УДК: 004.75:004.415.2 DOI: 10.31673/2412-9070.2026.318117

А. В. КОЛОДЮК, аспірант;
ORCID 0009-0001-1724-7531
Державний університет інформаційно-комунікаційних технологій, Київ

МЕТОД УПОРЯДКОВАНОЇ ПАРАЛЕЛЬНОЇ ДОСТАВКИ ПОДІЙ У МІКРОСЕРВІСНИХ СИСТЕМАХ ІЗ ВИКОРИСТАННЯМ RABBITMQ, РЕЗЕРВНИХ HTTP-КАНАЛІВ ТА ПАТЕРНУ SAGA ДЛЯ ПІДВИЩЕННЯ ЖИВУЧОСТІ ТА ЗМЕНШЕННЯ MTTR

У статті досліджено проблему забезпечення надійної, впорядкованої та відмовостійкої доставки подій у мікросервісних системах, що функціонують на основі асинхронного обміну повідомленнями. Актуальність роботи зумовлена тим, що в розподілених програмних системах порушення порядку подій, дублювання повідомлень або недоступність брокера можуть призводити до некоректного стану бізнес-процесів, збільшення часу відновлення та зниження живучості системи. Метою дослідження є розроблення методу упорядкованої паралельної доставки подій, який поєднує брокер RabbitMQ, резервний HTTP-канал і патерн Saga для керування розподіленими транзакціями.

У вступі обґрунтовано потребу в комплексному підході до організації доставки подій у мікросервісній архітектурі з урахуванням вимог до масштабованості, доступності та коректності виконання процесів. В огляді літератури проаналізовано наявні підходи до використання подієво-орієнтованої архітектури, брокерів повідомлень, механізмів повторної доставки, резервування каналів і патерну Saga. Встановлено, що більшість відомих рішень розглядає ці механізми окремо, тоді як питання їх інтеграції для одночасного забезпечення впорядкованості, живучості та зменшення MTTR залишається недостатньо дослідженим.

У методичній частині запропоновано архітектуру системи, що складається з джерел подій, брокера RabbitMQ, мікросервісів-споживачів, Saga-координатора, модуля контролю впорядкованості, буфера подій, механізму ідемпотентної обробки та резервного HTTP-маршруту. Подія формалізується як структура з унікальним ідентифікатором, ключем упорядкування, порядковим номером і корисним навантаженням. Для кожного ключа забезпечується послідовна обробка подій, а події з різними ключами можуть виконуватися паралельно. Це дає змогу поєднати високу продуктивність із контролем логічного порядку виконання.

У розділі аналітичного моделювання наведено показники оцінювання ефективності методу: імовірність успішної доставки, коефіцієнт упорядкованості, середній час відновлення та середню затримку доставки. Експериментальну перевірку виконано в симуляційному середовищі Python 3.11 із моделюванням нормального режиму роботи та сценаріїв відмов брокера. Отримані результати показали, що запропонований метод підвищує імовірність доставки подій до 99,8–99,9 %, забезпечує коефіцієнт упорядкованості близько 99,7 % і скорочує MTTR із 957 мс до 263 мс, тобто більш ніж на 70%. При цьому збільшення середньої затримки доставки є незначним і становить близько 3,7 %.

Практичне значення результатів полягає у можливості застосування запропонованого методу для побудови корпоративних і високонавантажених мікросервісних систем, у яких критичними є гарантована доставка подій, збереження їх послідовності, узгодженість розподілених транзакцій і швидке відновлення після часткових збоїв.

Ключові слова: мікросервісна архітектура, подієво-орієнтовані системи, RabbitMQ, Saga, резервний HTTP-канал, упорядкована доставка подій, відмовостійкість, MTTR.

Вступ

Сучасний розвиток розподілених інформаційних систем супроводжується активним впровадженням мікросервісної архітектури, яка забезпечує гнучкість масштабування, незалежність компонентів та підвищення ефективності супроводу програмних платформ. Важливою складовою таких систем є подієво-орієнтований підхід, що базується на асинхронному обміні повідомленнями між сервісами та дозволяє реалізувати слабке зв'язування компонентів і високий рівень продуктивності.

У високонавантажених корпоративних системах доставка подій здійснюється за допомогою брокерів повідомлень, серед яких значного поширення набув RabbitMQ. Використання брокерів дозволяє організувати буферизацію, маршрутизацію та масштабовану взаємодію між сервісами, проте зі збільшенням кількості компонентів та інтенсивності обміну даними зростають вимоги до надійності доставки, узгодженості розподілених процесів і забезпечення безперервності функціонування системи.

Особливої актуальності набувають питання збереження порядку виконання подій, підтримання живучості системи за умов часткових відмов, а також скорочення середнього часу відновлення після збоїв. У зв'язку з цим актуальним напрямом досліджень є розроблення методів організації доставки подій, здатних поєднати високу продуктивність, відмовостійкість та коректність виконання розподілених бізнес-процесів.

Постановка задачі

У мікросервісних системах, побудованих на основі асинхронного обміну повідомленнями, виникають проблеми, пов'язані з порушенням порядку подій, дублюванням повідомлень та переходом системи у неконсистентний стан внаслідок часткових відмов брокера або мережевої взаємодії. Особливо критичними такі ситуації є для розподілених бізнес-процесів, у яких послідовність виконання операцій безпосередньо впливає на коректність функціонування системи.

Існуючі підходи до забезпечення відмовостійкості мікросервісних архітектур, зокрема використання брокерів повідомлень, механізмів повторної доставки, резервних каналів зв'язку та патерну Saga, переважно розглядаються ізольовано та не забезпечують комплексного вирішення задачі впорядкованої паралельної доставки подій із одночасним підтриманням живучості системи та зменшенням MTTR.

У зв'язку з цим науково-прикладна задача полягає у розробленні методу доставки подій у мікросервісних системах, який забезпечує впорядковану паралельну обробку незалежних потоків подій, підтримує ідемпотентність виконання операцій, передбачає резервування каналів доставки та використовує механізм Saga для координації розподілених транзакцій і компенсації помилок.

Наукове значення задачі полягає у формалізації інтегрованого підходу до організації доставки подій у розподілених системах, тоді як практичне значення визначається можливістю застосування запропонованого методу для підвищення надійності та відмовостійкості високонавантажених мікросервісних платформ.

Аналіз останніх досліджень і публікацій

У сучасних умовах розвитку розподілених інформаційних систем мікросервісна архітектура та подієво-орієнтовані підходи (Event-Driven Architecture, EDA) займають провідне місце як ефективний засіб забезпечення масштабованості, гнучкості та відмовостійкості програмних систем [2], [3], [9], [10]. Основою таких архітектур є асинхронна взаємодія між сервісами, що реалізується за допомогою брокерів повідомлень, зокрема RabbitMQ, який забезпечує буферизацію, маршрутизацію та гарантовану доставку повідомлень між компонентами системи [2], [3], [9]. Аналіз сучасних досліджень свідчить, що використання брокерів повідомлень дозволяє досягти високих показників продуктивності та масштабованості. Водночас такі системи стикаються з низкою фундаментальних проблем, серед яких ключовими є забезпечення впо-

рядкованості подій, ідемпотентності обробки, а також відновлення після часткових збоїв [1], [7], [8].

Існуючі підходи здебільшого зосереджені на вирішенні окремих аспектів проблеми: брокери повідомлень забезпечують масштабованість і асинхронність, патерн Saga – узгодженість транзакцій, а механізми резервування – підвищення доступності [1], [2], [4], [6], [8]. Проте відсутні комплексні рішення, які б інтегрували зазначені підходи в єдину архітектуру з метою одночасного забезпечення впорядкованої паралельної доставки подій, живучості системи та мінімізації MTTR [1], [6], [8].

Аналіз літературних джерел виявляє наукову прогалину, що полягає у відсутності інтегрованих моделей доставки подій, які поєднують використання брокерів повідомлень, резервних каналів передачі та механізмів керування розподіленими транзакціями [1], [4], [6], [8]. Це зумовлює актуальність подальших досліджень, спрямованих на розробку методів, що забезпечують гарантовану, впорядковану та відмовостійку доставку подій у мікросервісних системах [1], [2], [6], [9].

Мета і задачі дослідження

Метою дослідження є підвищення надійності, доступності та живучості мікросервісних систем шляхом розроблення методу упорядкованої паралельної доставки подій, що поєднує використання брокера повідомлень RabbitMQ, резервного HTTP-каналу та механізму Saga для керування розподіленими транзакціями.

Для досягнення поставленої мети у статті передбачено розв'язання таких задач:

1. Проаналізувати наявні підходи до організації подієво-орієнтованої взаємодії в мікросервісних системах, зокрема із застосуванням брокерів повідомлень, резервних каналів доставки та патерну Saga.

2. Визначити обмеження наявних підходів щодо забезпечення впорядкованої паралельної доставки подій, живучості системи та скорочення середнього часу відновлення.

3. Розробити архітектуру методу доставки подій із використанням RabbitMQ як основного каналу, резервного HTTP-каналу та Saga-координатора.

4. Формалізувати механізм упорядкування подій при паралельній обробці на основі ключів упорядкування, порядкових номерів, буферизації та ідемпотентної обробки.

5. Побудувати аналітичну модель оцінювання ефективності запропонованого методу за показниками імовірності доставки, коефіцієнта впорядкованості, середнього часу відновлення та затримки доставки.

6. Провести експериментальну перевірку запропонованого методу в симуляційному середовищі та порівняти його з базовою архітектурою без резервного каналу.

Результати дослідження

Метод упорядкованої паралельної доставки подій

Загальна архітектура запропонованого методу доставки подій наведена на рис. 1.

У межах методу подія формалізується у вигляді кортежу E = (id, key, seq, payload), де id є глобально унікальним ідентифікатором події, key ∈ K – ключ впорядкування, який визначає належність події до певного логічного потоку, seq – монотонно зростаючий порядковий номер у межах відповідного ключа, а payload – корисне навантаження. Множина ключів K визначає рівень паралелізму в системі, оскільки події з різними ключами можуть оброблятися незалежно та одночасно.

Для забезпечення впорядкованості обробки в межах кожного ключа використовується механізм серіалізації, реалізований у координаторі Saga. Для кожного ключа key підтримується змінна стану last_seq[key], яка відображає номер останньої успішно обробленої події. Подія допускається до виконання лише у випадку, якщо її порядковий номер відповідає наступному очікуваному значенню, тобто виконується умова E.seq = last_seq[key] + 1. У випадку, коли подія надходить із більшим значенням порядкового номера, вона тимчасово буферизується до моменту отримання всіх попередніх подій. Такий підхід гарантує строгий порядок виконання в межах одного ключа при збереженні можливості паралельної обробки для різних ключів.

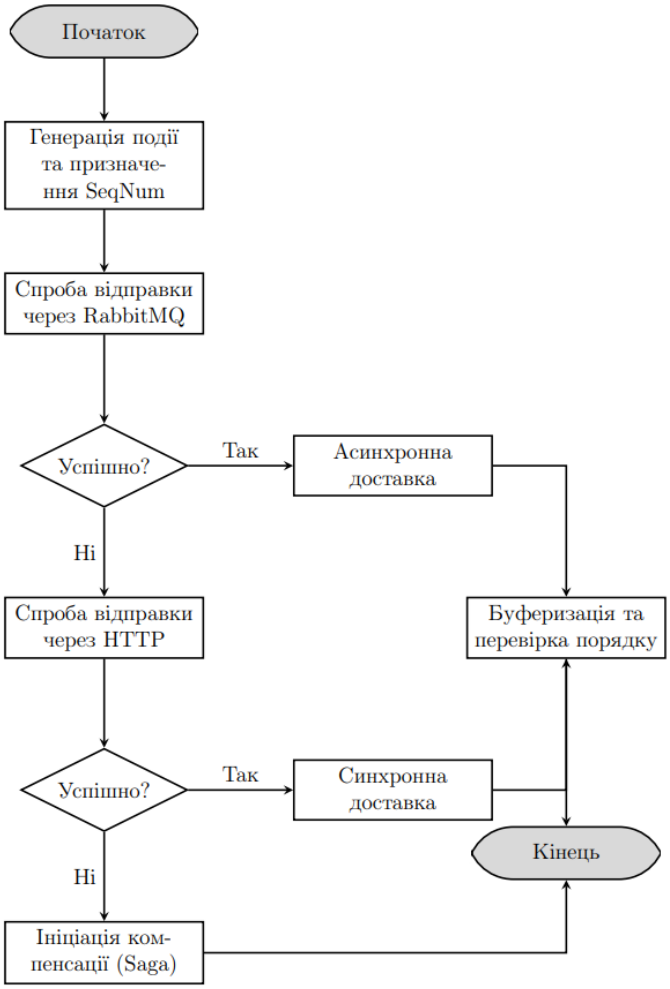

Рис. 1. Блок-схема функціонування архітектури методу

Важливою складовою методу є забезпечення ідемпотентності обробки подій. Повторно доставлені події, для яких виконується умова E.seq ≤ last_seq[key], ідентифікуються як дублікати та не передаються на виконання бізнес-логіки. Це дозволяє безпечно використовувати механізми повторної доставки повідомлень, які можуть виникати як на рівні брокера, так і при використанні резервного HTTP-каналу, без ризику некоректного дублювання операцій.

Алгоритм обробки подій передбачає послідовне виконання ряду перевірок незалежно від каналу, через який подія була отримана. Спочатку здійснюється перевірка на дублювання шляхом порівняння порядкового номера з останнім обробленим значенням для відповідного ключа. Далі перевіряється відповідність порядкового номера очікуваному значенню. У разі випередження подія буферизується, а у випадку коректної послідовності – передається на виконання з одночасним оновленням стану last_seq[key]. Лише після проходження всіх перевірок ініціюється виконання відповідного кроку Saga, що забезпечує узгодженість розподіленого процесу.

Завдяки використанню множини незалежних ключів досягається ефективна паралелізація обробки подій. Якщо кількість ключів становить m, система може одночасно обробляти до m незалежних потоків. Теоретична пропускна здатність у цьому випадку визначається як відношення кількості потоків до середнього часу обробки однієї події. При цьому механізм впорядкування накладає обмеження лише на обробку подій із однаковим ключем і не впливає на незалежні потоки, що дозволяє зберігати високу продуктивність системи.

Аналітична модель та оцінка ефективності

Для обґрунтування ефективності запропонованого методу упорядкованої паралельної доставки подій розроблено аналітичну модель функціонування мікросервісної системи, яка враховує особливості асинхронної взаємодії через брокер повідомлень RabbitMQ, наявність резервного HTTP-каналу, а також використання механізму Saga для керування розподіленими транзакціями.

У межах моделі система розглядається як сукупність взаємодіючих сервісів, що обмінюються подіями через основний та резервний канали доставки. Потік подій задається інтенсивністю λ, яка характеризує середню кількість подій, що надходять у систему за одиницю часу. Обробка подій виконується паралельно з урахуванням обмежень на впорядкованість, що визначається множиною ключів упорядкування K. Для кожного ключа забезпечується серіалізоване виконання відповідних підпроцесів.

Нехай T_{proc} – середній час обробки однієї події, T_{queue} – середній час перебування події в черзі брокера, T_{net} – затримка передачі повідомлення мережею. Тоді середній час обробки події в нормальному режимі може бути представлений як:

$$T_{total} = T_{queue} + T_{net} + T_{proc}. \quad (1)$$

У випадку відмови основного брокера система переходить до використання резервного HTTP-каналу. Нехай імовірність відмови брокера дорівнює p_{fail}, а середній час виявлення відмови – T_{detect}. Тоді середній час доставки події з урахуванням резервування визначається як:

$$T'_{total} = (1 - p_{fail}) \cdot T_{total} + p_{fail} \cdot (T_{detect} + T_{http} + T_{proc}), \quad (2)$$

де T_{http} – час доставки події через HTTP-канал.

Для оцінки надійності доставки введемо показник імовірності успішної доставки події $P_{deliver}$. У базовій системі без резервування він визначається як:

$$P_{base} = 1 - p_{loss}, \quad (3)$$

де p_{loss} – імовірність втрати повідомлення через відмову брокера або мережі.

У запропонованій архітектурі з резервним каналом:

$$P_{proposed} = 1 - p_{loss} \cdot p_{http_{loss}}, \quad (4)$$

де $p_{http_{loss}}$ – імовірність втрати повідомлення в резервному каналі. Таким чином, імовірність втрати подій суттєво зменшується за рахунок дублювання каналів доставки.

Для оцінки живучості системи використовується ймовірність завершення всіх кроків обробки події $P_{complete}$. У межах моделі Saga кожен процес складається з n послідовних кроків із імовірністю успішного виконання p_i для і-го кроку. Тоді:

$$P_{complete} = \prod(i = 1..n)p_i. \quad (5)$$

У разі виникнення помилки застосовуються компенсаційні транзакції, що дозволяє відновити систему до узгодженого стану, що забезпечує властивість eventual completion навіть при часткових відмовах, що є ключовою характеристикою liveness.

Оцінка середнього часу відновлення (Mean Time To Recovery, MTTR) виконується з урахуванням часу виявлення збою, часу перемикання на резервний канал і часу повторного виконання операцій. Для базової системи:

$$MTTR_{base} = T_{detect} + T_{restart}, \quad (6)$$

де $T_{restart}$ – час перезапуску сервісів і повторної обробки.

Для запропонованого підходу:

$$MTTR_{proposed} = T_{detect} + T_{switch} + T_{reprocess}, \quad (7)$$

де T_{switch} – час перемикання на HTTP-канал, $T_{reprocess}$ – час завершення незавершених кроків Saga. Оскільки перемикання відбувається без повного перезапуску системи, виконується нерівність:

$$MTTR_{proposed} < MTTR_{base}. \quad (8)$$

Продуктивність системи оцінюється через пропускну здатність, яка визначається як кількість оброблених подій за одиницю часу. У разі паралельної обробки для m незалежних потоків:

$$Throughput \approx m/T_proc. \quad (9)$$

Водночас введення механізму впорядкування накладає обмеження на паралельність для подій із однаковим ключем, що може призводити до зменшення максимальної пропускної здатності. Проте це компенсується підвищенням коректності обробки та зменшенням витрат на виправлення помилок.

Додатково вводиться показник коефіцієнта впорядкованості O, який характеризує частку подій, оброблених у правильній послідовності:

$$O = N_{ordered}/N_{total}, \quad (10)$$

де $N_{ordered}$ – кількість подій, оброблених без порушення порядку, N_{total} – загальна кількість подій. Для запропонованого методу O → 1 навіть за умов повторної доставки та часткових збоїв.

Аналітична модель демонструє, що інтеграція основного брокера повідомлень, резервного HTTP-каналу та механізму Saga дозволяє підвищити ймовірність доставки подій, забезпечити живучість системи та зменшити середній час відновлення. Отримані співвідношення створюють основу для подальшої експериментальної перевірки ефективності запропонованого методу.

Експериментальна перевірка

Для підтвердження адекватності аналітичної моделі та оцінки ефективності запропонованого методу було розроблено симуляційне середовище на мові Python 3.11. У межах моделі відтворено взаємодію основних компонентів мікросервісної системи: генератора подій, брокера повідомлень RabbitMQ, резервного HTTP-каналу та координатора Saga. Параметри симуляції були наближені до реальних умов функціонування розподілених систем: середня затримка брокера становила 12,4 ± 6,1 мс на рівні черги та 7,8 ± 3,2 мс на рівні мережі, затримка обробки подій – 23,7 ± 9,4 мс, затримка HTTP-каналу – 52,3 ± 18,6 мс. Кожен експериментальний сценарій виконувався 20 разів із обробкою 500 подій, використовуючи 10 ключів впорядкування та 4 паралельних потоки.

У рамках дослідження було визначено чотири сценарії тестування: нормальна робота базової системи без резервного каналу; нормальна робота із застосуванням запропонованого методу; функціонування базової системи за умов періодичних відмов брокера; а також робота запропонованої системи за аналогічних умов. Ін'єкція збоїв реалізовувалась шляхом примусового вимкнення брокера через кожні 45–60 подій із подальшим відновленням після завершення поточного циклу обробки.

Отримані результати свідчать про суттєве підвищення надійності доставки подій у запропонованому методі. Імовірність успішної доставки становить 99,8–99,9% порівняно з 95,3–95,5% у базовій системі. Підвищення показника пояснюється використанням резервного HTTP-каналу, через який у середньому доставляється близько 4,4–4,7% подій залежно від сценарію.

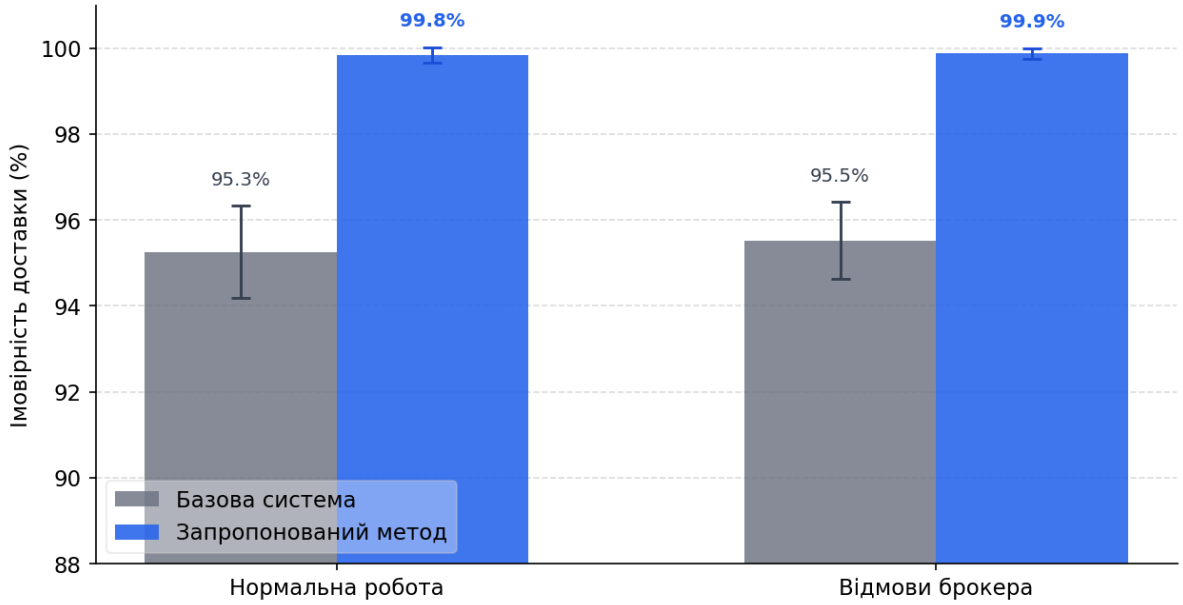

Рис. 2. Імовірність успішної доставки подій $P_{deliver}$

Втрати подій у запропонованій системі становлять не більше 0,12–0,16%, що узгоджується з теоретичною оцінкою комбінованої ймовірності відмови каналів. Коефіцієнт впорядкованості також демонструє суттєве покращення: значення показника O досягає 99,65–99,74% у порівнянні з приблизно 93,7% у базовій системі. Незначні відхилення від ідеального порядку пояснюються граничними ефектами синхронізації подій, що надходять із різних каналів.

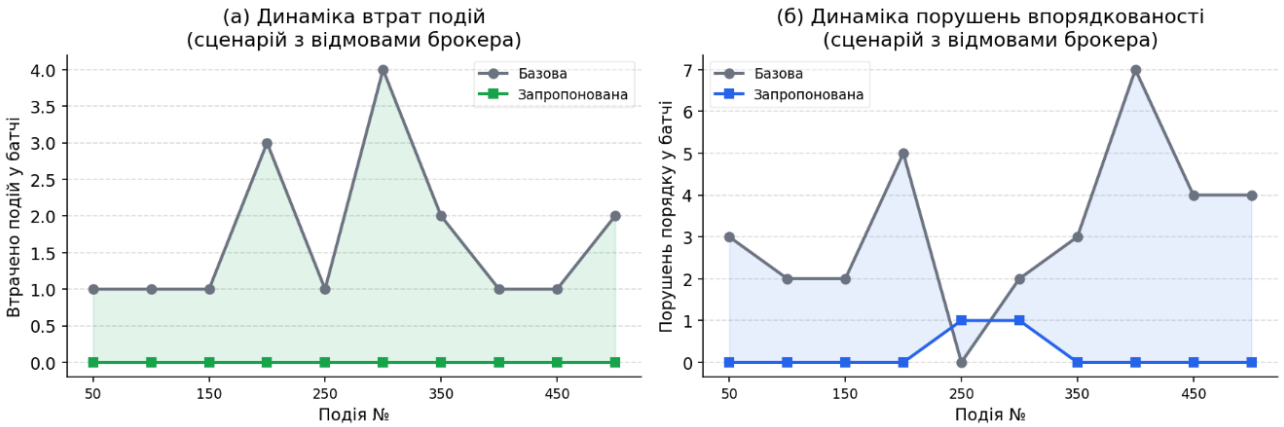

Рис. 3. Динаміка ключових метрик у часі (по батчах 50 подій)

Особливо значущим є скорочення середнього часу відновлення системи при відмовах брокера. У базовій архітектурі MTTR становить близько 957 мс, тоді як у запропонованому методі цей показник зменшується до 263 мс, що відповідає скороченню на 72,5%. Це досягається завдяки відсутності необхідності повного перезапуску системи та використанню механізму швидкого перемикання на резервний канал.

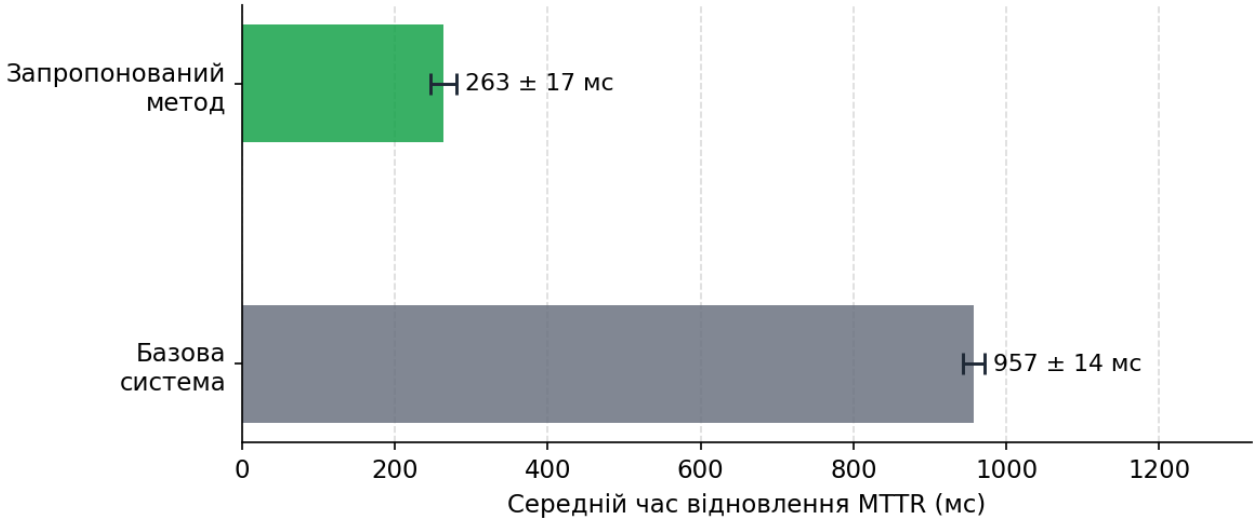

Рис. 4. Порівняння MTTR при відмовах брокера

При цьому середня затримка доставки подій збільшується незначно – приблизно на 1,6–1,8 мс, що становить близько 3,7% від базового значення. Це свідчить про прийнятний рівень накладних витрат на забезпечення відмовостійкості.

Результати симуляційної перевірки підтверджують ефективність запропонованого методу та його відповідність аналітичній моделі, демонструючи підвищення надійності доставки подій, збереження їх впорядкованості та суттєве скорочення часу відновлення системи.

Висновки та перспективи подальших досліджень

У роботі запропоновано метод упорядкованої паралельної доставки подій у мікросервісних системах, який базується на інтеграції асинхронного обміну повідомленнями через RabbitMQ, резервного HTTP-каналу та механізму керування розподіленими транзакціями на основі патерну Saga. Розроблено архітектуру системи, що забезпечує безперервність доставки подій, їх впорядковану обробку та підвищену відмовостійкість.

Розроблено аналітичну модель, яка описує вплив резервування каналів та використання механізму Saga на ключові характеристЗики системи, зокрема імовірність успішної доставки подій та середній час відновлення. Теоретично обґрунтовано, що запропонований підхід забезпечує підвищення надійності доставки та зменшення MTTR порівняно з базовими рішеннями.

Результати експериментальної перевірки підтвердили ефективність методу. Показано, що імовірність успішної доставки подій зростає до 99,8–99,9%, коефіцієнт впорядкованості наближається до 1, а середній час відновлення системи скорочується на понад 70% у порівнянні з традиційною архітектурою. При цьому додаткові накладні витрати на забезпечення відмовостійкості є незначними.

Отримані результати свідчать про доцільність використання запропонованого методу в корпоративних та високонавантажених мікросервісних системах, де критичними є вимоги до надійності, доступності та коректності обробки подій. Подальші дослідження можуть бути спрямовані на оптимізацію механізмів впорядкування, зменшення накладних витрат та адаптацію методу до різних типів брокерів повідомлень і доменних застосувань.

Декларація про штучний інтелект

Під час підготовки статті інструменти штучного інтелекту не використовувалися. Автори підтверджують самостійність виконання дослідження, достовірність поданих результатів і відповідальність за зміст публікації.

Конфлікт інтересів

Автор заявляє про відсутність конфлікту інтересів та підтверджує, що під час підготовки цієї роботи не існувало жодних комерційних, фінансових чи інших взаємовідносин, які могли б бути розцінені як такі, що здатні вплинути на результати дослідження або їх інтерпретацію. Робота виконана відповідно до принципів академічної доброчесності, етичних норм проведення наукових досліджень та вимог редакційної політики щодо запобігання конфлікту інтересів.

Список використаної літератури

1. Lee, W.-T., Song, P.-Y., Tsai, M.-K., et al. (2023). A high availability microservices architecture implementation using Saga and backup mechanism. Proceedings of the International Conference on Dependable Systems and Their Applications (DSA). https://doi.org/10.1109/DSA59317.2023.00063

2. Ramisetty, G. (2025). Event-driven microservices for ultra-low latency cloud workflows. Global Journal of Computer Science and Technology, 25(B1), 1–8. https://doi.org/10.34257/GJCSTBVOL25IS1PG1

3. Chejarla, J. R. (2025). Event-driven cloud-native order management system architecture. Zenodo Repository. https://doi.org/10.5281/zenodo.16408024

4. Garcia-Molina, H., & Salem, K. (1987). Sagas. ACM SIGMOD Record, 16(3), 249–259. https://doi.org/10.1145/38714.38742

5. Kassetty, N., & Jain, A. (2025). 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 Modern Engineering and Emerging Technology, 13(3), 204–215. https://doi.org/10.63345/ijrmeet.org.v13.i3.12

6. Bhupatiraju, R. V. (2025). Event-driven architecture for payment failover and redundancy: A framework for high-availability financial transaction processing. European Modern Studies Journal, 9(5), 815–830. https://doi.org/10.59573/emsj.9(5).2025.75

7. Бугаєва, І. Г. (2023). Реалізація saga з використанням шаблону outbox. Таврійський науковий вісник. Технічні науки, (4). https://doi.org/10.32782/tnv-tech.2023.4.4

8. Daraghmi, E. Y., Zhang, C.-P., & Yuan, S.-M. (2022). Enhancing Saga pattern for distributed transactions within a microservices architecture. Applied Sciences, 12(12), 6242. https://doi.org/10.3390/app12126242

9. Kumar, R. (2025). Event-driven architectures for real-time data processing: A deep dive into system design and optimization. Zenodo Repository. https://doi.org/10.5281/zenodo.15026989

10. Purella, S. (2025). Microservices and event-driven architectures in high-volume financial systems. Zenodo Repository. https://doi.org/10.5281/zenodo.16150784

A. Kolodiuk

METHOD FOR ORDERED PARALLEL EVENT DELIVERY IN MICROSERVICE SYSTEMS USING RABBITMQ, HTTP BACKUP CHANNELS, AND SAGA PATTERN TO INCREASE VIABILITY AND REDUCE MTTR

The article investigates the problem of reliable, ordered, and fault-tolerant event delivery in microservice systems based on asynchronous message exchange. The relevance of the study is determined by the fact that, in distributed software systems, violation of event ordering, message duplication, or broker unavailability may lead to an inconsistent state of business processes, increased recovery time, and reduced system liveness. The purpose of the research is to develop a method for ordered parallel event delivery that combines the RabbitMQ message broker, a backup HTTP channel, and the Saga pattern for managing distributed transactions.

The introduction substantiates the need for an integrated approach to event delivery in microservice architecture, taking into account scalability, availability, and correctness requirements. The literature review analyzes existing approaches to event-driven architecture, message brokers, retry mechanisms, channel redundancy, and the Saga pattern. It is shown that most existing solutions consider these mechanisms separately, while their integration for simultaneous ordering, liveness assurance, and MTTR reduction remains insufficiently studied.

The methodological part proposes a system architecture that includes event producers, the RabbitMQ broker, microservice consumers, a Saga coordinator, an ordering control module, an event buffer, an idempotent processing mechanism, and a backup HTTP route. Each event is formalized as a structure containing a unique identifier, an ordering key, a sequence number, and a payload. For each ordering key, sequential event processing is guaranteed, while events with different keys can be processed in parallel. This approach makes it possible to combine high throughput with control over the logical order of execution.

The analytical modeling section presents the main indicators used to evaluate the effectiveness of the proposed method: successful delivery probability, ordering coefficient, mean time to recovery, and average delivery latency. The experimental validation was carried out in a Python 3.11 simulation environment under normal operation and broker failure scenarios. The obtained results show that the proposed method increases the probability of successful event delivery to 99.8–99.9%, provides an ordering coefficient of approximately 99.7%, and reduces MTTR from 957 ms to 263 ms, which corresponds to a reduction of more than 70%. At the same time, the increase in average delivery latency remains minor and amounts to approximately 3.7%.

The practical value of the results lies in the possibility of applying the proposed method to the development of enterprise and high-load microservice systems where guaranteed event delivery, preservation of event sequence, consistency of distributed transactions, and fast recovery after partial failures are critical requirements.

Keywords: microservice architecture, event-driven systems, RabbitMQ, Saga, backup HTTP channel, ordered event delivery, fault tolerance, MTTR.

Надійшла до редакції: 06.03.2026
Прийнята до друку: 24.04.2026
Опубліковано: 29.06.2026