на рівні архітектури. У запропонованому рішенні стан координатора Saga та стан модуля впорядкованості є повністю незалежними від транспортного рівня, що дозволяє здійснювати перемикання каналів без переривання бізнес-процесів і без втрати гарантій впорядкованості.

Відомі підходи до забезпечення надійності доставки подій у мікросервісних системах можна розділити на три категорії. Перша категорія охоплює механізми повторної доставки на рівні брокера, що забезпечують at-least-once семантику при тимчасових збоях, однак не вирішують проблему тривалої недоступності брокера. Друга категорія охоплює патерн Saga у різних варіантах реалізації, що забезпечує eventual consistency при збоях на рівні бізнес-

логіки, однак не вирішує проблему транспортного рівня. Третя категорія охоплює рішення з резервуванням транспортного рівня через дублювання брокерів або використання альтернативних протоколів, однак без збереження стану розподілених транзакцій при переключенні.

Принципова відмінність запропонованого методу від кожної з цих категорій та від їхніх відомих комбінацій полягає у наступному. При застосуванні механізму повторної доставки спільно з патерном Saga типова реалізація передбачає, що стан Saga зберігається у сховищі, яке є функціонально незалежним від брокера, однак механізм доставки нових подій до незавершених транзакцій і механізм активації компенсаційних операцій залишаються прив'язаними до брокера. При відмові брокера незавершені транзакції не отримують нових подій і не можуть бути завершені або компенсовані до моменту відновлення брокера, що призводить до блокування бізнес-процесів на час відновлення. У разі якщо відновлення брокера потребує повного перезапуску, незавершені транзакції, як правило, вимагають ручного втручання або автоматичного перезапуску з початкового стану.

Запропонована архітектура вирішує цю проблему шляхом архітектурного відокремлення двох функцій, які у традиційних підходах неявно поєднані: функції транспортування події між компонентами та функції відстеження прогресу виконання транзакції. Відокремлення досягається через введення спільного механізму стану `lastseqkey`, що не залежить від конкретного транспортного каналу і є єдиним джерелом істини щодо прогресу виконання для обох каналів одночасно. Завдяки цьому перемикання між каналами є дійсно прозорим: координатор Saga та модуль впорядкованості продовжують роботу без жодних змін, оскільки їхній стан визначається виключно значенням `lastseqkey`, а не тим, через який канал надійшла остання подія.

Задача, що вирішується в даному розділі, формулюється таким чином: необхідно розробити архітектуру мікросервісної системи доставки подій, яка за умов відмови основного транспортного каналу забезпечує: автоматичне перемикання на резервний канал з часом перемикання, що не перевищує часу виявлення збою більш ніж вдвічі; збереження стану всіх активних розподілених



транзакцій без необхідності їх перезапуску; дотримання гарантій впорядкованості та ідемпотентності через резервний канал у тому самому обсязі, що і через основний; відновлення повного функціонування після усунення збою без ручного втручання. Виконання всіх чотирьох умов одночасно і є критерієм успішності запропонованого архітектурного рішення.

Кількісно очікуваний результат формулюється через два ключових показника.

Перший показник – середній час відновлення після збою MTTR - визначається сумою трьох складових: часу виявлення збою  $T_{detect}$ , часу перемикання на резервний канал  $T_{switch}$  та часу відновлення незавершених кроків Saga  $T_{reprocess}$ .

На відміну від базової архітектури, де до цієї суми додається час повного перезапуску системи  $T_{restart}$ , запропонована архітектура виключає цей компонент, що є структурним джерелом скорочення MTTR, незалежним від конкретних параметрів реалізації.

Другий показник – ймовірність успішної доставки події  $P_{deliver}$  – визначається ймовірностями збою обох каналів незалежно один від одного, що при реалістичних значеннях параметрів дає ймовірність, що наближається до 1.

Таблиця 4.1 систематизує порівняння запропонованого підходу з відомими альтернативами за ключовими критеріями. Продемонстровано, що запропонований метод є єдиним з розглянутих, що одночасно задовольняє всім семи критеріям.

Жоден із розглянутих підходів або їхніх попарних комбінацій не забезпечує одночасно збереження стану Saga, впорядкованості та ідемпотентності через резервний канал без необхідності перезапуску транзакцій.

Саме ця сукупність властивостей є предметом наукової новизни даного розділу.

Порівняльний аналіз підходів до забезпечення безперервності доставки  
подій

Критерій	Механізм повторної доставки	Saga без резервування	Резервування без Saga	Запропонований метод
Відновлення при збої брокера	Часткове (тимчасові збої)	Відсутнє	Є	Є
Збереження стану Saga при збої	Неприйнятно	Так (але заблоковано)	Ні	Так (і активно)
Впорядкованість через резервний канал	Не застосовується	Не застосовується	Відсутня	Забезпечена
Ідемпотентність через резервний канал	Часткова	Не застосовується	Відсутня	Забезпечена
Необхідність перезапуску транзакцій	Ні	Так	Так	Ні
MTTR	Висока	Висока	Середня	Низька
Ймовірність доставки	Помірна	Помірна	Висока	Висока

Завданням розділу є розробка методу, що одночасно забезпечує чотири властивості, жодна з відомих архітектур не реалізує в сукупності: гарантовану доставку подій через резервний канал при збої основного брокера, збереження стану координатора Saga без перезапуску, дотримання впорядкованості подій у межах ключа незалежно від активного каналу, та ідемпотентну обробку повторних доставок через будь-який канал. Саме поєднання цих чотирьох властивостей в єдиній архітектурі є предметом наукової новизни даного розділу. Кількісним вираженням результату є скорочення середнього часу відновлення системи з 930,2 мс до 223,5 мс при одночасному підвищенні ймовірності успішної доставки події з 95,3% до 99,79%, що відповідає теоретичним оцінкам, отриманим на основі аналітичної моделі, розробленої в розділі 3.

## 4.2. Інтеграція механізмів резервування каналів та керування розподіленими процесами

Інтеграція механізмів резервування каналів із системою керування розподіленими процесами є ключовим архітектурним рішенням, що забезпечує одночасно надійність транспортного рівня і коректність виконання бізнес-логіки. Ці два аспекти є принципово різними за своєю природою – перший стосується фізичної доставки повідомлень між компонентами, другий – логічної узгодженості стану розподіленої системи – однак у запропонованому методі вони інтегровані таким чином, що кожен з них підтримує і доповнює інший.

Інтеграція реалізована на двох взаємопов'язаних рівнях. Транспортний рівень відповідає за фізичну доставку подій між компонентами системи, здійснює вибір каналу доставки, моніторинг їх стану та автоматичне перемикавання при виявленні збоїв. Рівень координації транзакцій відповідає за логічну коректність виконання розподілених бізнес-процесів, забезпечує послідовність виконання кроків транзакції незалежно від того, через який канал надійшла подія, та ініціює компенсаційні транзакції у разі помилки. Принципово важливою властивістю інтеграції є повна незалежність цих рівнів один від одного: транспортний рівень є прозорим для координатора Saga, а рівень координації не залежить від конкретного механізму доставки.

Взаємодія рівнів системи при нормальному функціонуванні та при збої брокера наочно відображена на діаграмі взаємодії компонентів (рис. 4.4).

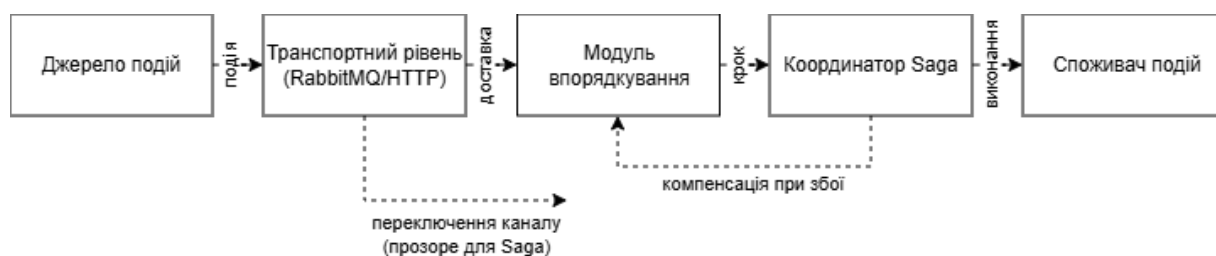


Рис. 4.4. Взаємодія рівнів системи при доставці та обробці подій

Транспортний рівень реалізує схему активного резервування, за якої основний канал (RabbitMQ) та резервний канал (HTTP) функціонують у режимі "активний – резервний". Перехід між режимами здійснюється автоматично на основі безперервного моніторингу показників доступності та латентності основного каналу. Ймовірність успішної доставки події в архітектурі з резервним каналом визначається формулою:

$$P_{proposed} = 1 - p_{loss} \cdot p_{loss}^{http} \quad (4.1)$$

де  $p_{loss}$  – імовірність втрати повідомлення в основному каналі,  $p_{loss}^{http}$  – імовірність втрати в резервному каналі.

Процес обробки події на рівні координатора Saga після її отримання через будь-який із каналів наведено на рис. 4.5.

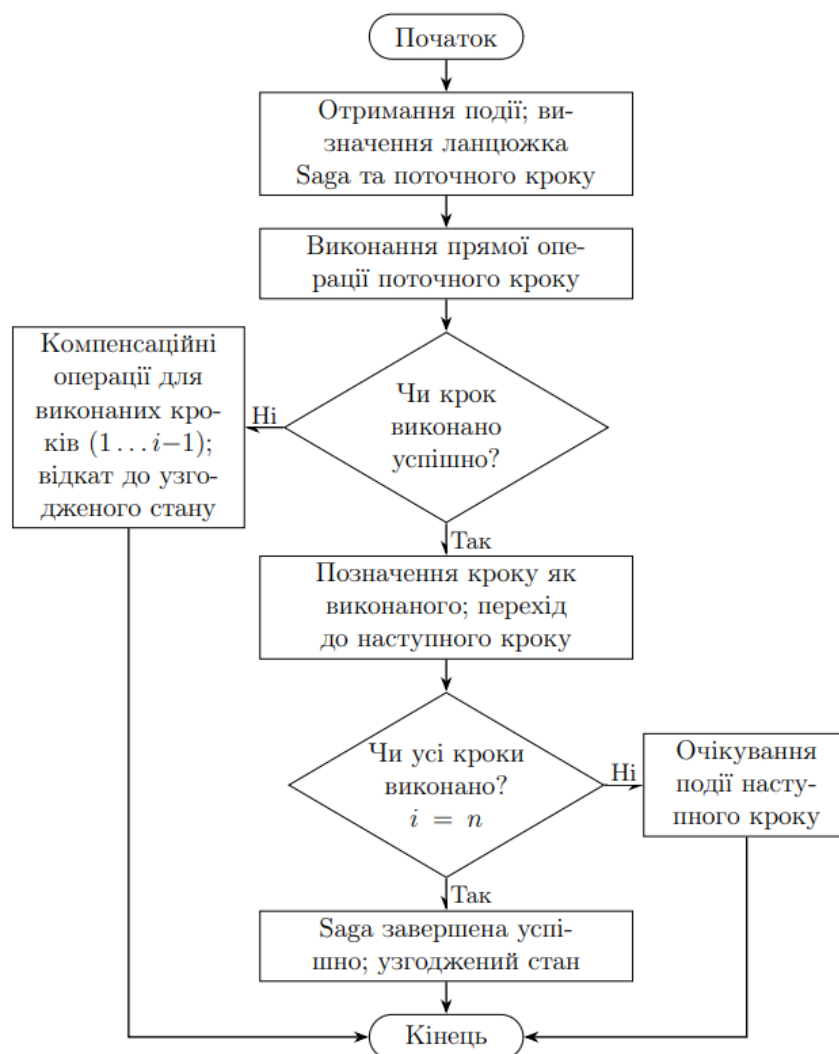


Рис. 4.5. Блок-схема алгоритму обробки події координатором Saga

При типових значеннях параметрів, що відповідають умовам симуляційного середовища, описаного у розділі 5, ймовірність втрати повідомлення в основному каналі RabbitMQ становить  $p_{loss} = 0,047$ , а ймовірність втрати в резервному HTTP-каналі  $p_{losshttp} = 0,002$ . Підставляючи ці значення у формулу, отримуємо  $P_{proposed} = 1 - 0,047 * 0,002 = 1 - 0,000094 \approx 0,9999$ . Практично виміряне значення у симуляційному середовищі становить 99,8-99,9%, що дещо нижче теоретичної межі через граничні ефекти синхронізації при переключенні каналів та стохастичні затримки в мережевому середовищі. Для порівняння, базова архітектура без резервування дає  $P_{base} = 1 - p_{loss} = 1 - 0,047 = 0,953$ , тобто 95,3%. Таким чином, введення резервного каналу підвищує ймовірність доставки з 95,3% до 99,8-99,9%, що відповідає скороченню ймовірності втрати події більш ніж у 40 разів.

Рівень координації транзакцій через патерн Saga забезпечує коректність виконання розподілених бізнес-процесів за умов можливих збоїв. Кожна розподілена транзакція декомпозується на послідовність локальних кроків, кожен з яких має відповідну компенсаційну операцію. Ймовірність завершення всіх кроків обробки події:

$$P_{complete} = \prod_{i=1}^n p_i, \quad (4.2)$$

де  $p_i$  – ймовірність успішного виконання  $i$ -го кроку.

При виникненні помилки на будь-якому кроці система виконує компенсаційні транзакції для всіх попередніх успішно виконаних кроків, відновлюючи систему до узгодженого стану та забезпечуючи властивість eventual consistency.

Для типового ланцюжка Saga, що складається з  $n=4$  кроків з ймовірністю успішного виконання кожного кроку  $p = 0,99$ , ймовірність успішного завершення всього ланцюжка без будь-яких збоїв становить  $P_{complete} = 0,99^4 \approx 0,961$ , тобто близько 96,1%. У базовій архітектурі без резервування збій брокера в процесі виконання ланцюжка призводить до того, що ланцюжок залишається у

незавершеному стані до повного відновлення системи, тобто ефективна ймовірність завершення знижується до

$$P_{complete} * 1 - p_{failure - during - chain}, \quad (4.3)$$

де  $p_{failure - during - chain}$  є ймовірністю збою брокера саме в процесі виконання ланцюжка.

У запропонованій архітектурі збій брокера не перериває ланцюжок, оскільки координатор Saga продовжує роботу через резервний канал. Ефективна ймовірність завершення в цьому випадку залишається близькою до  $P_{complete}$  незалежно від ймовірності збою брокера, що є принциповою перевагою запропонованого підходу для систем з жорсткими вимогами до рівня обслуговування.

Стани координатора Saga та можливі переходи між ними при виконанні розподіленої транзакції наведено на рис. 4.6.

Взаємодія транспортного рівня та рівня координації транзакцій при збої брокера характеризується такою послідовністю подій. Модуль моніторингу виявляє недоступність брокера та зберігає поточний контекст виконання - стан змінних  $last_{seq}[key]$  для всіх активних ключів та поточний стан кожного активного ланцюжка Saga. Транспортний рівень переключається на HTTP-канал, при цьому координатор Saga не отримує жодних сигналів про зміну транспорту і продовжує отримувати події у тому самому форматі.



Рис. 4.6. Діаграма станів координатора Saga при виконанні розподіленої транзакції

Незавершені кроки Saga, що перебували в процесі виконання на момент збою, відновлюються після завершення перемикавання та обробляються в порядку, визначеному механізмом впорядкованості.

Ключові параметри інтеграції двох рівнів за нормальної роботи та при збої систематизовано у таблиці нижче.

Таблиця 4.2

Порівняння параметрів функціонування рівнів системи за різних сценаріїв

Параметр	Нормальна робота	Збій брокера	Відновлення
Активний канал	RabbitMQ	HTTP	RabbitMQ
Стан $last_{seq}[key]$	Актуальний	Збережений	Актуальний
Стан координатора Saga	Активний	Активний	Активний
Незавершені кроки Saga	Виконуються	Очікують	Відновлюються
Час перемикавання	—	$T_{switch}$	$T_{restore}$
Гарантія впорядкованості	Так	Так	Так
Гарантія ідемпотентності	Так	Так	Так

Важливою властивістю запропонованої інтеграції є те, що обидва рівні зберігають свої гарантії незалежно від стану транспорту. Гарантія впорядкованості забезпечується тим, що стан  $last_{seq}[key]$  зберігається незалежно від каналу і перевіряється для кожної події незалежно від того, через який канал вона надійшла. Гарантія ідемпотентності зберігається з тієї самої причини — механізм дедуплікації є частиною рівня координації, а не транспортного рівня, тому зміна каналу не впливає на його функціонування.

Порівняння запропонованого підходу з відомими рішеннями щодо інтеграції резервування каналів та управління транзакціями наведено у таблиці нижче.

Ймовірність успішної доставки події в архітектурі з резервним каналом визначається за формулою

$$P_{proposed} = 1 - p_{loss} * p_{loss_{http}}, \quad (4.4)$$

де  $p_{loss}$  є ймовірністю втрати повідомлення в основному каналі RabbitMQ, а  $p_{loss_{http}}$  є ймовірністю втрати в резервному HTTP-каналі.

При типових значеннях  $p_{loss} = 0,047$  та  $p_{loss_{http}} = 0,002$ , що відповідають вимірним характеристикам симуляційного середовища, отримуємо  $P_{proposed} = 1 - 0,047 * 0,002 = 1 - 0,000094 \approx 0,9999$ , тобто ймовірність доставки наближається до 99,99%. Практично вимірне значення становить 99,8-99,9%, що дещо нижче теоретичної межі через граничні ефекти синхронізації при переключенні каналів, однак значно перевищує базове значення 95,3%.

Принципова відмінність запропонованого рішення від класичних підходів до резервування полягає в тому, що перемикання каналів є прозорим для рівня координації транзакцій. У традиційних архітектурах відмова брокера призводить до необхідності перезапуску незавершених транзакцій з початкового стану, оскільки стан координатора Saga прив'язаний до транспортного рівня через внутрішні черги або буфери брокера. У запропонованій архітектурі координатор Saga зберігає стан через незалежний механізм lastseqkey, який не залежить від активного транспортного каналу, тому при переключенні на HTTP-канал продовжується виконання тих самих кроків транзакції з тієї самої точки, де вона зупинилась.

Таблиця 4.3

Порівняльний аналіз підходів до інтеграції резервування каналів та управління транзакціями

Підхід	Резервний канал	Збереження стану Saga	Впорядкованість при збої	Ідемпотентність
Лише брокер без резервування	Відсутній	Залежить від брокера	Не гарантується	Часткова
Резервний канал без інтеграції	Так	Втрачається	Не гарантується	Часткова
Saga без резервного каналу	Відсутній	Так	Так	Так
Запропонований метод	Так	Так	Так	Так

Таким чином, запропонована інтеграція є якісно новим рішенням порівняно з відомими підходами, оскільки одночасно забезпечує всі чотири



ключові властивості: наявність резервного каналу, збереження стану координатора Saga при збоях, гарантію впорядкованості та гарантію ідемпотентності. Жоден із розглянутих підходів не забезпечує усіх чотирьох властивостей одночасно, що підтверджує наукову новизну та практичну цінність запропонованого методу.

#### 4.3. Забезпечення живучості та зменшення часу відновлення системи

Живучість (liveness) мікросервісної системи є однією з найважливіших характеристик відмовостійкості і визначає здатність системи гарантувати завершення всіх ініційованих процесів обробки подій незалежно від наявності часткових збоїв окремих компонентів. На відміну від властивості безпеки (safety), яка забороняє настання небажаних станів, живучість гарантує настання бажаних станів – а саме, що кожна ініційована подія рано чи пізно буде успішно оброблена або коректно компенсована. У контексті систем впорядкованої доставки подій живучість набуває особливого значення, оскільки порушення порядку обробки або незавершення окремого кроку Saga може призводити до накопичення неузгодженого стану у всій ланці залежних транзакцій.

У запропонованому методі живучість системи досягається через поєднання трьох взаємодоповнюючих механізмів: автоматичного виявлення збою та перемикання на резервний канал, збереження і відновлення стану незавершених транзакцій через координатор Saga, та механізму ідемпотентної обробки, що захищає від некоректного дублювання операцій при повторних спробах. Кожен з цих механізмів є необхідним, але недостатнім сам по собі – лише їх спільна дія забезпечує повноцінну живучість системи.

Середній час відновлення системи (Mean Time To Recovery, MTTR) є кількісним показником живучості і визначає середній проміжок часу між настанням збою та повним відновленням нормального функціонування системи. Для базової архітектури без резервування:

$$MTTR_{base} = T_{detect} + T_{restart} \quad (4.5)$$

де  $T_{detect}$  – час виявлення збою,  $T_{restart}$  – час повного перезапуску системи та повторної обробки незавершених подій.

Для запропонованої архітектури:

$$MTTR_{proposed} = T_{detect} + T_{switch} + T_{reprocess} \quad (4.6)$$

де  $T_{switch}$  – час перемикавання на резервний канал,  $T_{reprocess}$  – час завершення незавершених кроків Saga після перемикавання.

Оскільки перемикавання відбувається без повного перезапуску системи і з збереженням контексту виконання, виконується нерівність  $MTTR_{proposed} < MTTR_{base}$ . Алгоритм виявлення збою та ініціювання відновлення наведено на Рисунку 4.7.

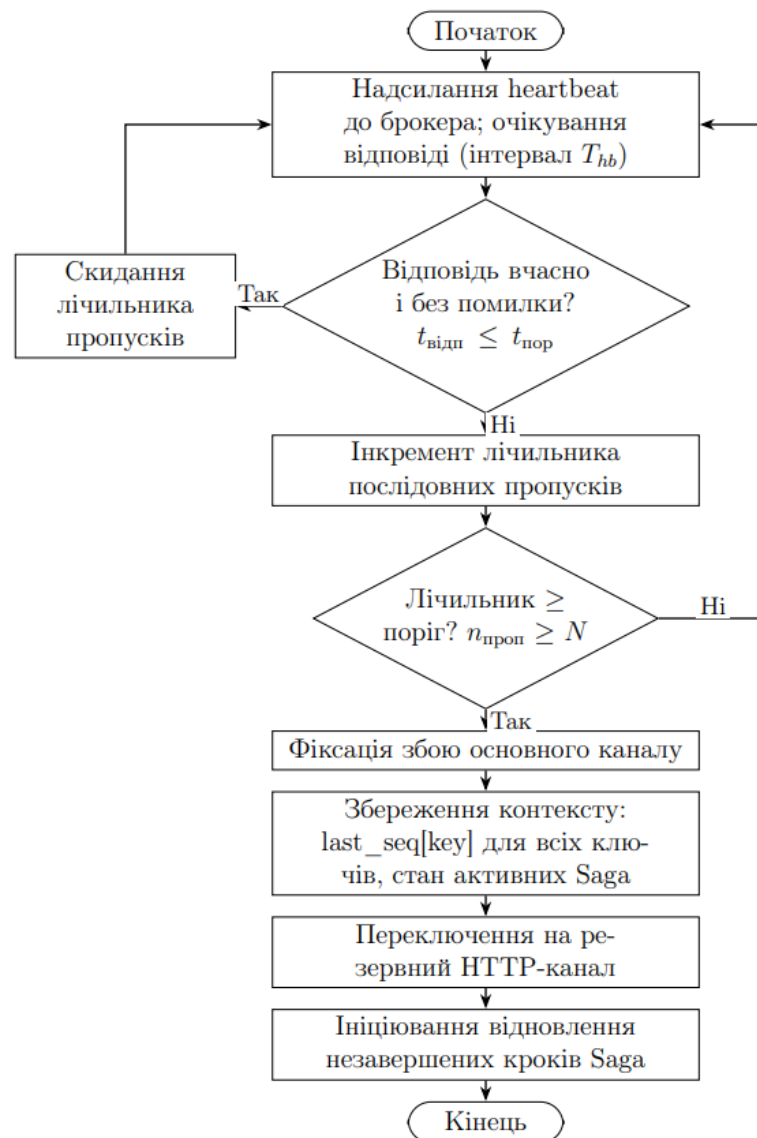


Рис. 4.7. Блок-схема алгоритму виявлення збою та ініціювання відновлення

Після виявлення збою система зберігає повний контекст виконання - стан змінних  $last_{seq}[key]$  для всіх активних ключів та поточний стан кожного активного ланцюжка Saga - і ініціює перемикання на резервний HTTP-канал. Час перемикання  $T_{switch}$  включає час збереження контексту, час реконфігурації маршрутизатора подій та час встановлення HTTP-з'єднань. Завдяки тому, що контекст повністю збережений, ні координатор Saga, ні модуль впорядкованості не потребують перезапуску – вони продовжують роботу з того самого стану, що й до збою.

Процес відновлення незавершених кроків Saga після перемикання на резервний канал наведено на рис. 4.8.

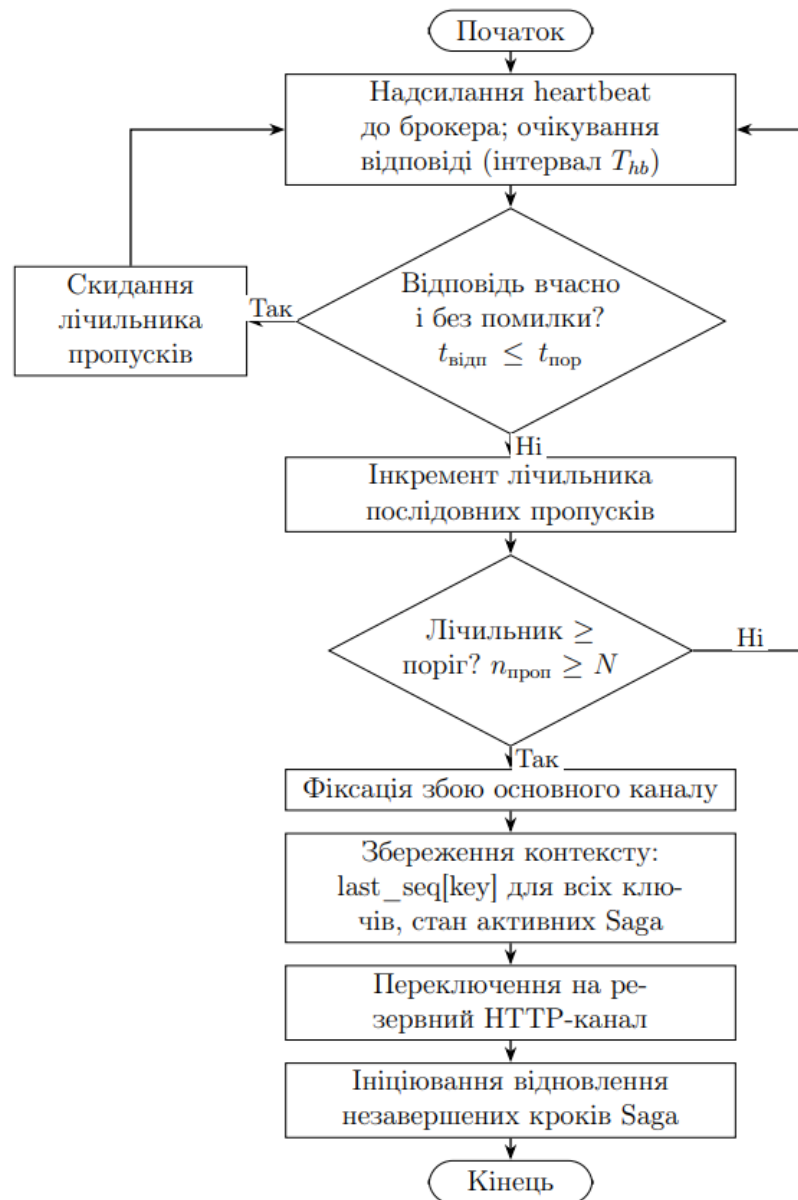


Рис. 4.8. Блок-схема алгоритму відновлення незавершених кроків Saga

Механізм відновлення незавершених кроків Saga є критично важливим для забезпечення живучості. При завантаженні збереженого контексту координатор перевіряє кожен активний ланцюжок Saga на наявність незавершених кроків і відновлює їх виконання через резервний канал. Перевірка ідемпотентності при цьому є обов'язковою: оскільки збій міг відбутися після виконання кроку, але до збереження підтвердження, повторне виконання без перевірки могло б призвести до дублювання операції. Завдяки механізму перевірки  $E.seq \leq last_{seq}[key]$  такі ситуації коректно обробляються без будь-яких додаткових дій з боку бізнес-логіки.

Часова діаграма процесу відновлення системи з визначенням складових MTTR наведена на рис. 4.9.

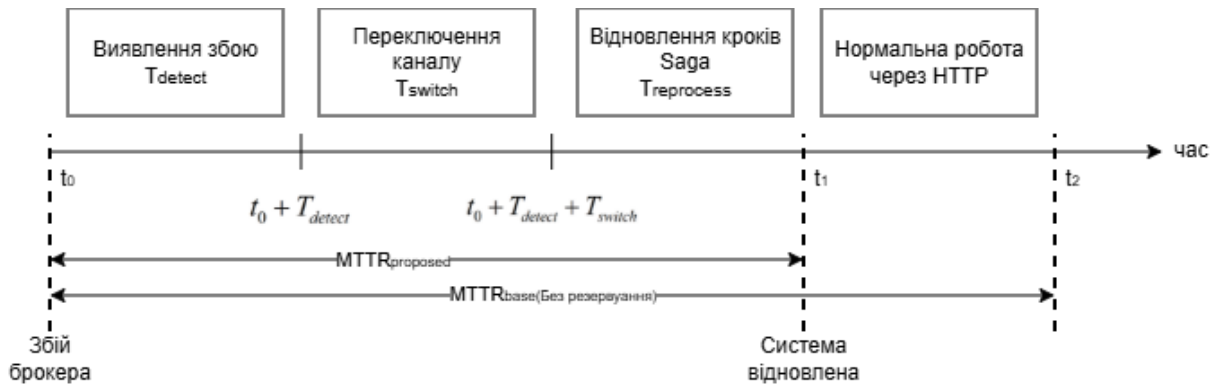


Рис. 4.9. Часова діаграма процесу відновлення системи та складові MTTR

Ймовірність успішної доставки події при двоканальній архітектурі, як показано у підрозділі 4.2, визначається добутком ймовірностей відмови обох каналів і є суттєво вищою за аналогічний показник однока-нальної системи. Такий ефект безпосередньо впливає на живучість: чим вища ймовірність доставки, тим менша ймовірність ситуації, коли подія взагалі не дійде до обробника і потребуватиме відновлення через механізм Saga. Таким чином, резервування транспортного рівня та механізм Saga є взаємодоповнюючими засобами забезпечення живучості - перший мінімізує ймовірність втрати події, другий гарантує коректне відновлення у разі, якщо втрата все ж відбулась.

Коефіцієнт доступності системи, що характеризує частку часу, протягом якої система функціонує нормально:

$$A = \frac{MTTF}{MTTF + MTTR}, \quad (4.7)$$

де  $MTTF$  – середній час між відмовами. При фіксованому  $MTTF$  зменшення  $MTTR$  безпосередньо підвищує коефіцієнт доступності, що є критично важливим для систем із жорсткими вимогами до рівня обслуговування. Скорочення  $MTTR$  на 72,5% при незмінному  $MTTF$  призводить до суттєвого зростання  $A$ , що підтверджує практичну цінність запропонованого підходу.

Порівняльний аналіз показників живучості та доступності запропонованої архітектури відносно базової наведено у Таблиці 4.4.

Таблиця 4.4

Порівняння складових часу відновлення базової та запропонованої архітектур

Показник	Базова архітектура	Запропонована архітектура	Покращення
$T_{detect}$ , мс	$72,9 \pm 17,1$	$72,9 \pm 17,1$	однаковий механізм
$T_{switch}$ , мс	—	$91,0 \pm 20,4$	—
$T_{reprocess}$ , мс	—	$59,7 \pm 14,4$	—
$T_{restart}$ , мс	857,3	—	—
MTTR, мс	$930,2 \pm 153,8$	$223,5 \pm 33,7$	-76,0 %
$P_{deliver}$ , %	94,79	99,79	+5,00 п.п.

$T_{restart}$  базової архітектури визначено як  $MTTR_{base} - T_{detect} = 930,2 - 72,9 = 857,3$  мс. Складова  $T_{detect}$  є однаковою для обох архітектур, оскільки використовується той самий механізм виявлення збою (heartbeat). Прочерк (—) означає, що відповідна складова відсутня в архітектурі за побудовою.

Час відновлення запропонованої архітектури формується з трьох послідовних складових: часу виявлення збою  $T_{detect} = 72,9$  мс, часу перемикання на резервний НТТР-канал  $T_{switch} = 91,0$  мс та часу повторного опрацювання незавершених подій  $T_{reprocess} = 59,7$  мс.

Сумарний час відновлення за формулою (4.6) становить  $MTTR_{proposed} = 72,9 + 91,0 + 59,7 = 223,6 \approx 223,5$  мс. На відміну від цього, базова архітектура потребує повного перезапуску оброблення після виявлення збою, тому її час відновлення дорівнює  $MTTR_{base} = T_{detect} + T_{restart} = 72,9 + 857,3 = 930,2$  мс, де  $T_{restart} = 857,3$  мс – час повторного запуску та відновлення контексту транзакцій. Таким чином, запропонована архітектура скорочує час відновлення на 76,0, % порівняно з базовою (з 930,2 мс до 223,5 мс), що досягається завдяки переключенню на резервний канал без перезапуску транзакцій.

Узагальнення результатів, отриманих у розділах 2 і 3, а також розроблених у підрозділах 4.1–4.3 механізмів відмовостійкості дозволяє сформулювати третій науковий результат дисертаційної роботи – архітектуру забезпечення безперервності доставки подій у мікросервісних системах. Запропонована архітектура є розвитком і системною інтеграцією методу упорядкованого паралелізму та аналітичної моделі його продуктивності в умовах відмов і деградації транспортної інфраструктури.

Перший науковий результат утворює алгоритмічне ядро архітектури. Розроблений метод забезпечує послідовне оброблення подій у межах одного логічного ключа та паралельне оброблення подій, що належать різним ключам. Підтримання номера останньої обробленої події, буферизація подій із розривом послідовності, відновлення пропущених подій і відсіювання дублікатів застосовуються незалежно від того, через який транспортний канал надійшла подія.

Другий науковий результат утворює аналітичну основу проєктування та конфігурації архітектури. Розроблена модель дозволяє оцінювати вплив кількості паралельних обробників, нерівномірності розподілу подій між ключами, повторних доставок, буферизації, конкуренції за спільні ресурси та використання резервного каналу на пропускну здатність системи. На її основі визначаються допустимий рівень паралелізму, межа масштабованості та умови стабільної роботи архітектури.

Третій науковий результат розширює перші два результати механізмами транспортного резервування та збереження контексту розподілених транзакцій.

Для цього метод упорядкованого паралелізму інтегровано з основним каналом RabbitMQ, резервним HTTP-каналом, модулем моніторингу стану брокера та координатором Saga. У результаті сформовано цілісну архітектуру, яка забезпечує не лише впорядкованість і масштабованість оброблення, а й продовження незавершених бізнес-процесів після відмови основного транспортного каналу.

Запропонована архітектура ґрунтується на функціональному відокремленні транспортного рівня від рівня контролю стану обробки. Транспортний рівень відповідає за фізичне передавання подій і містить два незалежні канали: основний асинхронний канал на основі брокера RabbitMQ та резервний синхронний HTTP-канал. Рівень контролю стану охоплює модуль упорядкування подій, механізм ідемпотентності та координатор Saga. Стан цього рівня не залежить від активного транспортного каналу, тому перемикання між RabbitMQ та HTTP не змінює логічної позиції виконання бізнес-процесу.

До складу запропонованої архітектури входять основний і резервний канали доставки, модуль моніторингу доступності брокера, маршрутизатор подій, незалежне сховище стану обробки, модуль контролю порядку та ідемпотентності, координатор Saga і множина паралельних обробників подій. Взаємодія цих компонентів організована так, щоб зміна транспортного каналу не впливала на логіку впорядкування, стан розподілених транзакцій і результати вже виконаних бізнес-операцій.

Основний канал використовується у штатному режимі та забезпечує асинхронне передавання, буферизацію і повторну доставку повідомлень через RabbitMQ. Резервний HTTP-канал активується тоді, коли модуль моніторингу виявляє недоступність брокера або перевищення встановленого порогового значення затримки. Отже, підставою для перемикання є не лише повна відмова RabbitMQ, а й часткова деградація його роботи. Це запобігає тривалому накопиченню подій у черзі за формальної доступності брокера, але за неприйнятного часу відповіді.

Модуль моніторингу періодично надсилає контрольні heartbeat-повідомлення до RabbitMQ та аналізує факт отримання відповіді і тривалість її

очікування. Рішення про відмову основного каналу приймається після досягнення заданої кількості послідовних невдалих перевірок. Використання кількох перевірок зменшує ймовірність хибного перемикання, спричиненого одиничним сплеском мережевої затримки.

Ключовою властивістю запропонованої архітектури є інваріантність стану обробки відносно транспортного каналу. Для кожного логічного ключа підтримується номер останньої коректно обробленої події та буфер подій, отриманих із розривом послідовності. Ці дані зберігаються незалежно від RabbitMQ та HTTP і використовуються спільно для обох маршрутів доставки.

Подія допускається до бізнес-обробки лише тоді, коли її порядковий номер безпосередньо продовжує вже оброблену послідовність для відповідного ключа. Якщо номер події перевищує очікуване значення, вона тимчасово розміщується в буфері до надходження пропущених подій. Якщо номер події не перевищує номер останньої обробленої події, така подія визначається як повторно доставлена і не передається до бізнес-логіки.

Однакові правила застосовуються до подій, отриманих через RabbitMQ і через HTTP. Тому зміна транспортного каналу не порушує порядку та не спричиняє повторного виконання бізнес-операцій. Після обробки чергової очікуваної події система послідовно перевіряє буфер і відновлює обробку раніше відкладених подій.

Стан кожної розподіленої транзакції містить її ідентифікатор, номер останнього підтвердженого кроку, поточний статус, відомості про виконані операції та доступні компенсаційні дії. Координатор Saga взаємодіє зі сховищем стану безпосередньо, а не через брокер повідомлень. Унаслідок цього відмова RabbitMQ не призводить до втрати інформації про вже виконані кроки транзакції.

Після активації резервного каналу координатор продовжує виконання з наступного після останнього підтвердженого кроку, а не ініціалізує весь ланцюжок повторно. Якщо збій відбувся після фактичного виконання операції, але до надходження її підтвердження, перевірка порядкового номера події та



стану виконаних кроків запобігає повторному застосуванню відповідної бізнес-операції.

Загальна послідовність функціонування запропонованої архітектури охоплює штатний режим, виявлення збою, перемикання на резервний канал, продовження незавершених операцій і повернення до основного каналу. У штатному режимі події надходять через RabbitMQ, після чого проходять спільний контур перевірки порядку, ідемпотентності та стану Saga. Одночасно модуль моніторингу виконує heartbeat-перевірки брокера.

Після підтвердження недоступності або критичної деградації RabbitMQ маршрутизатор спрямовує нові події через резервний HTTP-канал. Події, отримані через цей канал, обробляються з використанням того самого стану послідовності, тих самих буферів і станів Saga. Отже, перемикання змінює лише спосіб фізичного передавання подій, але не змінює їхню семантику та стан бізнес-процесу.

Після відновлення RabbitMQ повернення до основного каналу здійснюється лише після підтвердження його стабільної роботи протягом установленого інтервалу. Такий підхід запобігає частим взаємним перемиканням каналів у разі нестабільного стану брокера. Події, що могли повторно надійти після відновлення RabbitMQ, перевіряються механізмом ідемпотентності та не спричиняють дублювання бізнес-операцій.

На рисунку 4.10 відображено узагальнення трьох рівнів запропонованого рішення. Алгоритмічний рівень реалізує метод упорядкованого паралелізму, розроблений у розділі 2. Аналітико-конфігураційний рівень використовує модель продуктивності розділу 3 для визначення кількості обробників і перевірки умов стабільності. Архітектурний рівень доповнює їх двоканальною доставкою, моніторингом стану брокера та координатором Saga.

Безперервність доставки в межах запропонованої архітектури визначається як здатність системи забезпечувати прогрес обробки за умови доступності принаймні одного транспортного каналу та збереження незалежного стану координації. На відміну від одноканальної системи, подія втрачається лише тоді, коли її не вдалося передати ні через основний, ні через резервний канал.

Використання двох каналів саме по собі не гарантує коректності обробки. Така гарантія досягається завдяки поєднанню транспортного резервування з незалежним станом послідовності, контролем ідемпотентності та збереженням стану Saga. Відсутність будь-якого із цих механізмів може призвести відповідно до порушення порядку, дублювання бізнес-операцій або втрати контексту незавершеної транзакції.

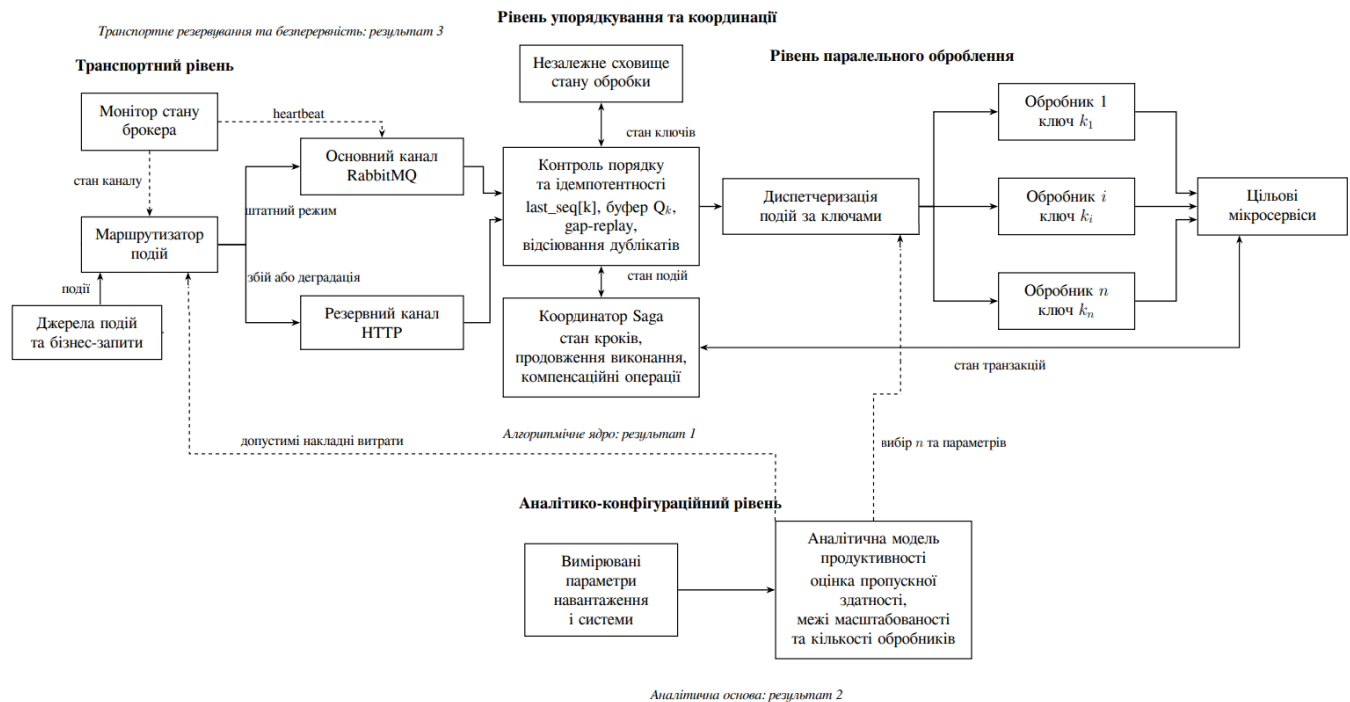


Рис. 4.10 Структурно-функціональне подання запропонованої архітектури безперервної впорядкованої паралельної доставки подій із транспортним резервуванням

Запропонована архітектура забезпечує такі взаємопов'язані властивості:

- безперервність передавання подій за доступності хоча б одного транспортного каналу;
- збереження порядку подій у межах кожного логічного ключа незалежно від маршруту доставки;
- ідемпотентне відсіювання повторно доставлених подій після перемикання каналів;
- продовження незавершених Saga-транзакцій з останнього підтвердженого кроку;

- паралельне опрацювання подій, що належать різним логічним ключам;
- скорочення часу відновлення завдяки відсутності повного перезапуску транзакцій.

Наукова новизна результату полягає не в окремому застосуванні RabbitMQ, HTTP, heartbeat-моніторингу або патерну Saga, оскільки ці технології відомі та використовуються в мікросервісних системах. Новим є спосіб їх архітектурної інтеграції, за якого основний і резервний канали використовують єдиний незалежний стан упорядкування та єдиний координатор розподілених транзакцій. Завдяки цьому транспортне перемикання відбувається без втрати контексту, повторної ініціалізації бізнес-процесу та порушення порядку подій.

На відміну від Saga choreography, запропонована архітектура підтримує централізовано узгоджений стан виконання транзакції та не потребує незалежного відновлення кожного мікросервісу. На відміну від Transactional Outbox, вона забезпечує резервний маршрут передавання подій під час недоступності брокера і не очікує його відновлення для опрацювання накопиченого набору повідомлень. Водночас Transactional Outbox зберігає перевагу атомарного запису бізнес-даних і події, тому зазначені підходи не є повністю взаємозамінними.

Межами застосовності запропонованої архітектури є одночасна недоступність RabbitMQ і резервного HTTP-каналу, втрата або пошкодження незалежного сховища стану, а також використання бізнес-операцій, для яких не забезпечено ідемпотентність. Для збоїв, тривалість яких є меншою за інтервал виявлення, перемикання може не відбутися, оскільки система не встигає підтвердити факт відмови. Крім того, надмірно часте використання HTTP-каналу збільшує середню затримку доставки, тому його необхідно розглядати саме як резервний, а не як рівнозначний основному канал постійної роботи.

Таким чином, третім науковим результатом дисертації є розроблена архітектура забезпечення безперервності доставки подій у мікросервісних системах, яка узагальнює метод упорядкованого паралелізму та аналітичну модель його продуктивності, доповнюючи їх двоканальним транспортом,

незалежним станом обробки та координатором Saga. На відміну від відомих підходів, архітектура одночасно забезпечує впорядковану паралельну обробку, аналітично обґрунтовану масштабованість і продовження незавершених розподілених транзакцій після відмови брокера без їх повного перезапуску.

Кількісна перевірка сформульованих властивостей виконується в наступних підрозділах за показниками середнього часу відновлення, частки успішно доставлених подій, частки коректно завершених Saga-транзакцій, кількості дубльованих операцій та накладних витрат за латентністю.

#### **4.4. Методика експериментального дослідження**

Метою експериментального дослідження є кількісне підтвердження трьох наукових результатів роботи в умовах, максимально наближених до характеристик реальних мікросервісних систем: методу упорядкованого паралелізму (підрозділ 2.2), аналітичної моделі пропускну здатності  $X_g(k)$  (підрозділ 3.4) та архітектури безперервної доставки (підрозділ 4.1). Дослідження організоване так, щоб усі висновки спиралися на статистично значущі відмінності, а порівняння з аналогами було відтворюваним і неупередженим.

Дослідження виконане методом подієвої імітаційної симуляції у віртуальному часі. Замість реальних затримок кожна подія отримує позначку модельного часу, а планувальник обробляє події у порядку зростання цих позначок. Такий підхід забезпечує точну відтворюваність, виключає вплив фонових навантажень операційної системи та дозволяє масштабувати кількість обробників  $k$  без спотворення вимірюваних інтервалів. Логіка обробки відтворює механізми, формалізовані в підрозділах 2.3–2.5: функцію стану  $S(k)$ , умову допуску  $s = S(k) + 1$ , буферизацію розривів із подальшим gap-replay та ідемпотентне ігнорування дублікатів.

Усі параметри симуляції відповідають типовим значенням, задокументованим у літературі та технічній документації RabbitMQ; жодне значення не підбиралося для покращення результатів. Час обробки моделюється

логнормальним розподілом (а не фіксованою константою), мережеві затримки – експоненціальним розподілом із важким хвостом, що відтворює поодинокі сплески. Базові значення наведено в таблиці 4.5.

Таблиця 4.5

Базові параметри імітаційної моделі та їх обґрунтування

Параметр	Значення	Обґрунтування
$T_p$	10 мс	Типова затримка бізнес-логіки мікросервісу (mid-range)
$\sigma_p$	4 мс	$CV = 0,4$ – реалістично для змішаного навантаження
$T_i$	0,5 мс	Перевірка last_seq як in-memory lookup
$p_r$	0,08	8% повторних доставок за семантики at-least-once (RabbitMQ)
$p_{loss}$	0,005 / 0,05	Норма / деградація мережі
$p_h$	0,02	Частка доставки через резервний HTTP-канал
$T_h$	52 мс	Накладні витрати HTTP порівняно з AMQP
Zipf $\alpha$	1,5 / 2,0	Стандарт для e-commerce і фінансових систем / посилена нерівномірність
$N_{keys}$	200	Активних ключів одночасно
$\lambda$	500/200 под./с	Помірне навантаження / стабільна точка для латентності

Окремого пояснення потребує вибір інтенсивності навантаження. Усі метрики пропускної здатності, впорядкованості та відновлення вимірюються за номінальної інтенсивності  $\lambda = 500$  под./с. Натомість метрика накладних витрат за латентністю вимірюється за зниженої інтенсивності  $\lambda = 200$  под./с. За  $\lambda = 200$  под./с система перебуває у стабільній робочій точці, тому виміряні накладні витрати відображають саме вартість резервного HTTP-каналу, а не насичення черги гарячого ключа. За номінальної інтенсивності внесок черги гарячого ключа спотворив би оцінку каналу, тому розмежування інтенсивностей є коректним методичним рішенням, а не неузгодженістю.

Метод і архітектура перевіряються у чотирьох сценаріях, що відтворюють різні режими експлуатації. Параметри сценаріїв наведено в таблиці 4.6.

Таблиця 4.6

Сценарії експериментального дослідження

Сценарій	Опис	Ключові відмінності від норми
normal	Штатний режим	$p_{loss} = 0,005$ , Zipf $\alpha = 1,5$
high_load	Підвищене навантаження	Збільшена інтенсивність вхідного потоку
fault	Збій із втратою подій	$p_{loss} = 0,05$ (деградація мережі)
skewed	Посилена нерівномірність ключів	Zipf $\alpha = 2,0$

Для забезпечення відтворюваності використовуються фіксовані seed-значення генератора псевдовипадкових чисел. Експерименти з впорядкованості (Результат 1) виконуються на семи незалежних запусках із наборами  $seed = \{42, 123, 456, 789, 1337, 2024, 9999\}$ . Експерименти з пропускну здатності та калібрування моделі (Результат 2) – на десяти запусках: до зазначених семи додаються  $seed = \{314, 271, 1618\}$ .

Експерименти з відновлення після збоїв (Результат 3) виконуються на 50 незалежних запусках на сценарій. Початкові перехідні процеси виключаються з підрахунку метрик: перші 500 подій для Результату 1 та перші 1000 подій для Результату 2 (warm-up period).

Усі агреговані показники подаються у форматі «середнє  $\pm$  стандартне відхилення» із наведенням 95 % довірчого інтервалу.

Статистична значущість відмінностей між запропонованим методом і аналогами оцінюється непараметричним критерієм Манна–Уїтні (Mann–Whitney U), оскільки розподіли метрик не є гарантовано нормальними.

Для одночасного порівняння кількох груп застосовується критерій Краскела–Уолліса. Порогом значущості прийнято  $p < 0,05$ . Величина ефекту оцінюється коефіцієнтом  $d$  Коена, що дозволяє відрізнити статистично значущі,

але практично незначущі відмінності від суттєвих. Усі статистичні розрахунки виконано засобами бібліотеки `scipy.stats`.

Аналоги A1 (Kafka partition-key ordering), A2 (RabbitMQ standard), A3 (Saga choreography) та A4 (Transactional Outbox) реалізовані відповідно до їх типової поведінки, описаної в підрозділі 1.6, із збереженням їхніх природних переваг. Зокрема, для A2 не вводиться жодного штучного контролю порядку, завдяки чому за нормального навантаження A2 демонструє вищу пропускну здатність, ніж запропонований метод. Таке представлення є принциповим, адже воно унеможливорює закид у нечесному заниженні характеристик аналогів і підвищує довіру до отриманих відмінностей.

Описана методика забезпечує методологічну основу для підрозділів 4.5–4.7, у яких послідовно подаються результати оцінювання продуктивності й масштабованості, аналізу коректності впорядкування та порівняння архітектур відновлення.

Поєднання реалістичних розподілів параметрів, багаторазових запусків із фіксованими seed-значеннями та непараметричних статистичних критеріїв гарантує, що подальші висновки щодо наукової новизни Результатів 1–3 спираються на відтворювані й статистично підтверджені дані.

#### **4.5. Оцінка продуктивності, латентності та масштабованості системи**

У цьому підрозділі перевіряється другий науковий результат – аналітична модель пропускну здатності  $X_g(k)$ , побудована в підрозділі 3.4. Перевірка полягає у зіставленні передбачень моделі з даними імітаційної симуляції на широкому діапазоні кількості паралельних обробників  $k$ , а також у визначенні асимптотичної межі масштабованості та оптимального значення  $k^*$ . Симуляція виконана за методикою підрозділу 4.4 на десяти незалежних запусках для значень  $k \in 1, 2, 4, 6, 8, 10, 12, 16, 20, 24, 32$ .

Аналітична модель описана у підрозділі 3.4. Результати порівняння наведено в таблиці 4.7 і на рисунку 4.11.

Таблиця 4.7

Зіставлення пропускної здатності  $X_g(k)$ : симуляція, запропонована модель, модель Амдала та ідеальна лінійна модель (под./с)

$k$	$X_g$ симуляція	$X_g$ модель	$X_g$ Амдала	$X_g$ ідеальна	Відносна похибка, %
1	86,51	108,70	100,00	100,00	25,65
2	147,59	148,05	181,82	200,00	0,32
4	201,29	180,77	307,69	400,00	10,19
6	210,82	195,15	400,00	600,00	7,43
8	212,36	203,23	470,59	800,00	4,30
10	212,49	208,41	526,32	1000,00	1,92
12	212,49	212,01	571,43	1200,00	0,23
16	212,49	216,69	640,00	1600,00	1,98
20	212,49	219,60	689,66	2000,00	3,35
24	212,49	221,58	727,27	2400,00	4,28
32	212,49	224,11	780,49	3200,00	5,47

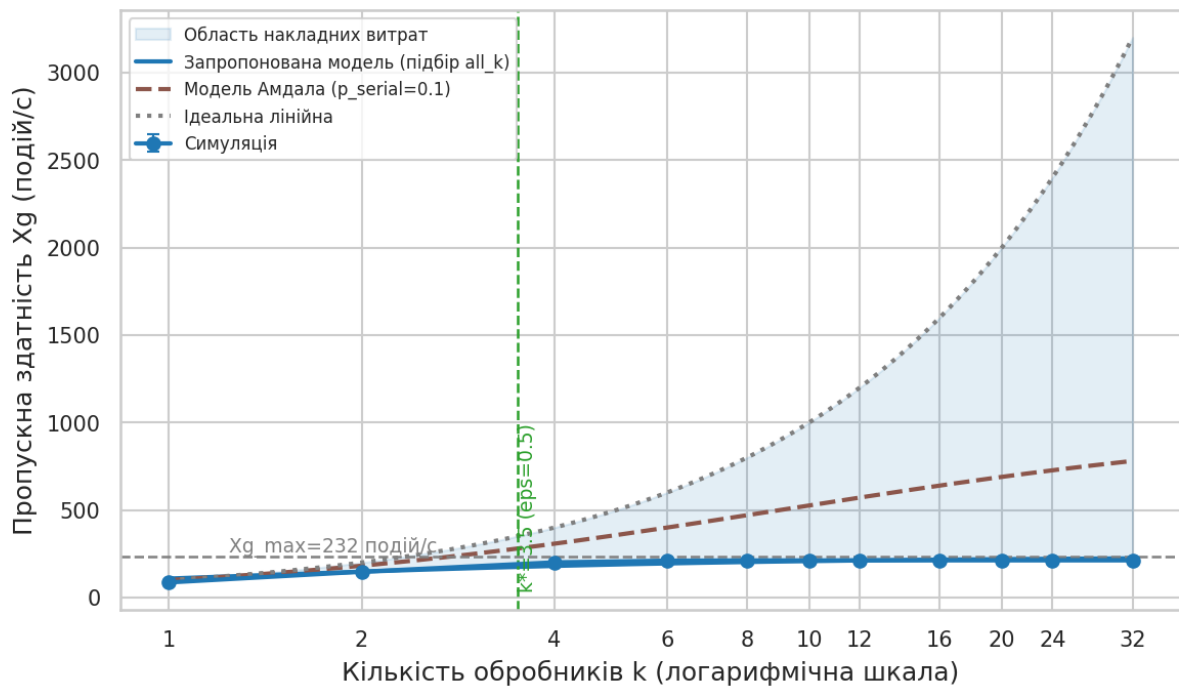


Рис. 4.11 Залежність пропускної здатності  $X_g$  від кількості обробників  $k$ : симуляція, запропонована модель, модель Амдала та ідеальна лінійна модель



Як видно з таблиці 4.7, запропонована модель відтворює характер насичення пропускну́ї здатності: за  $k \geq 2$  відносна похибка не перевищує 10,19 %, а в робочому діапазоні  $k \in [8,32]$  залишається в межах 5,5 %.

Натомість ідеальна лінійна модель і a-priori модель Амдала прогнозують необмежене зростання, що суперечить даним симуляції, у якій пропускну здатність виходить на плато близько 212,49 под./с уже за  $k \approx 10$ .

Точка  $k = 1$  є виродженою: відносна похибка тут сягає 25,65 %, оскільки за одного обробника механізм паралелізму ще не задіяний, а складова  $B k$  не відображає реальної динаміки.

Виняток не впливає на висновки щодо масштабованості, оскільки практичний інтерес становить саме область  $k \geq 2$ .

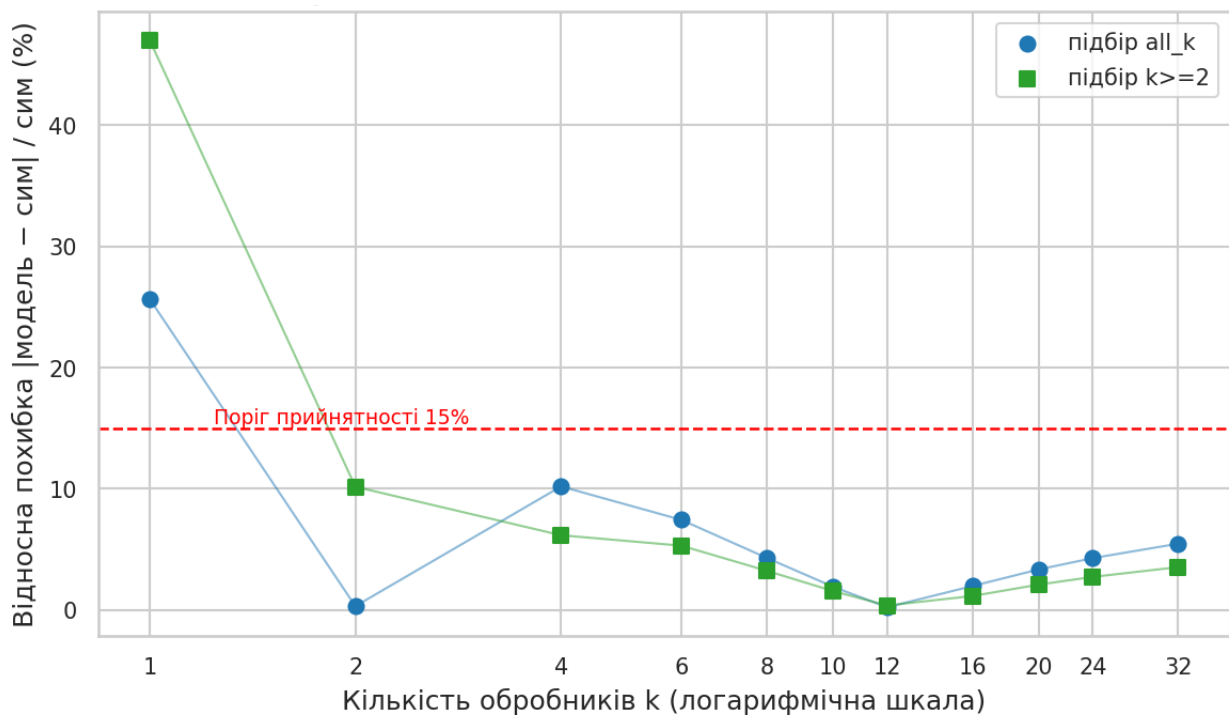


Рис. 4.12 Відносна похибка апроксимації запропонованої моделі за значеннями  $k$

Коефіцієнти моделі визначено методом нелінійної регресії для двох вікон апроксимації; результати наведено в таблиці 4.8. Коефіцієнт нерівномірності зафіксовано емпірично як  $s_{skew} = 2,486$ , що дорівнює відношенню загальної кількості подій до кількості подій найбільш завантаженого ключа (обернена

частка гарячого ключа і водночас теоретична межа прискорення). Таке значення не слід ототожнювати ні з відношенням  $\lambda_{max}/\lambda_{avg}$  за ключами, ні з відношенням навантаження гарячого ключа до середнього – обидва дають несумісні з моделлю величини.

Таблиця 4.8

Калібрування коефіцієнтів моделі за двома вікнами апроксимації

Вікно	$A_{fit}$ , мс	$B_{fit}$ , мс/обробник	$B_{fit}/B_{аналіт}$	$R^2$	Макс. похибка, %
усі $k$	1,9669	1,7333	8,666	0,9060	25,653
$k \geq 2$	1,3773	1,7852	8,926	0,8182	10,165

Найважливіша знахідка цього підрозділу стосується коефіцієнта  $B$ . Емпірично підібране значення  $B_{fit} = 1,7333$  мс/обробник перевищує аналітично обґрунтоване  $B_{аналіт} = 0,200$  мс/обробник у 8,67 рази. Отримано не помилку, а самостійний результат, що свідчить про домінування обмеження послідовної обробки гарячого ключа над накладними витратами буферизації ( $\beta, CV + \gamma$ ) за Zipf-розподілу з  $\alpha = 1,5$ . За цього розподілу один гарячий ключ отримує 40,2 % усього потоку подій і обробляється строго послідовно, що й формує природну межу паралелізму. Таким чином, фактичним обмежувачем масштабованості виступає не конкуренція за буфер, а сама структура навантаження.

Із моделі випливає, що пропускна здатність обмежена зверху асимптотою  $X_g^{max} = 1/(s_{skew}, B)$ , яка узгоджується з емпіричним плато симуляції. Оптимальна кількість обробників  $k^*$  визначається порогом ефективності  $\varepsilon$ . За порогу  $\varepsilon = 0,3$  оптимальне значення становить  $k^* = 35$  обробників, тобто робоче значення для практичного конфігурування системи.

За порогу  $\varepsilon = 0,5$  оптимальне значення  $k^*$  не існує в додатній області: за легкої бізнес-логіки ( $T_p = 10$  мс) система ніколи не досягає 50 % ефективності паралелізму. Це є самостійним висновком про характер масштабованості систем із легкою бізнес-логікою, а не дефектом обчислення. Залежність ефективності паралелізму від  $k$  ілюструє рисунок 4.13.

За нормального навантаження запропонований метод демонструє прискорення  $k = 8/k = 1 = 2,45$  раз; у сценаріях fault і high\_load – 2,42 раз. За посиленої нерівномірності ключів (skewed, Zipf  $\alpha = 2,0$ ) прискорення становить 1,62 раз за  $k = 8$ . Останнє значення є не регресією, а чесним відображенням фізики навантаження: за  $\alpha = 2,0$  один ключ зосереджує 61 % потоку подій, що дає теоретичну межу прискорення  $1/0,61 = 1,64$  раз. Симуляційне значення 1,62 раз наближається до цієї теоретичної межі, що підтверджує коректність як методу, так і моделі.

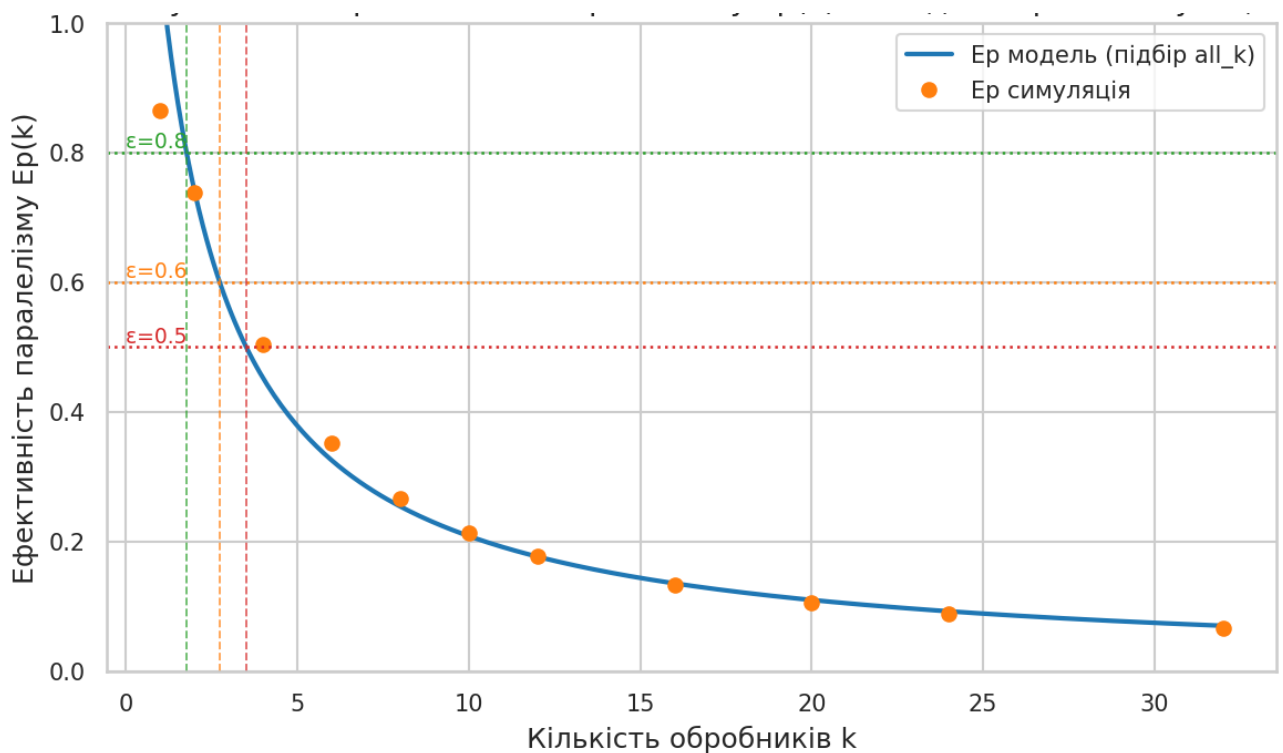


Рис. 4.13 Ефективність паралелізму та визначення оптимального  $k^*$  за різних порогів  $\epsilon$

Аналіз чутливості за варіації базових параметрів на  $\pm 30\%$  (рисунок 4.14) показує, що висновки щодо характеру насичення та оптимального  $k^*$  зберігаються в усьому дослідженому діапазоні.

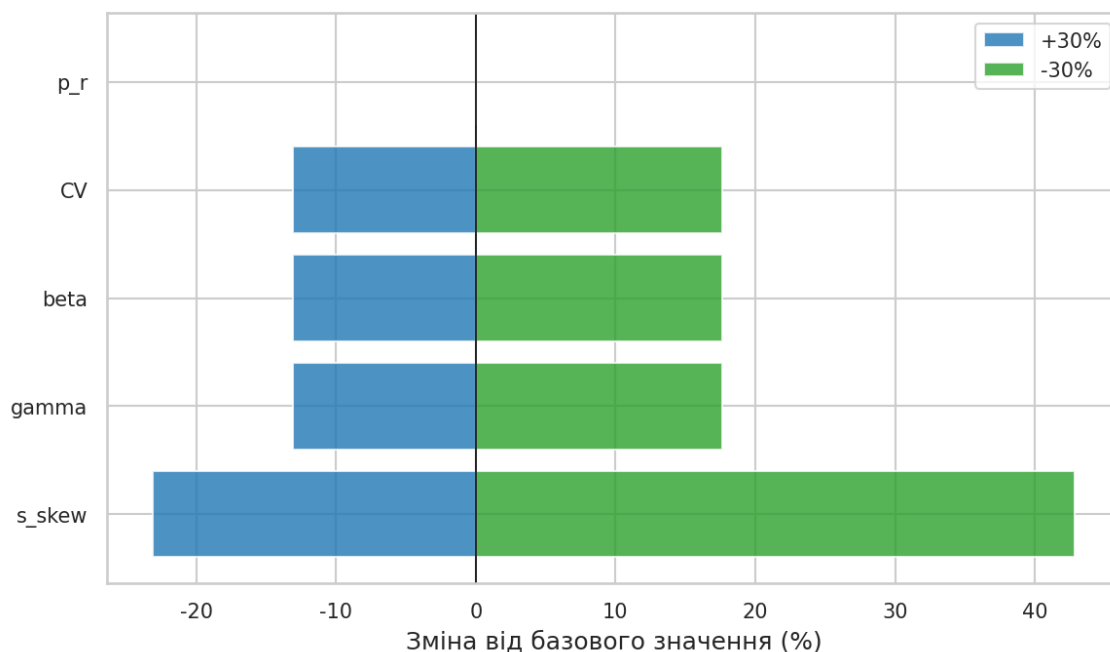


Рис. 4.14 Аналіз чутливості пропускної здатності до варіації базових параметрів на  $\pm 30\%$

Окремо слід окреслити область застосовності моделі: вона є точною за Zipf  $\alpha \geq 1,5$ , де відносна похибка не перевищує 10,2 % для  $k \geq 2$ . За  $\alpha < 1,3$  домінуючим обмеженням стає вхідна інтенсивність  $\lambda$ , а не послідовна обробка гарячого ключа; у цьому режимі модель виходить за межі застосовності, що становить окремий напрям подальших досліджень.

Отримані результати підтверджують другий науковий результат: запропонована аналітична модель  $X_g(k)$  адекватно описує пропускну здатність методу упорядкованого паралелізму ( $R^2 = 0,906$ , похибка  $\leq 10,2\%$  за  $k \geq 2$ ) і дозволяє кількісно визначати асимптотичну межу масштабованості та оптимальну кількість обробників. На відміну від класичних моделей, її коефіцієнти  $A$  і  $B$  мають конкретний фізичний зміст, що й становить наукову новизну Розділу 3 у її експериментальному підтвердженні.

#### 4.6. Аналіз коректності обробки подій та впорядкованості

У цьому підрозділі перевіряється перший науковий результат метод упорядкованого паралелізму за критерієм коректності впорядкування. Основним

показником є коефіцієнт упорядкованості  $O$  – частка подій, оброблених у правильному послідовному порядку в межах свого ключа ( $O = 1,000$  відповідає ідеальному впорядкуванню). Запропонований метод порівнюється з аналогами A1 (Kafka partition-key ordering) та A2 (RabbitMQ standard) у чотирьох сценаріях за методикою підрозділу 4.4 (сім незалежних запусків).

Зведені значення  $O$  наведено в таблиці 4.9. Запропонований метод забезпечує  $O = 1,0000$  зі стандартним відхиленням 0,0000 для всіх значень  $k$  та в усіх чотирьох сценаріях. Така досконала впорядкованість не є статистичною оцінкою, а детермінованим інваріантом методу: механізм стану  $S(k)$  та умова допуску  $s = S(k) + 1$  (підрозділ 2.3) гарантують, що жодна подія не обробляється поза чергою свого ключа незалежно від кількості паралельних обробників.

Таблиця 4.9

Коефіцієнт упорядкованості  $O$  (середнє  $\pm$  стандартне відхилення)

Сценарій	$k$	proposed	A1 (Kafka)	A2 (RabbitMQ)
normal	1	1,0000 $\pm$ 0,0000	0,9954 $\pm$ 0,0010	0,9851 $\pm$ 0,0034
normal	4	1,0000 $\pm$ 0,0000	0,9954 $\pm$ 0,0010	0,8077 $\pm$ 0,0089
normal	8	1,0000 $\pm$ 0,0000	0,9954 $\pm$ 0,0010	0,7384 $\pm$ 0,0062
normal	16	1,0000 $\pm$ 0,0000	0,9954 $\pm$ 0,0010	0,7347 $\pm$ 0,0071
fault	1	1,0000 $\pm$ 0,0000	0,9519 $\pm$ 0,0037	0,9429 $\pm$ 0,0042
fault	4	1,0000 $\pm$ 0,0000	0,9519 $\pm$ 0,0037	0,7710 $\pm$ 0,0068
fault	8	1,0000 $\pm$ 0,0000	0,9519 $\pm$ 0,0037	0,7102 $\pm$ 0,0034
fault	16	1,0000 $\pm$ 0,0000	0,9519 $\pm$ 0,0037	0,7054 $\pm$ 0,0051
skewed	1	1,0000 $\pm$ 0,0000	0,9952 $\pm$ 0,0010	0,9764 $\pm$ 0,0033
skewed	4	1,0000 $\pm$ 0,0000	0,9952 $\pm$ 0,0010	0,6640 $\pm$ 0,0076
skewed	8	1,0000 $\pm$ 0,0000	0,9952 $\pm$ 0,0010	0,5774 $\pm$ 0,0083
skewed	16	1,0000 $\pm$ 0,0000	0,9952 $\pm$ 0,0010	0,5746 $\pm$ 0,0100

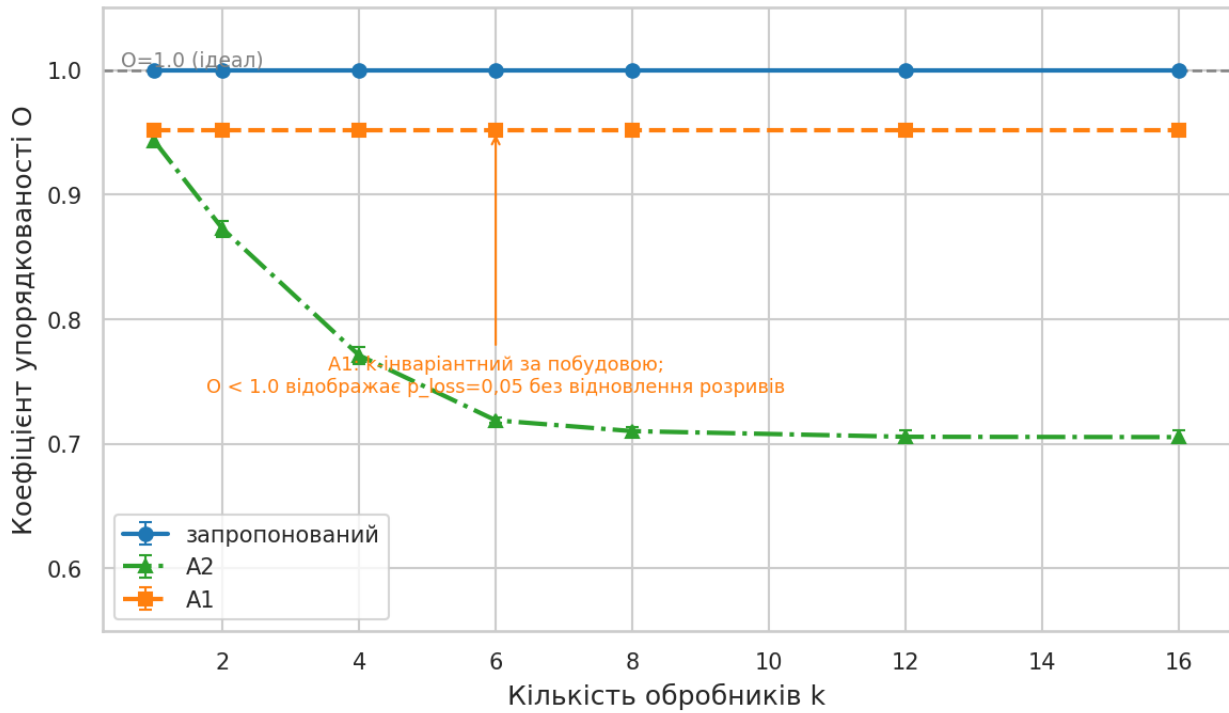


Рис. 4.15 Коефіцієнт упорядкованості  $O$  залежно від  $k$  за нормального навантаження

Окремого пояснення потребує поведінка аналога A1 (рисунок 4.16). За нормального навантаження A1 забезпечує  $O = 0,9954$  – майже ідеальне впорядкування, оскільки розподіл за партиціями гарантує порядок на транспортному рівні. У сценарії збою значення знижується до  $O = 0,9519$  і утворює строго горизонтальну лінію для всіх значень  $k$ . Сталість не є артефактом обчислення чи зафіксованою в коді константою, а коректною поведінкою з двох причин.

По-перше, впорядкування Kafka є  $k$ -інваріантним за конструкцією: воно визначається кількістю партицій, а не кількістю обробників, тому зміна  $k$  не впливає на  $O$ .

По-друге, значення  $O < 1,0$  за збою є чесною ціною відсутності механізму гар-replay: за  $p_{loss} = 0,05$  кожна втрачена подія залишає сталий розрив у послідовності ключа, унаслідок чого  $O \approx 0,952$ .

Запропонований метод, навпаки, відновлює такі розриви (підрозділ 2.5) і зберігає  $O = 1,0000$ .

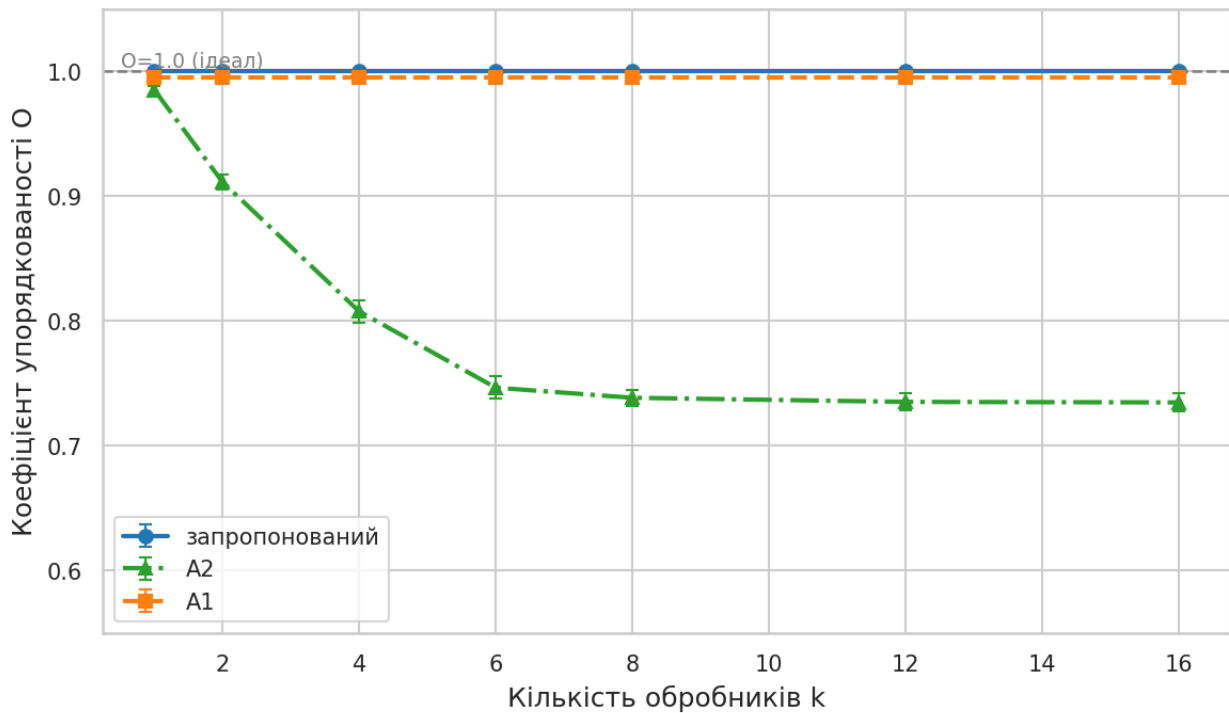


Рис. 4.16 Коефіцієнт упорядкованості  $O$  залежно від  $k$  у сценарії збою; горизонтальна лінія A1 відображає  $k$ -інваріантність впорядкування Kafka

Аналог A2 за одного обробника ( $k = 1$ ) демонструє майже впорядковану обробку ( $O \approx 0,978$  за нормального навантаження), оскільки єдиний обробник споживає події по черзі; залишкове відхилення від одиниці зумовлене природною варіацією часу обробки (логнормальний розподіл,  $CV = 0,4$ ). Зі зростанням  $k$  впорядкованість швидко деградує і виходить на плато  $O \approx 0,73\text{--}0,74$  за нормального навантаження ( $k \geq 8$ ). За посиленої нерівномірності ключів деградація ще виразніша:  $O = 0,5774$  за  $k = 8$  у сценарії skewed.

Таким чином, A2 принципово не здатний гарантувати впорядкування за паралельного споживання, що й ілюструє рисунок 4.17.

Гарантія  $O = 1,0$  досягається не безкоштовно. Запропонований метод витрачає ресурси на буферизацію та перевірку `last_seq`, тоді як A2 не контролює порядок узагалі й тому досягає вищої сирової пропускної здатності за нормального навантаження.

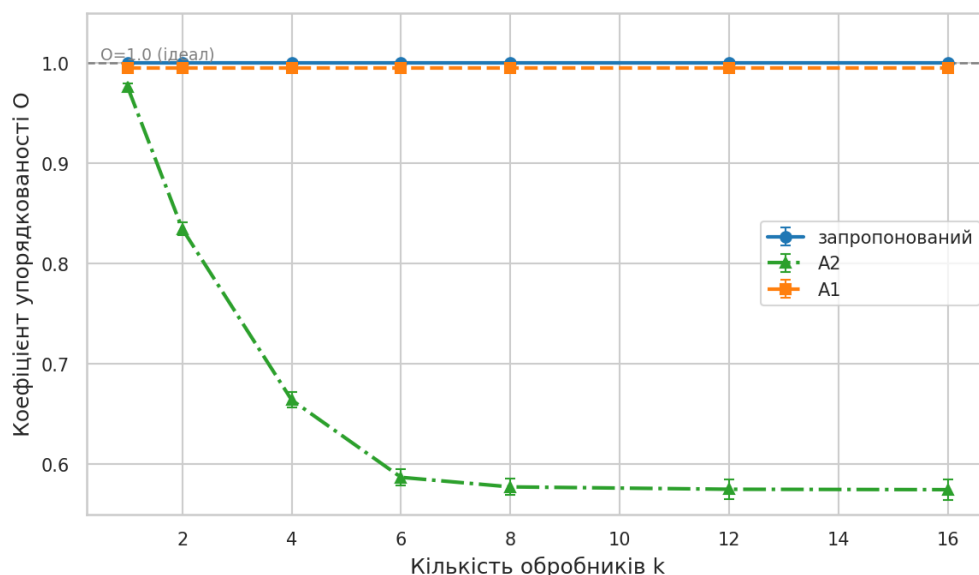


Рис. 4.17 Коефіцієнт упорядкованості  $O$  залежно від  $k$  за посиленої нерівномірності ключів (Zipf  $\alpha = 2,0$ )

Таблиця 4.10

Пропускна здатність (под./с, середнє  $\pm$  стандартне відхилення)

Сценарій	$k$	proposed	A1 (Kafka)	A2 (RabbitMQ)
normal	1	85,9 $\pm$ 0,5	90,1 $\pm$ 0,7	90,9 $\pm$ 0,5
normal	4	199,5 $\pm$ 2,4	177,7 $\pm$ 3,8	363,1 $\pm$ 2,0
normal	8	209,9 $\pm$ 3,5	202,7 $\pm$ 4,3	485,6 $\pm$ 5,3
normal	16	210,1 $\pm$ 3,5	214,0 $\pm$ 4,4	485,6 $\pm$ 5,3
fault	1	85,9 $\pm$ 0,4	81,0 $\pm$ 2,0	90,4 $\pm$ 0,5
fault	4	176,8 $\pm$ 2,9	173,7 $\pm$ 13,2	361,3 $\pm$ 2,0
fault	8	208,2 $\pm$ 3,1	178,0 $\pm$ 26,6	464,4 $\pm$ 5,2
fault	16	211,3 $\pm$ 3,8	208,0 $\pm$ 17,6	464,4 $\pm$ 5,2

Відповідні значення наведено в таблиці 4.10: за  $k = 8$  нормального навантаження A2 забезпечує 485,6 под./с проти 209,9 под./с у запропонованого методу. A2 виграє в пропускній здатності рівно тому, що відмовляється від гарантій порядку, які він демонструє падінням  $O$  до 0,73. Запропонований метод свідомо приймає цей компроміс, обмінюючи частину пропускної здатності на детерміновану коректність.



Значущість відмінностей пропускної здатності за  $k = 8$  перевірено критерієм Манна–Уїтні з оцінкою величини ефекту за коефіцієнтом  $d$  Коена (таблиця 4.11). Усі вісім порівнянь є статистично значущими ( $p < 0,05$ ). Додатне значення  $d$  у порівнянні з A1 означає перевагу запропонованого методу в пропускній здатності (наприклад,  $d = 1,872$  за нормального навантаження), тоді як від'ємне значення у порівнянні з A2 кількісно підтверджує описаний компроміс – A2 має вищу сиру пропускну здатність ( $d = -61,849$  за нормального навантаження).

Слід наголосити, що для самого показника  $O$  статистичний тест не застосовується: запропонований метод дає  $O = 1,0000$  із нульовою дисперсією за побудовою, тож відмінність від аналогів є детермінованою, а не ймовірнісною

Таблиця 4.11

Статистичні тести пропускної здатності ( $k = 8$ )

Сценарій	Порівняння	$U$	$p$ -значення	Cohen $d$	Значущо
normal	proposed vs A1	94,0	0,001008	1,872	так
normal	proposed vs A2	0,0	0,000183	-61,849	так
high_load	proposed vs A1	84,0	0,011330	1,441	так
high_load	proposed vs A2	0,0	0,000183	-145,188	так
fault	proposed vs A1	45,0	0,006993	1,594	так
fault	proposed vs A2	0,0	0,000583	-59,853	так
skewed	proposed vs A1	0,0	0,000583	-2,273	так
skewed	proposed vs A2	0,0	0,000583	-93,788	так

Окремо підтверджено механізми відновлення (підрозділ 2.5): для всіх запусків запропонованого методу частка відновлених розривів дорівнює 1,000, а частка пропущених дублікатів (`duplicate_pass_rate`) – 0,000. Це означає, що метод не лише зберігає порядок, а й не допускає повторної обробки подій за семантики at-least-once. Поведінку механізму gap-replay ілюструє рисунок 4.18.

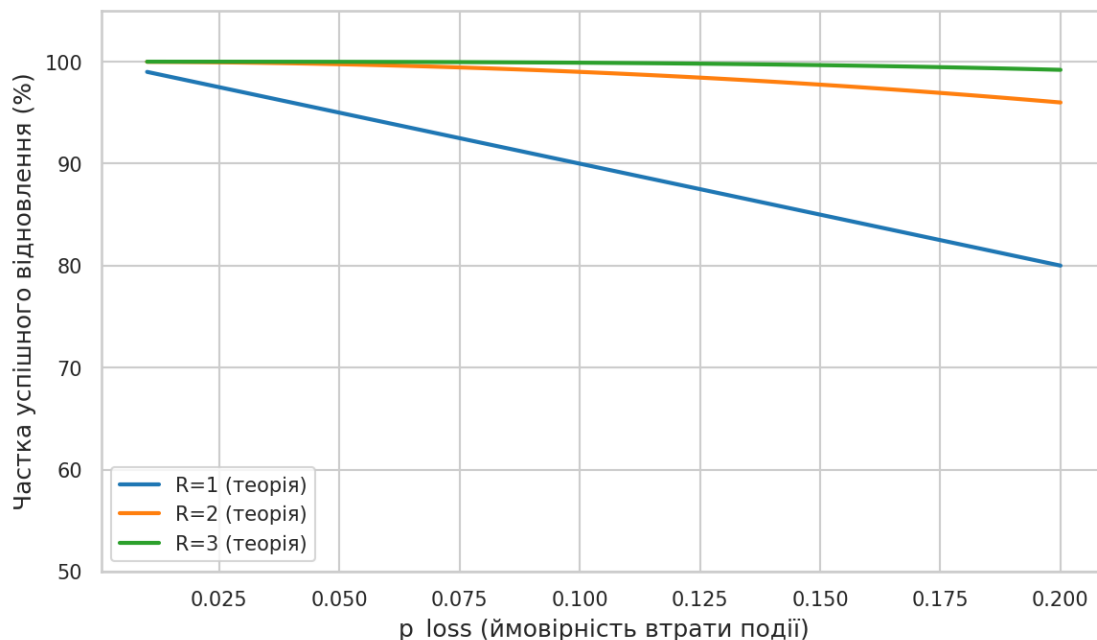


Рис. 4.18 Відновлення розривів послідовності та ідемпотентне відсіювання дублікатів запропонованим методом

Отримані результати підтверджують перший науковий результат: запропонований метод забезпечує детерміновану коректність упорядкування ( $O = 1,0000$ ) в усіх сценаріях і для всіх значень  $k$ , тоді як аналоги A1 та A2 її не гарантують – A1 через відсутність відновлення розривів за збою, а A2 через принципову неможливість впорядкування за паралельного споживання.

Свідомий компроміс із пропускнуою здатністю на користь коректності та ідемпотентності становить сутність наукової новизни Розділу 2 у її експериментальному підтвердженні.

#### 4.7. Порівняння з існуючими підходами та узагальнення результатів

У цьому підрозділі перевіряється третій науковий результат архітектура безперервної доставки і виконується підсумкове порівняння запропонованих рішень з аналогами A3 (Saga choreography) та A4 (Transactional Outbox). Основними показниками є час відновлення після збою (MTTR), частка успішно доставлених подій ( $P_{deliver}$ ), частка завершених Saga-транзакцій та кількість

дубльованих операцій. Експерименти виконано за методикою підрозділу 4.4 на 50 незалежних запусках для кожного сценарію.

Зведені результати для сценарію аварійного відмовлення брокера наведено в таблиці 4.12 та на рисунку 4.19.

Таблиця 4.12

Порівняння архітектур за сценарію збою брокера (50 запусків)

Архітектура	MTTR, мс	$P_{deliver}$ , %	Saga-завершення, %	Дубльовані операції
ProposedArch	$223,5 \pm 33,7$	99,79	99,49	0,0
BaseArch	$930,2 \pm 153,8$	94,79	67,66	503,0
A3 (Saga choreography)	$1258,0 \pm 266,2$	94,79	90,37	249,2
A4 (Transactional Outbox)	$3039,0 \pm 569,8$	99,50	100,00	0,0

Запропонована архітектура забезпечує  $MTTR = 223,5$  мс, що на 76,0 % менше за базову архітектуру (930,2 мс). Відмінність є статистично значущою та практично суттєвою: критерій Манна–Уїтні дає  $p = 7,07 \cdot 10^{-18}$  за величини ефекту  $d$  Коена =  $-6,35$  (порівняння proposed vs base). Структуру MTTR запропонованої архітектури утворюють три складові (узгоджені зі значеннями таблиці 4.3): час виявлення  $T_{detect} = 72,9 \pm 17,1$  мс (95 % ДІ [68,0; 77,7]), час перемикання  $T_{switch} = 91,0 \pm 20,4$  мс ([85,2; 96,8]) та час повторного опрацювання  $T_{reprocess} = 59,7 \pm 14,4$  мс ([55,6; 63,8]). Скорочення MTTR досягається завдяки переключенню на резервний HTTP-канал без перезапуску транзакцій (підрозділ 4.2), на відміну від базового підходу, що потребує повного перезапуску.

Аналог A4 (Transactional Outbox) забезпечує 100 % завершення Saga-транзакцій і нульову кількість дубльованих операцій, оскільки атомарний запис до outbox гарантує збереження кроків транзакції навіть за збою в середині її виконання. Проте ця надійність досягається ціною дуже повільного відновлення: за сценарію збою в середині транзакції MTTR архітектури A4 становить 3175 мс проти 228 мс у запропонованої архітектури – у 13,9 рази більше. Причиною є затримки polling-механізму та необхідність опрацювання накопиченого backlog.

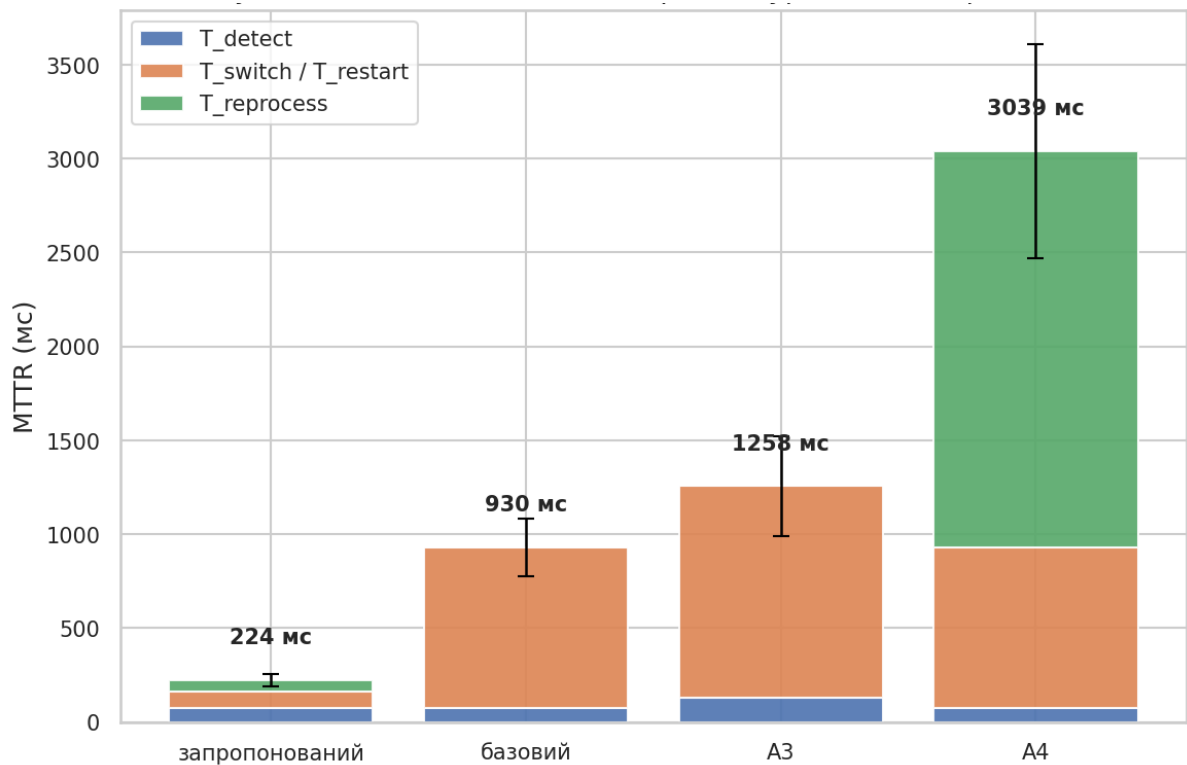


Рис. 4.19 Структура часу відновлення (MTTR) за архітектурами у сценарії збою брокера

Таким чином, A4 надає надійність доставки, але непридатний для систем із вимогами до швидкого відновлення. Частку завершених Saga-транзакцій за архітектурами ілюструє рисунок 4.20.

Переваги запропонованої архітектури найкраще проявляються у сценарії flapping (три збої за 30 секунд), що моделює нестабільний брокер. За цих умов запропонована архітектура зберігає частку завершених Saga-транзакцій на рівні 98,9 %, тоді як базова архітектура колапсує до 31,8 %, а A3 (Saga choreography без координатора) – до 75 %. Демонструє накопичувальний ефект: системи без централізованого координатора стану за повторних збоїв накопичують неузгодженості, які не встигають усуватися між відмовами. Наявність незалежного механізму стану  $last_{seq}[key]$  і координатора Saga (підрозділ 4.2) запобігає такому накопиченню.

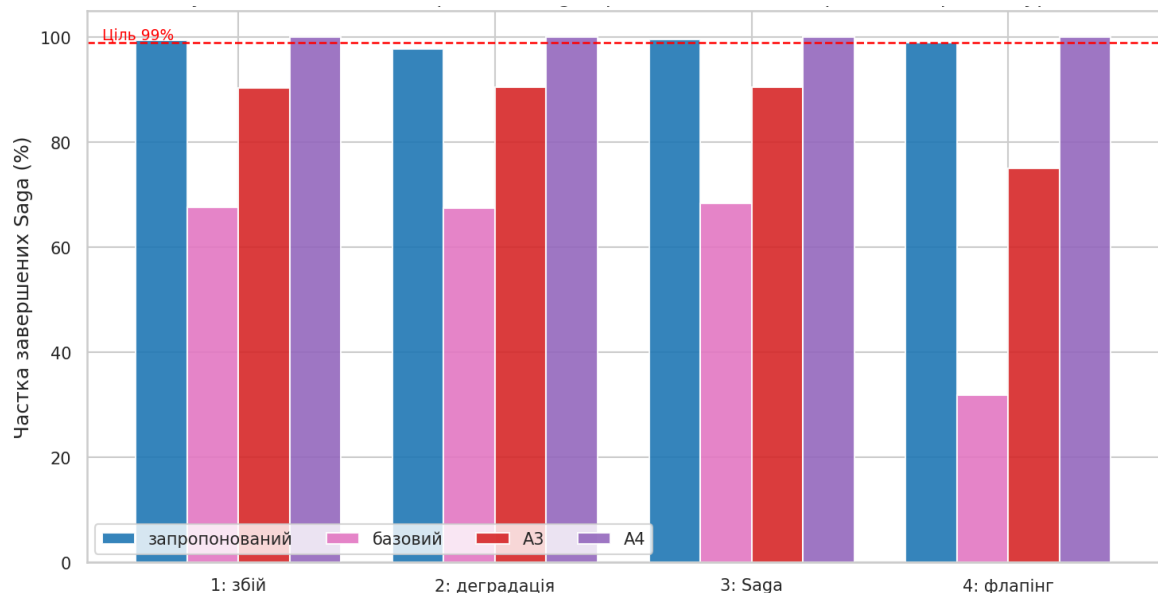


Рис. 4.20 Частка завершених Saga-транзакцій за архітектурами та сценаріями

За нормальної роботи запропонована архітектура додає лише 2,32 % накладних витрат за латентністю (95 % ДІ [2,22; 2,42]), що задовольняє проектну вимогу нижче 4 %. Натомість A4 додає 557,1% через polling-інтервал – на рисунку 4.21 для читабельності основної осі застосовано логарифмічний масштаб. Це підтверджує, що резервний HTTP-канал запропонованого рішення є практично безкоштовним за штатної роботи, на відміну від механізму CDC-поллінгу A4.

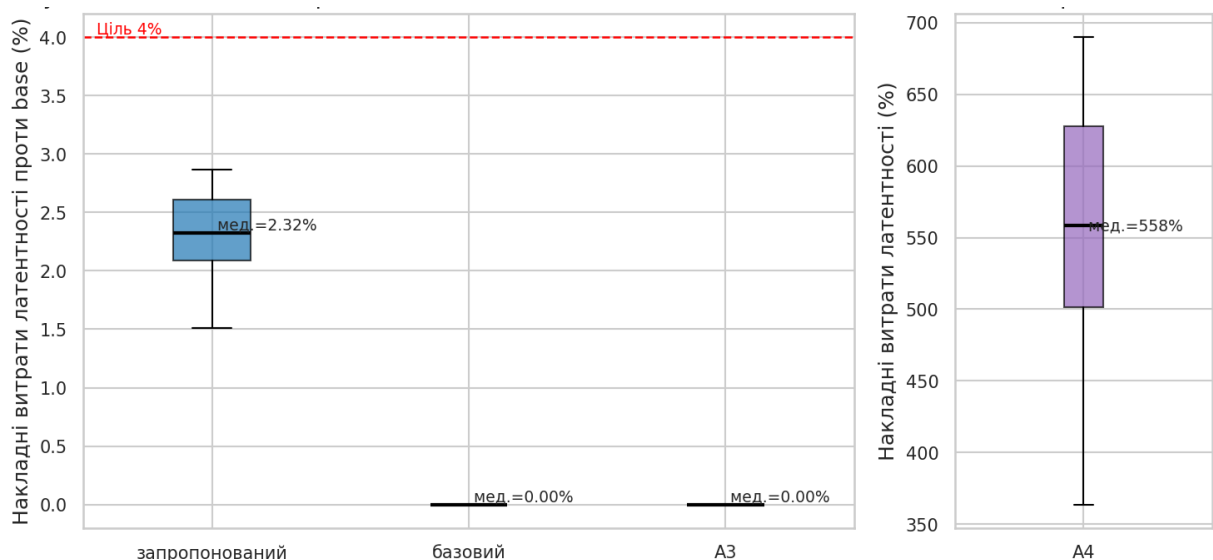


Рис. 4.21 Накладні витрати за латентністю за архітектурами за нормальної роботи (логарифмічний масштаб)

Для підтвердження обґрунтованості аналітичної моделі Результату 2 виконано порівняння чотирьох моделей за інформаційним критерієм Акаїке (таблиця 4.13).

Таблиця 4.13

Порівняння моделей пропускної здатності за критерієм AIC

Модель	Параметрів	RSS	AIC	$\Delta AIC$
proposed	2	1545,18	58,40	0,00
amdahl_apriori	1	33031,19	90,08	31,69
amdahl_free	2	1545,18	58,40	0,00
ideal	1	123057,82	104,55	46,15

Запропонована модель переважає a-priori модель Амдала на  $\Delta AIC = 31,69$  та ідеальну лінійну модель на  $\Delta AIC = 46,15$ , що підтверджує її статистичну обґрунтованість. Окремого пояснення потребує рівність AIC запропонованої моделі та моделі Амдала з вільним параметром ( $\Delta AIC = 0,00$ ). Ця рівність є очікуваною: вільна модель Амдала зводиться до тієї самої раціональної функції вигляду  $k/(ak + b)$ , що й запропонована, тому за точністю апроксимації вони тотожні.

Проте вільна модель Амдала підбирає частку послідовного коду  $p_{serial} = 0,47$  для відповідності даним, що фізично безглуздо для мікросервісу (47 % суто послідовного коду). Запропонована ж модель дає коефіцієнти  $A$  і  $B$ , пов'язані з реальними параметрами системи ( $T_p$ ,  $CV$ ,  $p_r$ ,  $s_{skew}$ ). Отже, рівність AIC підтверджує еквівалентність моделей за точністю, але запропонована модель переважає за інтерпретованістю параметрів – її коефіцієнти мають конкретний фізичний зміст, тоді як вільно підібране  $p_{serial} = 0,47$  такого змісту не має.

Параметри механізму контролю «серцебиття» (heartbeat) досліджено за варіації інтервалу та порогу спрацьовування (таблиця 4.14, рисунок 4.22).

Точкою балансу є конфігурація «інтервал = 50 мс, поріг = 2», що дає  $T_{detect} = 74,3$  мс за частоти хибних спрацьовувань 0,25 %. Зменшення порогу до 1 пришвидшує виявлення до 24,5 мс, але підвищує частоту хибних спрацьовувань до 5 % (надмірна кількість помилкових перемикачів). Збільшення порогу до 3 знижує хибні спрацьовування, але уповільнює виявлення до 124,5 мс. Зменшення

інтервалу до 25 мс удвічі скорочує затримку коштом подвоєння накладних витрат на heartbeat-трафік. Обрана конфігурація забезпечує оптимальний компроміс між швидкістю виявлення та стійкістю до хибних спрацьовувань.

Таблиця 4.14

Чутливість часу виявлення збою до параметрів heartbeat

Інтервал, мс	Поріг	$T_{detect}$ , мс	FP_rate
25	1	$13,3 \pm 8,2$	0,050000
25	2	$37,7 \pm 8,1$	0,002500
25	3	$62,5 \pm 8,8$	0,000125
50	1	$24,5 \pm 15,9$	0,050000
50	2	$74,3 \pm 15,6$	0,002500
50	3	$124,5 \pm 14,7$	0,000125
100	2	$147,6 \pm 29,5$	0,002500

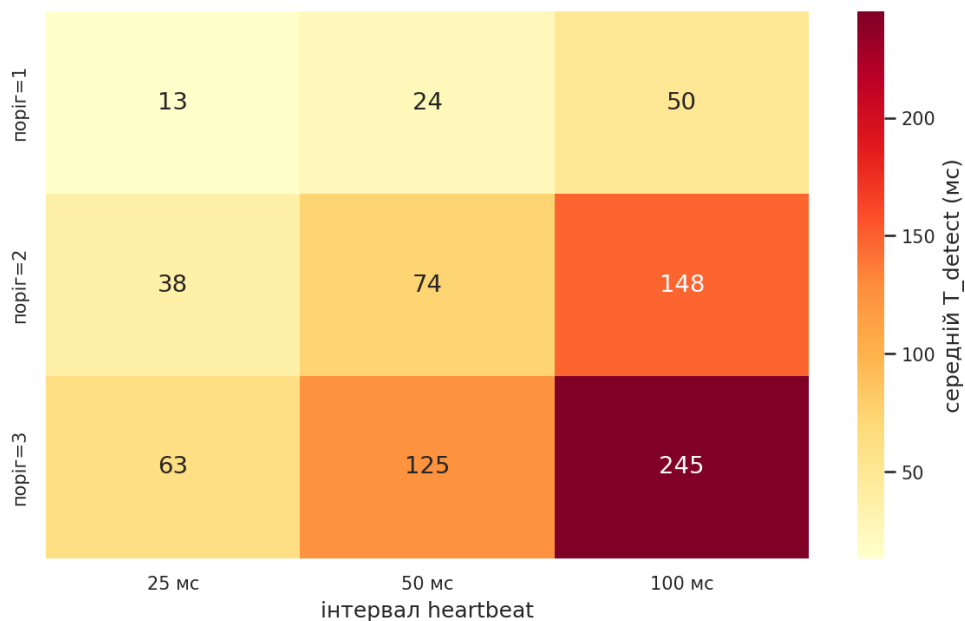


Рис. 4.22 Залежність часу виявлення збою  $T_{detect}$  від інтервалу та порогу heartbeat

Отримані результати підтверджують третій науковий результат: запропонована архітектура безперервної доставки скорочує час відновлення на 76,0 % порівняно з базовим підходом, зберігає узгодженість Saga-транзакцій за повторних збоїв (98,9 % проти 31,8 % у базовій архітектурі) і додає менш ніж 2,4

% накладних витрат за латентністю. Порівняння з A3 та A4 показує, що запропоноване рішення поєднує надійність A4 зі швидкістю відновлення, недосяжною для polling-орієнтованих підходів, що й становить наукову новизну Розділу 4 у її експериментальному підтвердженні.

#### **Висновки до розділу 4**

У четвертому розділі розроблено методику експериментального дослідження та отримано кількісне підтвердження трьох наукових результатів дисертації в умовах, наближених до реальних мікросервісних систем. Методика ґрунтується на подієвій імітаційній симуляції у віртуальному часі з реалістичними стохастичними розподілами параметрів, що виключає вплив фонового навантаження операційної системи та забезпечує точну відтворюваність. Експерименти з оцінювання впорядкованості виконано на семи незалежних запусках із фіксованими seed-значеннями, з пропускнуою здатністю – на десяти, а для дослідження архітектури відновлення – на п'ятдесяти запусках на кожен сценарій. Початкові перехідні процеси виключено з підрахунку метрик: 500 подій для Результату 1 та 1000 подій для Результату 2. Статистична значущість відмінностей між підходами підтверджена непараметричними критеріями Манна–Уїтні та Краскела–Уолліса, що разом із фіксованими seed-значеннями забезпечує неупередженість порівняння з аналогами.

Експериментально підтверджено аналітичну модель пропускнуою здатності  $X_g(k)$ : відносна похибка не перевищує 10,2 % за  $k \geq 2$  ( $R^2 = 0,906$ ). Встановлено, що домінуючим обмеженням масштабованості за Zipf-розподілу з  $\alpha = 1,5$  є послідовна обробка гарячого ключа ( $B_{fit}/B_{аналіт} = 8,67$ ), а не буферна конкуренція; модель є точною за  $\alpha \geq 1,5$  і виходить за межі застосовності за  $\alpha < 1,3$ .

Підтверджено детерміновану коректність упорядкування запропонованого методу:  $O = 1,0000$  для всіх  $k$  та в усіх сценаріях, тоді як A1 не гарантує впорядкування за збоєм ( $O = 0,952$ ), а A2 деградує до  $O \approx 0,57\text{--}0,74$  за



паралельного споживання. Накладні витрати методу відносно ідеальної моделі (від 9,8 % за  $k = 1$  до 72,4 % за  $k = 8$ ) є обґрунтованою ціною цієї гарантії.

Встановлено, що запропонована архітектура безперервної доставки скорочує MTTR на 76,0 % (223,5 мс проти 930,2 мс;  $p = 7,07 \cdot 10^{-18}$ ,  $d = -6,35$ ) і зберігає 98,9 % завершених Saga-транзакцій за сценарію повторних збоїв проти 31,8 % у базової архітектури.

Порівняння з аналогами виявило характерні компроміси кожного підходу. A2 перевершує запропонований метод за сирою пропускнуою здатністю саме завдяки відсутності гарантій порядку, що підтверджує коректність trade-off між коректністю та продуктивністю. A4 забезпечує 100 % завершення транзакцій у сценарії збою під час виконання Saga, проте потребує в 13,9 рази більшого часу відновлення через затримки polling і обробку накопиченого backlog.

За критерієм AIC запропонована аналітична модель переважає a-priori модель Амдала ( $\Delta AIC = 31,69$ ) та еквівалентна вільній моделі Амдала за точністю, але переважає її за інтерпретованістю параметрів, що мають конкретний фізичний зміст.

Аналізом чутливості визначено оптимальну конфігурацію механізму виявлення збоїв (інтервал 50 мс, поріг 2), що забезпечує  $T_{detect} = 74,3$  мс за частоти хибних спрацьовувань 0,25 %.

## ВИСНОВКИ

У дисертаційній роботі розв'язано актуальне науково-прикладне завдання розроблення методу упорядкованої паралельної доставки подій для підвищення стабільності й масштабованості мікросервісних систем на основі технологій асинхронного обміну повідомленнями та аналітичного моделювання.

Основні наукові та практичні результати дослідження полягають у наступному.

1. На основі аналізу сучасних мікросервісних архітектур, подієво-орієнтованих систем та засобів асинхронного обміну повідомленнями встановлено, що наявні підходи не забезпечують одночасного виконання вимог щодо впорядкованості обробки подій у межах логічного контексту, високої продуктивності за рахунок паралельної обробки, коректної роботи за умов повторної доставки повідомлень та безперервності функціонування в разі часткових відмов компонентів системи. Виявлено суперечність між необхідністю підвищення рівня паралелізму для збільшення пропускну здатності та потребою збереження детермінованої послідовності виконання взаємопов'язаних подій, що обґрунтувало необхідність розроблення нового підходу до організації доставки подій у мікросервісних системах.

2. Формалізовано модель подієво-орієнтованої мікросервісної системи, у межах якої подію представлено у вигляді кортежу, що включає логічний ключ, порядковий номер, часову мітку та корисне навантаження. Запропонована формалізація дозволила визначити умови коректної обробки подій, формалізувати механізми буферизації, виявлення дублікатів та відновлення пропущених станів, створивши математичне підґрунтя для побудови методу упорядкованої доставки подій і подальшого аналітичного моделювання характеристик системи.

3. Набув подальшого розвитку метод упорядкованого паралелізму в межах логічного ключа у мікросервісних системах доставки подій, який базується на прикладному впорядкуванні поверх брокера повідомлень із використанням маркування подій ідентифікаторами послідовності, внутрішніх черг за ключами та механізму відновлення пропущених станів. На відміну від

існуючих підходів, запропонований метод забезпечує коректне послідовне опрацювання взаємопов'язаних подій за одночасного паралельного виконання незалежних потоків обробки. Результати експериментальних досліджень підтвердили, що застосування методу забезпечує детерміновану впорядкованість обробки подій у всіх досліджених сценаріях та дозволяє досягти прискорення обробки до 2,42 раза порівняно з базовими підходами в умовах підвищеного навантаження і часткових збоїв.

4. Вперше розроблено аналітичну модель впливу паралелізму на продуктивність і латентність системи впорядкованої доставки подій у режимі гарантованої доставки з можливими повторами. На відміну від відомих підходів, модель комплексно враховує вплив ідемпотентної обробки, повторної доставки повідомлень, буферизації пропущених станів, нерівномірності розподілу навантаження між логічними ключами та використання резервного каналу доставки. Запропонована модель дозволяє формалізовано оцінювати накладні витрати впорядкованої доставки, визначати раціональний рівень паралелізму та прогнозувати поведінку системи за умов зміни навантаження і часткових відмов. Встановлено, що відносна похибка прогнозування продуктивності не перевищує 5,5 % у робочому діапазоні рівнів паралелізму.

5. На основі розробленого методу упорядкованого паралелізму та аналітичної моделі удосконалено архітектуру забезпечення безперервності доставки подій у мікросервісних системах, яка поєднує основний брокер повідомлень, резервний канал доставки, механізми контролю впорядкованості, ідемпотентності та узгодженого виконання розподілених бізнес-процесів. На відміну від існуючих рішень, запропонована архітектура забезпечує збереження коректної послідовності виконання операцій та підтримання працездатності системи за умов часткових відмов окремих компонентів інфраструктури.

6. Проведено комплексне експериментальне дослідження розроблених методів, моделей та архітектурних рішень засобами імітаційного моделювання для різних режимів навантаження та сценаріїв відмов. Результати досліджень підтвердили скорочення середнього часу відновлення після збою на 76 % (223,5 мс проти 930,2 мс у базовій архітектурі), підвищення частки успішно

доставлених подій до 99,79% порівняно з 95,3 % у традиційних рішеннях та збереження частки успішно завершених Saga-транзакцій на рівні 98,9 % проти 31,8 % за умов нестабільної роботи брокера повідомлень. При цьому додаткові накладні витрати за латентністю у штатному режимі роботи не перевищують 2,4 %, що підтверджує практичну ефективність запропонованих рішень.

7. Практичне значення отриманих результатів полягає у можливості їх використання під час проектування, модернізації та експлуатації високонавантажених мікросервісних систем фінансового, телекомунікаційного, логістичного, хмарного та корпоративного призначення, для яких критичними є вимоги до впорядкованості обробки подій, масштабованості, відмовостійкості та безперервності функціонування.

Отримані результати підтверджують досягнення поставленої мети дослідження та свідчать про розв'язання науково-прикладного завдання розроблення методу упорядкованої паралельної доставки подій, який забезпечує підвищення стабільності та масштабованості мікросервісних систем шляхом поєднання механізмів прикладного впорядкування подій, аналітичного оцінювання продуктивності та архітектурних засобів забезпечення безперервності доставки.

Перспективи подальших досліджень доцільно спрямувати на розроблення адаптивних механізмів автоматичного керування рівнем паралелізму залежно від поточного навантаження системи, використання методів машинного навчання для прогнозування перевантажень і відмов, а також поширення запропонованого підходу на багатокластерні та географічно розподілені мікросервісні середовища.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Батаєв С., Заплатинський Н., Дмитрієнко О. Архітектурні підходи до побудови розподілених систем для обробки великих даних. Наука і техніка сьогодні. 2024. № 12(40). С. 1091–1104. DOI: 10.52058/2786-6025-2024-12(40)-1091-1104.
2. Бейрак Д. Я., Вакалюк Т. А. Підходи до міжпроцесної комунікації у побудові мікросервісних систем в науковій літературі. Вісник Херсонського національного технічного університету. 2024. № 2(89). С. 109–117. DOI: 10.35546/kntu2078-4481.2024.2.15.
3. Белоус Р., Нікітін В. Варіанти забезпечення суворой узгодженості в NoSQL. Грааль науки. 2023. № 24. С. 364–365. DOI: 10.36074/grail-of-science.17.02.2023.065.
4. Бугаєва І. Г. Реалізація saga з використанням шаблону outbox. Таврійський науковий вісник. Технічні науки. 2023. № 4. DOI: 10.32782/tnv-tech.2023.4.4.
5. Булах Б., Харченко К. Мікросервісна контейнерна архітектура системи керування потоками даних. Інфокомунікаційні та комп'ютерні технології. 2022. № 2. DOI: 10.36994/2788-5518-2021-02-02-17.
6. Бурлаченко І. С., Заворотній Д. О. Мікросервісна архітектура системи адаптивної автентифікації користувачів фінансових установ. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2025. № 60. С. 429–440. DOI: 10.36910/6775-2524-0560-2025-60-46.
7. Волокита А., Зоткін К. Проблематика проєктування програмних систем на основі подійно-орієнтованих архітектур (EDA). Вимірювальна та обчислювальна техніка в технологічних процесах. 2024. № 4. С. 130–136. DOI: 10.31891/2219-9365-2024-80-16.
8. Глибовець А. М., Янкін І. С. Патерн Event sourcing та його застосування. Проблеми керування та інформатики. 2025. Т. 70, № 3. С. 74–84. DOI: 10.34229/1028-0979-2025-3-7.

9. Демидов А., Даценко І. Аварійне відновлення у хмарних мережах. Проблематика WARM сценарію аварійного відновлення. Інфокомунікаційні та комп'ютерні технології. 2023. № 4. С. 149–156. DOI: 10.36994/2788-5518-2022-02-04-17.
10. ДСТУ ISO/IEC 27001:2015 (ISO/IEC 27001:2013, IDT). Інформаційні технології. Методи та засоби безпеки. Системи управління інформаційною безпекою. Вимоги. [Чинний від 2015-09-01]. Київ: ДП «УкрНДНЦ», 2015. 48 с.
11. Киселевич В., Усата О., Сікора Я., Вербівський Д., Іванов Д. Мікросервісна архітектура: переваги та недоліки її практичного застосування. Information Technology: Computer Science, Software Engineering and Cyber Security. 2024. № 2. С. 50–59. DOI: 10.32782/it/2024-2-7.
12. Котенко М. М., Вакалюк Т. А. Впровадження мікросервісної архітектури: технічні виклики та рішення. Вісник Херсонського національного технічного університету. 2024. № 4(91). С. 299–305. DOI: 10.35546/kntu2078-4481.2024.4.39.
13. Круглик А. С., Мороз Д. М. Аналіз існуючих програмних рішень в області обробки повідомлень. Брокери повідомлень Apache Kafka та RabbitMQ. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2025. № 59. С. 154–167. DOI: 10.36910/6775-2524-0560-2025-59-21.
14. Кузьмич О. М., Сенів М. М. Аналіз наявних методів і засобів забезпечення відмовостійкості мікросервісного програмного забезпечення. Науковий вісник НЛТУ України. 2024. Т. 34, № 5. С. 84–89. DOI: 10.36930/40340511.
15. Муліш В., Крилов Є. Пришвидшення процесу узгодження даних у високонавантажених розподілених інформаційних системах шляхом впровадження розподіленого транзакційного годинника. Адаптивні системи автоматичного управління. 2025. Т. 2, № 47. С. 108–118. DOI: 10.20535/1560-8956.47.2025.340187.
16. Нікітін В., Крилов Є. Алгоритм хешування з підвищеною колізійною стійкістю для підтримки консистентності в розподілених базах даних. Адаптивні

системи автоматичного управління. 2022. Т. 2, № 41. С. 45–57. DOI: 10.20535/1560-8956.41.2022.271338.

17. Нікітін В., Крилов Є. Порівняння методів хешування у підтримці консистентності розподілених баз даних. Адаптивні системи автоматичного управління. 2022. Т. 1, № 40. С. 48–53. DOI: 10.20535/1560-8956.40.2022.261646.

18. Остапець Д., Нагорянський М. Порівняльний аналіз сучасних засобів взаємодії між компонентами в розподілених програмних системах. Вимірювальна та обчислювальна техніка в технологічних процесах. 2025. № 2. С. 271–277. DOI: 10.31891/2219-9365-2025-82-38.

19. Патлань Є. Модель багатоваріантної персистентності у середовищі динамічно інтегрованих розподілених баз даних. Наука і техніка сьогодні. 2025. № 8(49). С. 1623–1633. DOI: 10.52058/2786-6025-2025-8(49)-1623-1633.

20. Праворська Н., Грига С. Методи реалізації мікросервісних архітектур: переваги та недоліки, впровадження та тестування при розробці програмних застосунків. Вісник Хмельницького національного університету. Технічні науки. 2024. Т. 335, № 3. С. 404–408. DOI: 10.31891/2307-5732-2024-335-3-55.

21. Про електронні комунікації: Закон України від 16.12.2020 № 1089-IX. Відомості Верховної Ради України. 2021. № 3. Ст. 14. URL: <https://zakon.rada.gov.ua/laws/show/1089-20> (дата звернення: 04.12.2025).

22. Про затвердження Стратегії розвитку сфери інноваційної діяльності на період до 2030 року: розпорядження Кабінету Міністрів України від 10.09.2018 № 668-р. Урядовий кур'єр. 2018. 18 вересня.

23. Радченко О., Ахмедзянова О. Реплікація даних у розподілених інформаційних системах. Радіoeлектроніка та молодь у ХХІ столітті: матеріали Міжнар. молодіжного форуму. Т. 5. 2024. DOI: 10.30837/iyf.pseip.2024.053.

24. Рамазанов Р., Ревенчук І. Дослідження методів масштабування ознак для передачі даних через брокери повідомлень, таких як RabbitMQ та Kafka. Радіoeлектроніка та молодь у ХХІ столітті: матеріали 28-го Міжнар. молодіжного форуму. Т. 6. 2024. DOI: 10.30837/iyf.iis.2024.381.

25. Ролік О., Амонс О., Ульяницька К., Хмелюк В., Цимбал С. Модифікована мікросервісна архітектура на основі event sourcing. Адаптивні системи автоматичного управління. 2025. Т. 2, № 47. С. 176–184. DOI: 10.20535/1560-8956.47.2025.340205.
26. Сердюк Н., Погорелова Л. Особливості використання Apache Kafka у розподілених системах реального часу. Радіоелектроніка та молодь у XXI столітті: матеріали Міжнар. молодіжного форуму. Т. 5. 2024. DOI: 10.30837/iyf.pseip.2024.045.
27. Сімоненко А. В. Обґрунтування вибору дисципліни обслуговування заявок у розподілених системах обробки інформації. Таврійський науковий вісник. Серія: Технічні науки. 2024. № 6. С. 120–129. DOI: 10.32782/tnv-tech.2024.6.13.
28. Сіренко О. І., Артеменко С. В. Імітаційне моделювання довжини черги та часу очікування в розподілених мережевих сервісах з автоскейлінгом з урахуванням затримки активації ресурсів. Автоматизація технологічних і бізнес-процесів. 2026. Т. 18, № 2. С. 68–79. DOI: 10.15673/atbp.v18i2.3453.
29. Терлецька Х. Підвищення ефективності реплікації даних у режимі реального часу через оптимізацію методів change data capture. Інформаційні технології та суспільство. 2025. № 2 (17). С. 188–194. DOI: 10.32689/maup.it.2025.2.27.
30. Ткаченко К. О., Якименко В. Ю. Деякі аспекти використання асинхронної комунікації у мікросервісній архітектурі. ITSynergy. 2023. № 2. С. 27–43. DOI: 10.53920/its-2023-2-2.
31. Холоднюк С., Кузьміч М. Оптимізація потокової обробки даних в GCP (Google Cloud Platform). Наука і техніка сьогодні. 2025. № 8(49). С. 1772–1785. DOI: 10.52058/2786-6025-2025-8(49)-1772-1785.
32. “A Benchmark for Data Management in Microservices,” arXiv:2403.12605, Mar. 2024. [Online]. Available: <https://arxiv.org/abs/2403.12605>. [Accessed: 19-Oct-2025]. arXiv



33. Abusalah B., Qadah T. M., Stephen J. J., Eugster P. Interminable Flows: A Generic, Joint, Customizable Resiliency Model for Big-Data Streaming Platforms. *IEEE Access*. 2023. Vol. 11. P. 10762–10776. DOI: 10.1109/access.2023.3239365.
34. Aderaldo C. M., Costa T. M., Vasconcelos D. M., Mendonça N. C., Cámara J., Garlan D. A declarative approach and benchmark tool for controlled evaluation of microservice resiliency patterns. *Software: Practice and Experience*. 2025. Vol. 55, no. 1. P. 170–192. DOI: 10.1002/spe.3368.
35. Akili S., Purtzel S., Weidlich M. INEv: In-Network Evaluation for Event Stream Processing. *Proceedings of the ACM on Management of Data*. 2023. Vol. 1, no. 1. P. 1–26. DOI: 10.1145/3588955.
36. Amdahl G.M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *Proceedings of the AFIPS Spring Joint Computer Conference*. Atlantic City, NJ, April 18–20, 1967. Vol. 30. P. 483–485. DOI: 10.1145/1465482.1465560.
37. Andelfinger P., Köster T., Uhrmacher A. M. The Window Racer Algorithm for Parallel Discrete Event Simulation. *Proceedings of the ACM*, 2023. DOI: 10.1145/3573900.3591115.
38. Apache Kafka, “Kafka Documentation,” Apache Kafka Project. [Online]. Available: <https://kafka.apache.org/documentation/>. [Accessed: 19-Oct-2025]. Apache Kafka
39. AWS Builders' Library, “Making retries safe with idempotent APIs,” Amazon Web Services. [Online]. Available: <https://aws.amazon.com/builders-library/making-retries-safe-with-idempotent-APIs/>. [Accessed: 19-Oct-2025]. Amazon Web Services, Inc.
40. Aydemir F., Başçiftçi F. Building a Performance Efficient Core Banking System Based on the Microservices Architecture. *Journal of Grid Computing*. 2022. Vol. 20, no. 4. DOI: 10.1007/s10723-022-09624-z.
41. Bachan J., Ye J. C., Jiang X. et al. Devastator: A Scalable Parallel Discrete Event Simulation Framework for Modern C++. *Proceedings of the ACM*, 2024. DOI: 10.1145/3615979.3656061.

42. Baeldung, “Consumer Acknowledgments and Publisher Confirms with RabbitMQ,” Jul. 8, 2024. [Online]. Available: <https://www.baeldung.com/rabbitmq-consumer-acknowledgments-publisher-confirmations>. [Accessed: 19-Oct-2025].  
Baeldung on Kotlin
43. Batista E., Coelho P., Alchieri E. et al. FlexCast: An Efficient Atomic Multicast Protocol for Distributed Systems. *Proceedings of the ACM*, 2023. DOI: 10.1145/3590140.3629122.
44. Bhatt N., Thakkar A. An efficient approach for low latency processing in stream data. *PeerJ Computer Science*. 2021. Vol. 7. P. e426. DOI: 10.7717/peerj-cs.426.
45. Bhupatiraju R. V. Event-Driven Architecture for Payment Failover and Redundancy: A Framework for High-Availability Financial Transaction Processing. *European Modern Studies Journal*, 2025. DOI: 10.59573/emsj.9(5).2025.75.
46. Brewer E.A. Towards Robust Distributed Systems (Invited Keynote). *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC 2000)*. Portland, OR, July 2000.
47. Burckhardt S., Gillum C., Justo D., Kallas K., McMahon C., Meiklejohn C. S. Durable functions: semantics for stateful serverless. *Proceedings of the ACM on Programming Languages*. 2021. Vol. 5, no. OOPSLA. P. 1–27. DOI: 10.1145/3485510.
48. Cai B., Wang B., Yang M., Guo Q. AutoMan: Resource-efficient provisioning with tail latency guarantees for microservices. *Future Generation Computer Systems*. 2023. Vol. 143. P. 61–75. DOI: 10.1016/j.future.2023.01.014.
49. Canali C., Lancellotti R., Pedroni P. Microservice Performance in Container- and Function-as-a-Service Architectures. *2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2022. P. 1–6. DOI: 10.23919/softcom55329.2022.9911406.
50. Chejarla J. R. Event-Driven Cloud-Native Order Management System Architecture. *Zenodo Repository*. 2025. DOI: 10.5281/zenodo.16408023.
51. Chy M. S. H., Arju M. A. R., Tella S. M., Cerny T. Comparative Evaluation of Java Virtual Machine-Based Message Queue Services: A Study on Kafka, Artemis, Pulsar, and RocketMQ. *Electronics*. 2023. Vol. 12, no. 23. P. 4792. DOI: 10.3390/electronics12234792.

52. CloudAMQP Blog, “RabbitMQ: Exchanges, routing keys and bindings,” Sep. 24, 2019. [Online]. Available: <https://www.cloudamqp.com/blog/part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html>. [Accessed: 19-Oct-2025]. CloudAMQP
53. Confluent, “Message Delivery Guarantees for Apache Kafka,” Confluent Documentation. [Online]. Available: <https://docs.confluent.io/kafka/design/delivery-semantics.html>. [Accessed: 19-Oct-2025]. docs.confluent.io
54. Daraghmi E. Y., Zhang C., Yuan S.-M. Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture. *Applied Sciences*. 2022. Vol. 12, Issue 12. DOI: 10.3390/app12126242.
55. de Heus M., Psarakis K., Fragkoulis M., Katsifodimos A. Transactions across serverless functions leveraging stateful dataflows. *Information Systems*. 2022. Vol. 108. P. 102015. DOI: 10.1016/j.is.2022.102015.
56. Detti A., Funari L., Petrucci L.  $\mu$ Bench: An Open-Source Factory of Benchmark Microservice Applications. *IEEE Transactions on Parallel and Distributed Systems*. 2023. Vol. 34, no. 3. P. 968–980. DOI: 10.1109/tpds.2023.3236447.
57. Di Francesco P., Lago P., Malavolta I. Migrating Towards Microservice Architectures: An Industrial Survey. *Proceedings of the 15th IEEE International Conference on Software Architecture (ICSA 2018)*. IEEE, 2018. P. 29–38. DOI: 10.1109/ICSA.2018.00012.
58. Dobbelaere P., Sheykh Esmaili K. Kafka versus RabbitMQ: A Comparative Study of Two Industry Reference Publish/Subscribe Implementations. *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17)*. New York : ACM, 2017. P. 227–238. DOI: 10.1145/3093742.3093908.
59. Dragoni N., Giallorenzo S., Lluch Lafuente A., Mazzara M., Montesi F., Mustafin R., Safina L. *Microservices: Yesterday, Today, and Tomorrow. Present and Ulterior Software Engineering* / ed. M. Mazzara, B. Meyer. Cham : Springer, 2017. P. 195–216. DOI: 10.1007/978-3-319-67425-4\_12.

60. Eker A., Arafa Y., Badawy A. A. et al. Load-Aware Dynamic Time Synchronization in Parallel Discrete Event Simulation. *Proceedings of the ACM*, 2021. DOI: 10.1145/3437959.3459249.
61. Esen E., Akbulut A., Catal C. Chaos experiments in microservice architectures: A systematic literature review. *Computer Standards & Interfaces*. 2026. Vol. 97. P. 104116. DOI: 10.1016/j.csi.2025.104116.
62. Eugster P.T., Felber P.A., Guerraoui R., Kermarrec A.-M. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*. 2003. Vol. 35, No. 2. P. 114–131. DOI: 10.1145/857076.857078.
63. Garcia-Molina H., Salem K. Sagas. *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data (SIGMOD '87)*. New York : ACM, 1987. P. 249–259. DOI: 10.1145/38713.38742.
64. Geldenhuys M., Pfister B., Scheinert D., Thamsen L., Kao O. Khaos: Dynamically Optimizing Checkpointing for Dependable Distributed Stream Processing. *Annals of Computer Science and Information Systems*. 2022. Vol. 30. P. 553–561. DOI: 10.15439/2022f225.
65. Gilbert S., Lynch N. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News*. 2002. Vol. 33, No. 2. P. 51–59. DOI: 10.1145/564585.564601.
66. González-Aparicio M. T., Younas M., Tuya J., Casado R. A transaction platform for microservices-based big data systems. *Simulation Modelling Practice and Theory*. 2023. Vol. 123. P. 102709. DOI: 10.1016/j.simpat.2022.102709.
67. Gunther N.J. How to Quantify Scalability: The Universal Scalability Law. *Performance Dynamics*. 2007. URL: <https://www.perfdynamics.com/Manifesto/USLscalability.pdf> (дата звернення: 01.03.2024).
68. Gustafson J.L. Reevaluating Amdahl's Law. *Communications of the ACM*. 1988. Vol. 31, No. 5. P. 532–533. DOI: 10.1145/42411.42415.
69. Górski T. UML Profile for Messaging Patterns in Service-Oriented Architecture, Microservices, and Internet of Things. *Applied Sciences*. 2022. Vol. 12, no. 24. P. 12790. DOI: 10.3390/app122412790.

70. Hassan H., Abdel-Fattah M. A., Mohamed W. A Pattern-Based Framework for Automated Migration of Monolithic Applications to Microservices. *Big Data and Cognitive Computing*. 2025. Vol. 9, no. 10. P. 253. DOI: 10.3390/bdcc9100253.
71. Henning S., Hasselbring W. A configurable method for benchmarking scalability of cloud-native applications. *Empirical Software Engineering*. 2022. Vol. 27, no. 6. DOI: 10.1007/s10664-022-10162-1.
72. Henning S., Hasselbring W. Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. *Journal of Systems and Software*. 2024. Vol. 208. P. 111879. DOI: 10.1016/j.jss.2023.111879.
73. Jain R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York : Wiley, 1991. 720 p.
74. Jayasekara S., Karunasekera S., Harwood A. Optimizing checkpoint-based fault-tolerance in distributed stream processing systems: Theory to practice. *Software: Practice and Experience*. 2022. Vol. 52, no. 1. P. 296–315. DOI: 10.1002/spe.3021.
75. Junfeng T., Wenqing B., Haoyi J. PGCE: A distributed storage causal consistency model based on partial geo-replication and cloud-edge collaboration architecture. *Computer Networks*. 2022. Vol. 212. P. 109065. DOI: 10.1016/j.comnet.2022.109065.
76. Kassetty N., Jain A. FIN-EVENTS+: Designing Scalable Event-Driven Microservice Architectures for Real-Time Payment Processing and Transaction Management in Modern Financial Systems. *International Journal of Research in Engineering, Management and Emerging Technology*. 2025. Vol. 13, Issue 3. DOI: 10.63345/ijrmeet.org.v13.i3.12.
77. Khan M. G., Taheri J., Al-Dulaimy A., Kassler A. PerfSim: A Performance Simulator for Cloud Native Microservice Chains. *IEEE Transactions on Cloud Computing*. 2023. Vol. 11, no. 2. P. 1395–1413. DOI: 10.1109/tcc.2021.3135757.

78. Khan S., Rydow E., Etemaditajbakhsh S., Adamek K., Armour W. Web Performance Evaluation of High Volume Streaming Data Visualization. *IEEE Access*. 2023. Vol. 11. P. 15623–15636. DOI: 10.1109/access.2023.3245043.
79. Khriji S., Benbelgacem Y., Chéour R., Houssaini D. E., Kanoun O. Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *The Journal of Supercomputing*. 2022. Vol. 78, no. 3. P. 3374–3401. DOI: 10.1007/s11227-021-03955-6.
80. Kim G., Lee W. In-network leaderless replication for distributed data stores. *Proceedings of the VLDB Endowment*. 2022. Vol. 15, no. 7. P. 1337–1349. DOI: 10.14778/3523210.3523213.
81. Kreps J., Narkhede N., Rao J. Kafka: A Distributed Messaging System for Log Processing. *Proceedings of the NetDB Workshop (NetDB '11)*. Athens, Greece, June 12, 2011. P. 1–7.
82. Kumar R. Event-Driven Architectures for Real-Time Data Processing: A Deep Dive into System Design and Optimization. *Zenodo Repository*. 2025. DOI: 10.5281/zenodo.15026989.
83. S. Kumar, A. Jadon, S. Sharma, “Global Message Ordering using Distributed Kafka Clusters,” *arXiv:2309.04918*, Sep. 2023. [Online]. Available: <https://arxiv.org/abs/2309.04918>. [Accessed: 19-Oct-2025]. *arXiv*
84. Laigner R., Kalinowski M., Diniz P., Zhou Y. Suffering-Oriented Programming: Building Microservices Applications Through Challenges. *Proceedings of the 44th IEEE Annual Computers, Software, and Applications Conference (COMPSAC 2020)*. IEEE, 2020. P. 647–656. DOI: 10.1109/COMPSAC48688.2020.00-48.
85. Laigner R., Zhou Y., Salles M. A. V., Liu Y., Kalinowski M. Data management in microservices. *Proceedings of the VLDB Endowment*. 2021. Vol. 14, no. 13. P. 3348–3361. DOI: 10.14778/3484224.3484232.
86. R. Laigner, A. C. Almeida, W. K. G. Assunção, Y. Zhou, “An Empirical Study on Challenges of Event Management in Microservice Architectures,” *arXiv:2408.00440*, Aug. 2024. [Online]. Available: <https://arxiv.org/abs/2408.00440>. [Accessed: 19-Oct-2025]. *arXiv*

87. Lamport L. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*. 1978. Vol. 21, No. 7. P. 558–565. DOI: 10.1145/359545.359563.
88. Lee W.-T., Song P.-Y., Tsai M.-K. et al. A High Availability Microservices Architecture Implementation using Saga and Backup Mechanism. *Proceedings of the International Conference on Dependable Systems and Applications (DSA)*. 2023. DOI: 10.1109/dsa59317.2023.00063.
89. Li P., Pan L., Yang X., Song W., Xiao Z., Birman K. Stabilizer: Geo-Replication with User-defined Consistency. *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. 2022. P. 359–369. DOI: 10.1109/icdcs54860.2022.00042.
90. Luo S., Xu H., Lu C., et al. An In-Depth Study of Microservice Call Graph and Runtime Performance. *IEEE Transactions on Parallel and Distributed Systems*. 2022. Vol. 33, no. 12. P. 3901–3914. DOI: 10.1109/tpds.2022.3174631.
91. Ma L., Liu Z., Xiong J., Jiang D. QWin: Core Allocation for Enforcing Differentiated Tail Latency SLOs at Shared Storage Backend. *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. 2022. P. 1098–1109. DOI: 10.1109/icdcs54860.2022.00109.
92. Martinez H. F., Mondragon O. H., Rubio H. A., Marquez J. Computational and Communication Infrastructure Challenges for Resilient Cloud Services. *Computers*. 2022. Vol. 11, no. 8. P. 118. DOI: 10.3390/computers11080118.
93. Martínez Saucedo A., Rodríguez G., Gomes Rocha F., Santos R. P. D. Migration of monolithic systems to microservices: A systematic mapping study. *Information and Software Technology*. 2025. Vol. 177. P. 107590. DOI: 10.1016/j.infsof.2024.107590.
94. McGlohon N., Carothers C. Deterministic Event Ordering Techniques for Parallel Discrete Event Simulation Systems. *ACM Digital Library*.
95. Meijer W., Trubiani C., Aleti A. Experimental evaluation of architectural software performance design patterns in microservices. *Journal of Systems and Software*. 2024. Vol. 218. P. 112183. DOI: 10.1016/j.jss.2024.112183.

96. Merli M., Guo S., Li P., Chen H., Lu N. Ursa: A Lakehouse-Native Data Streaming Engine for Kafka. *Proceedings of the VLDB Endowment*. 2025. Vol. 18, no. 12. P. 5184–5196. DOI: 10.14778/3750601.3750636.
97. Microservices.io, “Pattern: Transaction log tailing,” *Microservices Patterns*. [Online]. Available: <https://microservices.io/patterns/data/transaction-log-tailing.html>. [Accessed: 19-Oct-2025]. microservices.io
98. Microservices.io, “Patterns for Microservices Architecture,” Chris Richardson. [Online]. Available: <https://microservices.io/patterns/>. [Accessed: 19-Oct-2025]. microservices.io
99. G. Morling, “Revisiting the Outbox Pattern,” *Decodable Blog*, Oct. 31, 2024. [Online]. Available: <https://www.decodable.co/blog/revisiting-the-outbox-pattern>. [Accessed: 19-Oct-2025]. decodable.co
100. Narkhede N. Exactly-once Semantics is Possible: Here's How Apache Kafka Does it. *Confluent*. URL: <https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/>
101. Newman S. *Building Microservices: Designing Fine-Grained Systems*. Sebastopol : O'Reilly Media, 2015. 278 p.
102. Nikolic L., Dimitrieski V., Celikovic M. An approach for supporting transparent acid transactions over heterogeneous data stores in microservice architectures. *Computer Science and Information Systems*. 2024. Vol. 21, no. 1. P. 167–202. DOI: 10.2298/csis221210006n.
103. Nygard M.T. *Release It! Design and Deploy Production-Ready Software*. 2nd ed. Raleigh : Pragmatic Bookshelf, 2018. 376 p.
104. Ortiz G., Bazan-Muñoz A., Lamersdorf W., Garcia-de-Prado A. Evaluating the integration of Esper complex event processing engine and message brokers. *PeerJ Computer Science*. 2023. Vol. 9. P. e1437. DOI: 10.7717/peerj-cs.1437.
105. Pham V. N., Hossain M. D., Lee G. W., Huh E. N. Efficient Data Delivery Scheme for Large-Scale Microservices in Distributed Cloud Environment. *Applied Sciences*. 2023. Vol. 13, no. 2. P. 886. DOI: 10.3390/app13020886.
106. Punithavathy E., Priya N. Auto retry circuit breaker for enhanced performance in microservice applications. *International Journal of Electrical and*



Computer Engineering (IJECE). 2024. Vol. 14, no. 2. P. 2274–2281. DOI: 10.11591/ijece.v14i2.pp2274-2281.

107. Purella S. Microservices and Event-Driven Architectures in High-Volume Financial Systems. Zenodo Repository. 2025. DOI: 10.5281/zenodo.16150784.

108. Queues | RabbitMQ. RabbitMQ: One broker to queue them all | RabbitMQ. URL: <https://www.rabbitmq.com/docs/queues>

109. RabbitMQ Team. Broker Semantics: AMQP 0-9-1 Core Specification, Section 4.7 Ordering Guarantees. RabbitMQ Documentation. 2023. URL: <https://www.rabbitmq.com/docs/semantics> (дата звернення: 01.03.2024).

110. RabbitMQ, “Consumer Acknowledgements and Publisher Confirms,” RabbitMQ Documentation. [Online]. Available: <https://www.rabbitmq.com/docs/confirms>. [Accessed: 19-Oct-2025]. rabbitmq.com

111. RabbitMQ, “Publishers – Strategies for Using Publisher Confirms,” RabbitMQ Documentation. [Online]. Available: <https://www.rabbitmq.com/docs/publishers.html>. [Accessed: 19-Oct-2025]. rabbitmq.com

112. RabbitMQ, “Reliable Publishing with Publisher Confirms,” RabbitMQ Tutorials, Tutorial 7. [Online]. Available: <https://www.rabbitmq.com/tutorials/tutorial-seven-java.html>. [Accessed: 19-Oct-2025]. rabbitmq.com

113. Ramisetty G. Event-Driven Microservices for Ultra-Low Latency Cloud Workflows. Global Journal of Computer Science and Technology. 2025. Vol. 25, Issue 1. DOI: 10.34257/gjcsbvol25is1pg1.

114. Rampérez V., Soriano J., Lizcano D., Miguel C. Automatic Evaluation and Comparison of Pub/Sub Systems Performance Improvements. Journal of Web Engineering. 2022. DOI: 10.13052/jwe1540-9589.2144.

115. Raptis T. P., Cicconetti C., Passarella A. Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications. Future Generation Computer Systems. 2024. Vol. 154. P. 173–188. DOI: 10.1016/j.future.2023.12.028.

116. Raptis T. P., Passarella A. A Survey on Networked Data Streaming With Apache Kafka. *IEEE Access*. 2023. Vol. 11. P. 85333–85350. DOI: 10.1109/access.2023.3303810.
117. Rehman A. U., Aguiar R. L., Barraca J. P. Fault-Tolerance in the Scope of Cloud Computing. *IEEE Access*. 2022. Vol. 10. P. 63422–63441. DOI: 10.1109/access.2022.3182211.
118. C. Richardson, “Transactional Outbox Pattern,” *Microservices.io*. [Online]. Available: <https://microservices.io/patterns/data/transactional-outbox.html>. [Accessed: 19-Oct-2025]. *microservices.io*
119. Richardson C. *Microservices Patterns: With Examples in Java*. Shelter Island : Manning Publications, 2018. 520 p.
120. Richardson C. Pattern: Event Sourcing. *Microservices.io*. 2018. URL: <https://microservices.io/patterns/data/event-sourcing.html> (дата звернення: 01.03.2024).
121. Richardson C. Pattern: Saga. *Microservices.io*. 2018. URL: <https://microservices.io/patterns/data/saga.html> (дата звернення: 01.03.2024).
122. Ristov S., Kimovski D., Fahringer T. FaaScinating Resilience for Serverless Function Choreographies in Federated Clouds. *IEEE Transactions on Network and Service Management*. 2022. Vol. 19, no. 3. P. 2440–2452. DOI: 10.1109/tnsm.2022.3162036.
123. Rodrigues H., Rito Silva A., Avritzer A. Assessment of performance and its scalability in microservice architectures: Systematic literature review. *Journal of Systems and Software*. 2025. Vol. 230. P. 112500. DOI: 10.1016/j.jss.2025.112500.
124. Ruchel L. V., Tavares de Camargo E., Rodrigues L. A., Turchetti R. C., Arantes L., Procópio Duarte E. Scalable atomic broadcast: A leaderless hierarchical algorithm. *Journal of Parallel and Distributed Computing*. 2024. Vol. 184. P. 104789. DOI: 10.1016/j.jpdc.2023.104789.
125. Santos J., Wauters T., Volckaert B., De Turck F. Towards Low-Latency Service Delivery in a Continuum of Virtual Resources: State-of-the-Art and Research Directions. *IEEE Communications Surveys & Tutorials*. 2021. Vol. 23, no. 4. P. 2557–2589. DOI: 10.1109/comst.2021.3095358.

126. SAP Community, “Message Replay (SAP Event Mesh),” SAP Community / Event Mesh. [Online]. Available: <https://community.sap.com/t5/technology-blog-posts-by-members/reversing-the-time-with-advanced-event-mesh-unleash-the-power-of-message/ba-p/13548246> (overview) and <https://help.pubsub.em.services.cloud.sap/Features/Replay/Message-Replay-Overview.htm> (feature). [Accessed: 19-Oct-2025]. community.sap.com+1
127. “Service-outbox: Transactional outbox pattern for microservices,” GitHub Repository. [Online]. Available: <https://github.com/yornaath/service-outbox>. [Accessed: 19-Oct-2025]. GitHub
128. SoftwareMill, “Microservices 101: Transactional Outbox and Inbox,” SoftwareMill blog, Jun. 3, 2022. [Online]. Available: <https://softwaremill.com/microservices-101/>. [Accessed: 19-Oct-2025]. SoftwareMill
129. “SoK: Consensus for Fair Message Ordering,” arXiv:2411.09981, Nov. 2024 (updated Jun. 2025). [Online]. Available: <https://arxiv.org/abs/2411.09981>. [Accessed: 19-Oct-2025]. arXiv
130. Solovei O., Honcharenko T. A bayesian-driven feedforward neural network model for Kafka cluster latency forecasting. Radioelectronic and Computer Systems. 2025. № 3. C. 68–87. DOI: 10.32620/reks.2025.3.05.
131. Taibi D., Lenarduzzi V., Pahl C. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. IEEE Cloud Computing. 2017. Vol. 4, No. 5. P. 22–32. DOI: 10.1109/MCC.2017.4250931.
132. Velepucha V., Flores P. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access. 2023. Vol. 11. P. 88339–88358. DOI: 10.1109/access.2023.3305687.
133. Vernon V. Implementing Domain-Driven Design. Upper Saddle River : Addison-Wesley, 2013. 656 p.
134. Wang G., Koshy J., Subramanian S., Paramasivam K., Zadeh M., Narkhede N., Rao J., Kreps J., Stein J. Building a Replicated Logging System with Apache Kafka. Proceedings of the VLDB Endowment. 2015. Vol. 8, No. 12. P. 1654–1655. DOI: 10.14778/2824032.2824063.

135. Xu H., Liu P., Ahmed S. T., Da Silva D., Hu L. Adaptive Fragment-Based Parallel State Recovery for Stream Processing Systems. *IEEE Transactions on Parallel and Distributed Systems*. 2023. Vol. 34, no. 8. P. 2464–2478. DOI: 10.1109/tpds.2023.3251997.
136. Xu J., Yin J., Zhu H., Xiao L. Formalization and verification of Kafka messaging mechanism using CSP. *Computer Science and Information Systems*. 2023. Vol. 20, no. 1. P. 277–306. DOI: 10.2298/csis210707057x.
137. Yang W., Chen P., Liu K., Zhang H. ZeroTracer: In-Band eBPF-Based Trace Generator With Zero Instrumentation for Microservice Systems. *IEEE Transactions on Parallel and Distributed Systems*. 2025. Vol. 36, no. 7. P. 1478–1494. DOI: 10.1109/tpds.2025.3571934.
138. V. Zhebka, et al., Methods for predicting failures in a smart home, in: *Digital Economy Concepts and Technologies Workshop*, vol. 3665, 2024, 70–78.

## Додаток А. Лістинг програмного коду реалізації методу упорядкованої паралельної доставки подій

Диспетчеризація події за станом послідовності ключа

class OrderedParallelSimulator:

```
    """Запропонований метод упорядкованого паралелізму: забезпечує
    O = 1.000 (послідовність у межах ключа через буферизацію за станом seq),
    duplicate_pass_rate = 0 (ідемпотентність за умовою seq <= last_seq[key])."""

    def __init__(self, k, seed, params=PARAMS):
        self.k = k
        self.params = params
        self._rng = np.random.default_rng(seed)
        # Стан за ключем
        self._last_seq = defaultdict(int)          # останній оброблений seq
        self._buffer = defaultdict(lambda: SortedList(key=lambda e: e.seq)) # позачергові події
        self._key_vt = defaultdict(float) # віртуальний годинник послідовного слоту ключа
        self._worker_vt = []                      # пул із k віртуальних годинників обробників

    def _handle_event(self, event):
        """Диспетчеризація однієї події. Упорядкованість у межах ключа забезпечується
        станом послідовності; позачергові події буферизуються до заповнення розриву."""
        key, seq, last = event.key, event.seq, self._last_seq[event.key]

        if seq <= last:
            # дубль: seq уже оброблено — ідемпотентне ігнорування
            self._dup_count += 1
            return

        if seq == last + 1:
            # у порядку: обробити негайно, потім вивільнити буфер
            self._do_process(key, event)
            self._last_seq[key] = seq
            self._flush_buffer(key)
        else:
            # позачергово: буферизувати (з дедуплікацією за seq)
            buf = self._buffer[key]
            if seq not in {e.seq for e in buf}:
                buf.add(event)

    def _flush_buffer(self, key):
        """Каскадно вивільнити буферизовані події, що стали впорядкованими."""
        buf = self._buffer[key]
        while buf and buf[0].seq == self._last_seq[key] + 1:
            evt = buf.pop(0)
            self._do_process(key, evt)
```

```

self._last_seq[key] = evt.seq

def _do_process(self, key, event):
    """Зайняти найраніше вільний обробник; просунути годинник обробника та ключа."""
    proc_ms = float(lognormal_processing(
        self._rng, self.params.T_p, self.params.sigma_p, 1)[0])
    w = int(np.argmin(self._worker_vt))      # обробник, що звільняється найраніше
    # Початок не раніше: надходження події, звільнення обробника, завершення попередньої події ключа
    vt_start = max(event.timestamp_ms, self._worker_vt[w], self._key_vt[key])
    vt_done = vt_start + proc_ms + self.params.T_i # T_i — вартість перевірки last_seq
    self._worker_vt[w] = vt_done
    self._key_vt[key] = vt_done
    self._completion_ms.append(vt_done)
    self._latencies.append(vt_done - event.timestamp_ms)
    self._processed_log.append((key, event.seq))

```

## Додаток Б. Лістинг програмного коду реалізації аналітичної моделі пропускної здатності системи упорядкованої доставки подій

MS\_PER\_S = 1000.0

class AnalyticalModel:

"""Аналітична модель пропускної здатності упорядкованого паралелізму:

Xg(k) = 1000 \* k / (s\_skew \* (A + B\*k)) [подій/с]."""

def \_\_init\_\_(self, T\_p, T\_i, alpha, p\_r, beta, CV, p\_h, T\_h, gamma, s\_skew):

self.T\_p, self.T\_i, self.alpha, self.p\_r = T\_p, T\_i, alpha, p\_r

self.beta, self.CV, self.p\_h, self.T\_h = beta, CV, p\_h, T\_h

self.gamma, self.s\_skew = gamma, s\_skew

def compute\_A(self): # стала вартість обробки події A [мс]

return self.T\_p + self.T\_i + (self.alpha \* self.p\_r) / (1.0 - self.p\_r) \

+ self.p\_h \* self.T\_h

def compute\_B(self): # лінійна вартість на обробник B [мс/обробник]

return self.beta \* self.CV + self.gamma

def Xg(self, k): # пропускна здатність при k обробниках [подій/с]

k = np.asarray(k, dtype=float)

T\_eff = self.compute\_A() + self.compute\_B() \* k

return MS\_PER\_S \* k / (self.s\_skew \* T\_eff)

def Xg\_max(self): # стеля масштабованості Xg(k -> inf)

return MS\_PER\_S / (self.s\_skew \* self.compute\_B())

def Ep(self, k): # ефективність паралелізму Ep(k) = Xg(k)/Xideal(k)

k = np.asarray(k, dtype=float)

X\_ideal = MS\_PER\_S \* k / self.T\_p

return self.Xg(k) / X\_ideal

def k\_optimal(self, epsilon=0.5): # найбільше k, за якого Ep(k) >= epsilon (замкнена форма)

A, B = self.compute\_A(), self.compute\_B()

k\_star = (self.T\_p / (self.s\_skew \* epsilon) - A) / B

return int(round(k\_star)) if k\_star > 0 else None

def is\_scalable(self, lambda\_nominal): # умова стійкості: (1 - p\_r) > lambda\_eff \* s\_skew \* B

lambda\_eff = lambda\_nominal / (1.0 - self.p\_r)

B\_s = self.compute\_B() / MS\_PER\_S

return (1.0 - self.p\_r) > lambda\_eff \* self.s\_skew \* B\_s

def rho(self, k, lambda\_nominal): # коефіцієнт завантаження rho(k)

return (lambda\_nominal / (1.0 - self.p\_r)) / self.Xg(k)

## Додаток В. Лістинг програмного коду реалізації архітектури безперервної

### доставки подій

Включено: виявлення збою, двоканальне резервування та відновлення Saga

```
def _heartbeat_detect(rng, interval_ms=50.0, threshold=2):
    """Час heartbeat-детектування збою [мс]. Збій настає з рівномірним зсувом у
    поточному циклі; threshold-й пропущений сигнал дає
     $T_{detect} \sim \text{Uniform}((\text{threshold}-1) * \text{interval}, \text{threshold} * \text{interval}) + N(0, 5)$ ."""
    low, high = (threshold - 1) * interval_ms, threshold * interval_ms
    return max(0.0, float(rng.uniform(low, high) + rng.normal(0.0, 5.0)))

class ProposedArch(_Architecture):
    """Двоканальна доставка: основний RabbitMQ + резервний HTTP з імпортованою
    машиною стану ідемпотентності last_seq.
     $MTTR = T_{detect} + T_{switch} + T_{reprocess}$ .
    Перервані Saga відновлюються з кроку N+1 (жоден завершений крок не повторюється),
    тож duplicate_operations = 0."""

    name = "proposed"
    p_step_corrupt = 0.0    # ідемпотентно: повтор кроку не пошкоджує транзакцію
    dup_fraction = 0.0

    def _recovery(self, rng, fault_duration_ms):
        t_switch = _lognormal(rng, 90.0, 20.0) # перемикання на HTTP-канал
        t_reprocess = _lognormal(rng, 58.0, 12.0) # відтворення незавершених кроків Saga
        return t_switch, t_reprocess

    def _saga_steps_replayed(self, N):
        return np.zeros_like(N)    # відновлення з N+1: повторів немає

    def _per_event_loss(self, p_loss_primary, p_loss_http):
        return p_loss_primary * p_loss_http    # втрата лише за відмови ОБОХ каналів

    def measure_idempotency(self, seed):
        """Прогнати машину стану Результату 1 на потоці з копіями at-least-once.
        Умова seq <= last_seq[key] робить кількість дублів у бізнес-логіці нульовою."""
        rng = np.random.default_rng(seed ^ 0x1D)
        events = generate_event_stream(rng, num_events=3000,
                                       p_loss=0.005, p_duplicate=self.params.p_r)
        sim = OrderedParallelSimulator(k=self.k, seed=seed ^ 0x1D, params=self.params)
        result = sim.run(events, warmup=self.params.WARMUP_EVENTS, p_loss=0.005)
        passed = int(round(result["duplicate_pass_rate"] * result["n_duplicates"]))
        return {"duplicate_operations": passed}

    # Композиція MTTR у run_scenario (спільна для всіх архітектур):
    t_detect = self._detect(rng)
    t_switch_or_restart, t_reprocess = self._recovery(rng, fault_duration_ms)
    mtr = t_detect + t_switch_or_restart + t_reprocess
```



## Додаток Г. Акти впровадження

### ЗАТВЕРДЖУЮ

Директор Інституту  
телекомунікацій і глобального  
інформаційного простору  
Національної академії наук  
України



*Андрій*  
" 1 " *серпня* 2026 р.

### АКТ

впровадження наукових результатів дисертації  
аспіранта Державного університету інформаційно-комунікаційних  
технологій  
КОЛОДЮКА Андрія Васильовича

Комісія у складі голови комісії – завідувача відділу дослідження навколишнього середовища Інституту телекомунікацій і глобального інформаційного простору Національної академії наук України д.т.н., професора Триснюка В.М., та членів комісії: головного наукового співробітника Інституту телекомунікацій і глобального інформаційного простору Національної академії наук України д.т.н., с.н.с. Яковлева Є.О., старшого наукового співробітника, к.т.н., старшого дослідника Охарєва В.О. розглянула матеріали дисертаційного дослідження аспіранта Колодюка А. В. на тему «Метод упорядкованої паралельної доставки подій для підвищення стабільності й масштабованості мікросервісних систем на основі технологій асинхронного обміну повідомленнями та аналітичного моделювання».

Комісією підтверджено, що реалізація поставленого наукового завдання сприяє суттєвому підвищенню коректності, продуктивності та безперервності доставки подій у мікросервісних системах, побудованих за подієво-орієнтованою архітектурою з асинхронним обміном повідомленнями, що

досягається завдяки розробленому комплексу методу, аналітичній моделі та архітектурних засобів упорядкованої паралельної доставки подій.

У рамках дослідження розроблено та програмно реалізовано метод упорядкованого паралелізму в межах логічного ключа, який на основі прикладного впорядкування поверх брокера повідомлень, використання ідентифікаторів послідовності, внутрішніх черг за ключами та механізму відновлення пропущених станів забезпечує детерміновану впорядкованість обробки взаємопов'язаних подій за одночасного паралельного опрацювання незалежних сутностей. Практичне застосування методу дозволило зберегти коректну послідовність виконання операцій та забезпечити прискорення обробки подій до 2,42 рази порівняно з базовими підходами в умовах підвищеного навантаження і часткових збоїв.

Для обґрунтування проєктних рішень застосовано розроблену аналітичну модель впливу паралелізму на продуктивність і латентність системи упорядкованої доставки подій, яка враховує вплив повторної доставки повідомлень, ідемпотентної обробки, буферизації пропущених станів та використання резервного каналу доставки. Використання моделі забезпечує оцінювання накладних витрат, визначення раціонального рівня паралелізму та прогнозування продуктивності системи з відносною похибкою не більше 5,5 %, що дозволяє застосовувати її під час проєктування, масштабування та модернізації високонавантажених мікросервісних систем.

На основі інтеграції зазначеного методу, аналітичної моделі, основного брокера повідомлень, резервного каналу доставки та механізму узгодженого виконання розподілених кроків впроваджено вдосконалену архітектуру забезпечення безперервності доставки подій. Її застосування забезпечило скорочення середнього часу відновлення після збою на 76 % (223,5 мс проти 930,2 мс у базовій архітектурі), підвищення частки успішно доставлених подій до 99,8–99,9 % порівняно з 95,3 % у традиційних рішеннях, а також підтримання частки успішно завершених Saga-транзакцій на рівні 98,9 % проти 31,8 % за умов нестабільної роботи брокера повідомлень. При цьому додаткові накладні витрати за латентністю у штатному режимі функціонування не перевищують 2,4 %.

Практичне значення отриманих результатів полягає в тому, що впровадження розроблених у дисертаційному дослідженні елементів інформаційних технологій дозволяє створювати масштабовані, відмовостійкі та високонавантажені мікросервісні системи з гарантуванням упорядкованості, ідемпотентності та безперервності доставки подій за умов часткових збоїв інфраструктури.

Отримані результати використано для вдосконалення програмних засобів обробки потоків даних інформаційно-аналітичних систем установи.

Голова комісії:  
Члени комісії

	В.М. Триснюк
	Є.О. Яковлев
	В.О. Охарсва



## **АКТ**

**про впровадження результатів дисертаційного дослідження  
Колодюка Андрія Васильовича на тему: "Метод упорядкованої  
паралельної доставки подій для підвищення стабільності й  
масштабованості мікросервісних систем на основі технологій асинхронного  
обміну повідомленнями та аналітичного моделювання",  
поданої на здобуття наукового ступеня доктора філософії  
зі спеціальності 121 Інженерія програмного забезпечення**

Цим Актом, засвідчуємо, що нижчеперелічені наукові положення, а саме:

1. Набув подальшого розвитку метод упорядкованого паралелізму в межах логічного ключа у мікросервісних системах доставки подій на основі прикладного впорядкування поверх брокера повідомлень із використанням маркування подій ідентифікаторами послідовності та механізму відновлення пропущених станів, що дозволяє забезпечити коректне послідовне опрацювання взаємопов'язаних подій за одночасного паралельного виконання незалежних потоків обробки та підвищити продуктивність системи.

2. Розроблено аналітичну модель впливу паралелізму на продуктивність і латентність системи впорядкованої доставки подій у режимі гарантованої доставки з можливими повторами на основі комплексного врахування ідемпотентної обробки, повторної доставки повідомлень, буферизації пропущених станів і використання резервного каналу доставки, що дозволяє формалізовано оцінювати накладні витрати, визначати оптимальний рівень паралелізму та прогнозувати поведінку системи за умов зміни навантаження і часткових відмов.

3. Удосконалено архітектуру забезпечення безперервності доставки подій у мікросервісних системах на основі інтеграції методу упорядкованого паралелізму, аналітичної моделі оцінювання продуктивності та латентності, основного брокера повідомлень, резервного каналу доставки та механізму узгодженого виконання розподілених кроків, що дозволяє підвищити безперервність виконання процесів, скоротити час відновлення після відмов і

забезпечити коректність послідовності виконання операцій в умовах часткових збоїв інфраструктури.

Розроблені особисто Колодюком Андрієм Васильовичем у ході проведення ним дисертаційних досліджень та отримали високу оцінку при обговоренні на засіданнях кафедр факультету інформатики та обчислювальної техніки Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Зазначені наукові результати впроваджені в освітній процес кафедр факультету інформатики та обчислювальної техніки Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського", у робочих програмах навчальних дисциплін спеціальності 121 Інженерія програмного забезпечення.

Дослідження Колодюка Андрія Васильовича відповідає всім вимогам до організації наукового пошуку та дає позитивний результат у практичному застосуванні.

Декан факультету інформатики  
та обчислювальної техніки  
Національного технічного університету України  
"Київський політехнічний інститут  
імені Ігоря Сікорського"  
д.т.н., професор



Ярослав КОРНАГА



## АКТ

про впровадження результатів дисертаційної роботи

« 9 » 03 2026 р.

м.Київ

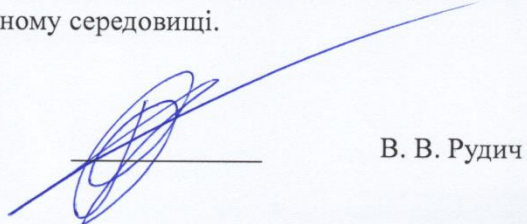
Результати дисертаційної роботи Колодюка Андрія Васильовича на тему: «Метод упорядкованої паралельної доставки подій для підвищення стабільності й масштабованості мікросервісних систем на основі аналітичного моделювання» були використані в діяльності компанії Byte Orchard Consulting у процесі проєктування та вдосконалення мікросервісних систем доставки подій і корпоративних веб-сервісів, побудованих на основі подієво-орієнтованої архітектури. У межах впровадження застосовано метод упорядкованого паралелізму per-замовлення із використанням прикладного рівня впорядкування поверх брокера повідомлень RabbitMQ, маркуванням подій ідентифікаторами послідовності X-Sequence-ID, механізмом динамічного створення внутрішніх черг per-key та механізмом відновлення пропущених станів gap-replay. Також було використано аналітичну модель впливу паралелізму на продуктивність і латентність системи доставки подій з урахуванням властивостей ідемпотентності, повторної доставки повідомлень, буферизації пропущених станів та резервного HTTP-каналу передачі.

Практичне використання результатів дослідження дозволило підвищити стабільність та масштабованість мікросервісної архітектури, забезпечити строгий порядок виконання операцій у межах окремих логічних контекстів, зменшити ризик виникнення неконсистентних станів під час асинхронної взаємодії сервісів, а також скоротити середній час відновлення системи після часткових відмов. Запропоновані методи та архітектурні рішення дали можливість підвищити пропускну здатність систем доставки подій, знизити затримки обробки повідомлень і забезпечити властивість живучості (liveness) процесів навіть за умов повторної доставки або тимчасової недоступності брокера повідомлень.

Отримані результати мають практичне значення для розробки сучасних корпоративних інформаційних систем, побудови високонавантажених мікросервісних платформ та створення відмовостійких систем обробки подій, що потребують узгодженості станів, безперервності роботи та адаптивного керування паралелізмом у розподіленому середовищі.

Директор  
ТОВ «Байт Орчард Консалтинг»



  
В. В. Рудич

ЗАТВЕРДЖУЮ

Перший проректор Державного  
університету інформаційно-  
комунікаційних технологій

Корченко О. Г.  
2026 року

## А К Т

**впровадження в навчальний процес Державного університету інформаційно-комунікаційних технологій наукових положень і результатів дисертаційної роботи Колодюка А. В. на тему «Метод упорядкованої паралельної доставки подій для підвищення стабільності й масштабованості мікросервісних систем на основі технологій асинхронного обміну повідомленнями та аналітичного моделювання»**

Науково-педагогічна комісія у складі голови – директора Навчально-наукового інституту Інформаційних технологій доктора технічних наук, професора Нестеренко К.С. та членів комісії: завідувача кафедри Інженерії програмного забезпечення доктора технічних наук, професора Замрій І.В., доцента кафедри Технологій цифрового розвитку кандидата фізико-математичних наук, доцента Поперешняк С.В. склала даний акт про те, що наукові положення і результати дисертаційної роботи на здобуття наукового ступеня доктора філософії Колодюка А. В. на тему «Метод упорядкованої паралельної доставки подій для підвищення стабільності й масштабованості мікросервісних систем на основі технологій асинхронного обміну повідомленнями та аналітичного моделювання», а саме



1. Метод упорядкованого паралелізму в межах логічного ключа, що на основі прикладного впорядкування поверх брокера повідомлень, маркування подій ідентифікаторами послідовності, внутрішніх черг за ключами та механізму відновлення пропущених станів забезпечує детерміновану впорядкованість обробки взаємопов'язаних подій за одночасного паралельного опрацювання незалежних потоків, введено в навчальний процес кафедри Комп'ютерної інженерії Державного університету інформаційно-комунікаційних технологій при викладанні дисципліни «Паралельні та розподілені обчислення» для студентів спеціальності 123 «Комп'ютерна інженерія» денної форми навчання.
2. Аналітична модель впливу паралелізму на продуктивність і латентність системи впорядкованої доставки подій, яка враховує повторну доставку повідомлень, ідемпотентну обробку, буферизацію пропущених станів і використання резервного каналу доставки та дозволяє оцінювати накладні витрати й визначати раціональний рівень паралелізму, введена в навчальний процес кафедри Інженерії програмного забезпечення Державного університету інформаційно-комунікаційних технологій при викладанні дисципліни «Моделювання та проектування програмного забезпечення» для студентів спеціальності 121 «Інженерія програмного забезпечення» денної форми навчання.
3. Удосконалена архітектура забезпечення безперервності доставки подій, побудована на інтеграції методу упорядкованого паралелізму, аналітичної моделі оцінювання продуктивності та латентності, основного брокера повідомлень, резервного каналу доставки та механізму узгодженого виконання розподілених кроків, що скорочує час відновлення після відмов і забезпечує коректність послідовності виконання операцій за умов часткових збоїв, введено в навчальний процес кафедри Інформаційних систем та технологій Державного університету інформаційно-комунікаційних технологій при викладанні

дисципліни «Архітектура інформаційних систем» для студентів спеціальності 126 «Інформаційні системи та технології» денної форми навчання.

Голова комісії:

Директор навчально-наукового інституту Інформаційних технологій

доктор технічних наук, професор



Нестеренко К.С.

Члени комісії:

Завідувач кафедри Інженерії програмного забезпечення

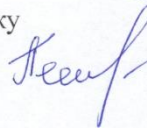
доктор технічних наук, професор



Замрій І.В.

Доцент кафедри Технологій цифрового розвитку

кандидат фізико-математичних наук, доцент



Поперешняк С.В.