КОМП'ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

УДК 004.42:004.75
DOI https://doi.org/10.32782/tnv-tech.2023.2.1

УПРАВЛІННЯ ТРАНЗАКЦІЯМИ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Бугаєва І. Г. *– кандидат технічних наук,*
доцент кафедри технічної кібернетики й інформаційних технологій
імені професора Р. В. Меркта
Одеського національного морського університету
ORCID ID: 0000-0002-2839-9266

Розум М. В. *– кандидат фізико-математичних наук,*
доцент кафедри технічної кібернетики й інформаційних технологій
імені професора Р. В. Меркта
Одеського національного морського університету
ORCID ID: 0000-0002-9459-8044

Ларін Д. Г. *– кандидат технічних наук,*
доцент кафедри технічної кібернетики й інформаційних технологій
імені професора Р. В. Меркта
Одеського національного морського університету
ORCID ID: 0009-0006-4882-0683

Ткаченко М. Г. *– кандидат фізико-математичних наук,*
доцент кафедри технічної кібернетики й інформаційних технологій
імені професора Р. В. Меркта
Одеського національного морського університету
ORCID ID: 0000-0002-3166-2627

Використання мікросервісної архітектури є популярним підходом до розробки сучасних програмних систем. Мікросервіси застосовуються у великих Інтернет-проєктах, які обслуговують значну кількість користувачів. Додатки з мікросервісною архітектурою мають багато переваг, таких, як висока доступність, відмовостійкість, гнучкість, масштабованість. Поряд з перевагами такого підходу існують і недоліки. У розподіленій системі, що складається з багатьох служб, управління транзакціями, що охоплюють

кілька сервісів, є складним завданням. Оскільки мікросервіси слабко пов'язані між собою, а дані, що використовуються розподіленою транзакцією, зберігаються на різних вузлах у мережі, існує проблема узгодженості даних. Метою статті є дослідження способів управління транзакціями в мікросервісах та забезпечення узгодженості даних, виявлення переваг та недоліків застосування існуючих шаблонів проєктування. У статті обговорюється використання протоколу двофазної фіксації 2PC і шаблону проєктування Saga. 2PC – це надійний протокол узгодженості даних для мікросервісів. Але це форма синхронної взаємодії, в результаті якої служби зв'язуються під час виконання, що значно впливає на доступність додатку. Якщо висока доступність застосування є пріоритетною вимогою, то як основу для розробки краще вибрати шаблон проєктування Saga. Він вирішує проблему виконання транзакцій, що охоплюють декілька сервісів. Для забезпечення узгодженості даних Saga використовується у поєднанні з патерном Event sourcing або Transactional Outbox. Використання шаблону саги на основі хореографії забезпечує вищу продуктивність сервісів порівняно з оркестрацією. Його краще використовувати для сценаріїв, у яких час відповіді критичний, кількість сервісів, що приймають участь у розподіленій транзакції, невелика. Шаблон на основі оркестрації повільніший за хореографію, але він є кращим вибором для обробки складних сценаріїв транзакцій.

Ключові слова: *мікросервіси, мікросервісна архітектура, розподілені транзакції, узгодженість даних, двофазна фіксація, шаблон проєктування Saga.*

Buhaieva I. H., Rozum M. V., Larin D. H., Tkachenko M. H. Transaction management in microservice architecture

The use of microservice architecture is a popular approach to the development of modern software systems. Microservices are used in large Internet projects serving a large number of users. Applications with microservice architecture have many advantages such as high availability, fault tolerance, flexibility, scalability. Along with the advantages, this approach also has disadvantages. In a distributed system consisting of many services, managing transactions spanning multiple services is a complex task. Since microservices are loosely coupled and the data used by a distributed transaction is stored on different nodes in the network, there is a problem of data consistency. The purpose of the article is to explore ways to manage transactions in microservices and ensure data consistency, identifying the advantages and disadvantages of applying existing design patterns. This article discusses the use of the 2PC two-phase commit protocol and the Saga design pattern. 2PC is a robust data consistency protocol for microservices. But it is a form of synchronous communication that results in microservices being coupled at runtime, which has a significant impact on the availability of the application. If high availability of the application is a priority requirement, then it is better to choose the Saga design pattern as a basis for development. It solves the problem of executing transactions spanning multiple services. To ensure data consistency, Saga is used in conjunction with the Event sourcing or Transactional Outbox pattern. Using the choreography-based saga pattern provides better service performance compared to orchestration. It is better to use it for scenarios in which response time is critical, the number of services involved in a distributed transaction is small. The orchestration-based pattern is slower than choreography, but is the best choice for handling complex transaction scenarios.

Key words: *microservices, microservice architecture, distributed transactions, data consistency, two-phase commit, Saga design pattern.*

Вступ. Додаток з мікросервісною архітектурою є набором сервісів, що незалежно розгортаються, слабо пов'язані, організовані навколо потреб бізнесу [1; 2]. Екземпляри служб зазвичай запускаються на різних комп'ютерах, спілкуються один з одним шляхом обміну повідомленнями, використовуючи різні механізми, які забезпечують міжпроцесну взаємодію [3]. Хоча така розподілена архітектура має багато переваг (висока доступність, відмовостійкість, гнучкість, масштабованість), існують проблеми, пов'язані з управлінням транзакціями, що охоплюють кілька сервісів.

У традиційному монолітному додатку використання спільної реляційної бази даних забезпечує узгодженість даних за допомогою транзакцій, які задовільняють чотирьом вимогам ACID (атомарність, узгодженість, ізольованість, стійкість) – стандартному набору властивостей, що гарантують надійність транзакцій.

В архітектурі мікросервісів служби мають власні сховища даних, які можуть бути реалізовані за допомогою різних технологій. Бізнес-транзакції можуть складатися з кількох локальних транзакцій у межах окремих служб. При виконанні такої розподіленої транзакції виникає проблема дотримання вимоги атомарності, яка гарантує, що жодна транзакція не буде зафіксована в системі частково. Будуть або завершені всі її локальні транзакції, або не виконано жодної. Взаємодія сервісів здійснюється через мережу, яка є ненадійною. У будь-який момент здійснення розподіленої транзакції може статися збій у роботі окремої служби. Необхідно це передбачити і мати можливість скасувати зміни даних, які вже було зафіксовано в сховищах даних інших сервісів, що приймають участь у цій транзакції.

Метою статті є дослідження способів управління транзакціями у додатках з мікросервісною архітектурою та забезпечення узгодженості даних в мікросервісах, виявлення переваг та недоліків різних підходів.

Аналіз досліджень і публікацій. Дослідницька робота [3] була спрямована на вирішення таких питань: з'ясування проблем з розподіленими транзакціями в ізольованих базах даних NoSQL у мікросервісній архітектурі; проведення порівняльних тестів продуктивності додатків з застосуванням шаблону Saga на основі хореографії та оркестрації; вироблення рекомендацій щодо реалізації шаблону саги для різних варіантів використання.

Автори провели тести продуктивності застосувань, розроблених з використанням фреймворка Java Spring Boot, бази даних MongoDB та брокеру повідомлень ActiveMQ. Додатки було реалізовано за допомогою шаблону проєктування Saga на основі хореографії та оркестрації.

Результати тестів показали, що використання шаблону на основі хореографії забезпечує більшу продуктивність додатку порівняно з іншим підходом. Хореографія подій може добре підійти для сценаріїв, де кількість викликів мікросервісу обмежена, а час відповіді критичний. Шаблон на основі хореографії є рекомендованим підходом, коли кількість мікросервісів, що беруть участь у розподіленій транзакції, невелика. Додатки на основі оркестрації більш повільні, але цей шаблон є кращим у випадку, коли сценарії транзакцій є складними.

У роботі [4] досліджувалося, яку реалізацію шаблону саги, хореографію чи оркестрацію, слід використовувати за різних сценаріїв. Було розроблено дві моделі програми на основі мікросервісів з використанням Java Spring Boot, бази даних MySQL та брокера повідомлень Kafka. Служби розгорталися на Google Cloud Platform з використанням інструменту для розгортання, масштабування та управління контейнерними програмами Kubernetes. Програма JMeter використовувалася для проведення навантажувального тестування та порівняння продуктивності шаблонів хореографії та оркестрації. Дані збиралися шляхом моделювання транзакцій за участю кількох сервісів, і кожен параметр виміру записувався від початку транзакції до завершення.

Моделювання виконувалося з використанням різної кількості сервісів, задіяних у сазі (2, 4, 6 та 8). Було проведено експерименти щодо збільшення кількості користувачів, які проводять транзакції одночасно, починаючи з одного користувача, потім 100, 200, далі з кроком 100 до 1000 користувачів. Використовувалася також різна кількість екземплярів служб для порівняльного аналізу продуктивності двох моделей.

Продуктивність додатків оцінювалася шляхом вимірювання наступних параметрів – час відгуку, пропускна здатність та завантаження процесора. За результатами експериментів встановлено, що зі збільшенням кількості сервісів

та користувачів, які проводять транзакції одночасно, час відгуку обох моделей збільшується. Підхід «хореографія» має кращий час відгуку, ніж «оркестрація», і додавання кількості сервісів та користувачів більш впливає на цей показник при використанні шаблону оркестрації. Також хореографія сервісів забезпечує кращу пропускну здатність порівняно з оркестрацією, менше завантажує процесор.

При збільшенні кількості сервісів, що беруть участь у сазі, автори рекомендують використовувати шаблон Saga на основі оркестрації.

Виклад основного матеріалу. Розглянемо використання різних шаблонів проєктування для реалізації розподілених транзакцій у мікросервісах на прикладі замовлення подорожі, яке включає бронювання готелю та авіаквитків для перельоту. Три мікросервіси, кожен з яких має своє сховище даних, забезпечують цей функціонал. Сервіс замовлення подорожі надає API для взаємодії з користувачами. Після отримання запиту від клієнта ця служба має виконати запити до двох інших сервісів (рис. 1).

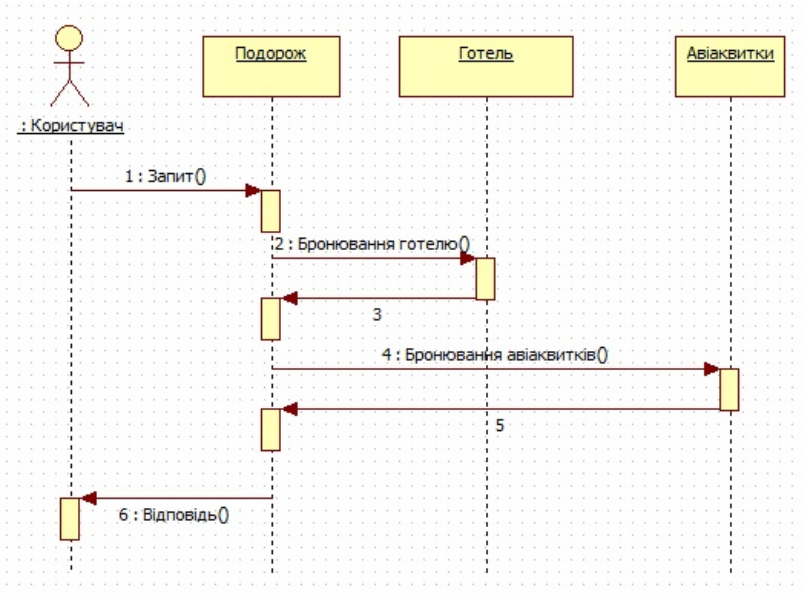

Рис. 1. Виконання запиту на замовлення подорожі

Бронювання готелю та авіаквитків повинно виконуватися як єдина транзакція. Система на основі мікросервісів за замовчуванням не має центрального координатора транзакцій. У наведеному вище прикладі, якщо у сервісі «Авіабілети» відбудеться збій, потрібен механізм для скасування змін даних, які відбулися в базі даних сервісу «Готель».

Розподілену транзакцію можна реалізувати двома способами: за допомогою протоколу двофазної фіксації (2PC) або шаблону проєктування Saga.

Протокол 2PC гарантує, що при завершенні транзакції всі зміни, зроблені над усіма ресурсами, або повністю фіксуються, або повністю відкочуються [5]. У цьому шаблоні розподілена транзакція складається з двох кроків: етапу підготовки та етапу фіксації або відкату. Одним з учасників транзакції є координатор транзакції. На

першому етапі всі учасники транзакції готуються до фіксації та повідомляють координатору про те, що вони готові завершити транзакцію. На другому етапі координатор транзакції надсилає всім учасникам команду фіксації або відкату.

На рис. 2 наведено діаграму послідовності виконання розподіленої транзакції з використанням двофазної фіксації для випадку, коли на першому етапі обидва сервіси були готові завершити транзакцію. І всю транзакцію було завершено з успішним результатом.

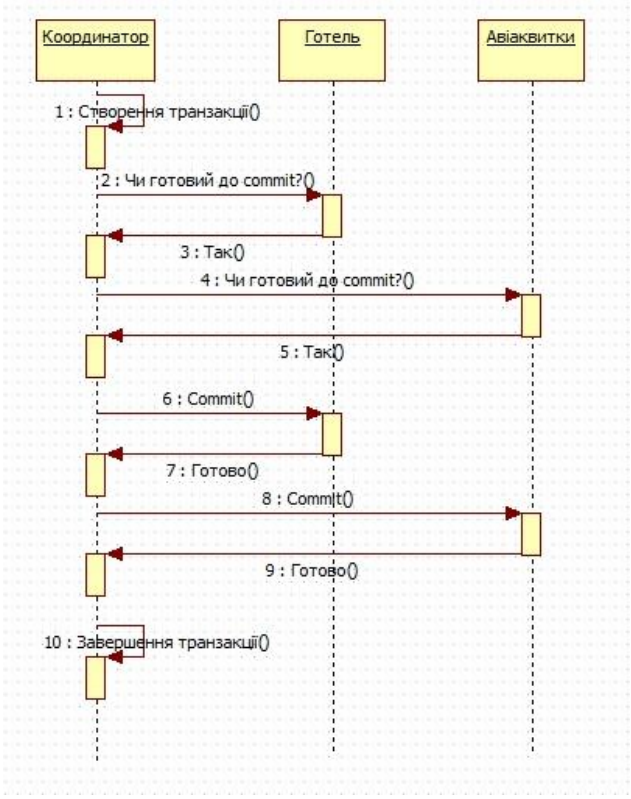

Рис. 2. Успішне виконання розподіленої транзакції з використанням двофазної фіксації

На рис. 3 представлений інший можливий варіант розвитку подій, коли на першому етапі транзакції сервіс «Авіаквитки» підтвердив, що він готовий виконати фіксацію змін, а від сервісу «Готель» було отримано негативну відповідь. Після цього координатор транзакції дав команду сервісу «Авіаквитки» виконати відкат.

Перевагою використання протоколу 2PC є те, що це дуже надійний протокол узгодженості. По-перше, фази підготовки та фіксації гарантують, що транзакція є атомарною. По-друге, 2PC дозволяє ізолювати читання-запис. Це означає, що зміни в полі даних не видно, доки координатор не зафіксує зміни.

Недоліки такого способу реалізації розподіленої транзакції:

– це форма синхронного зв’язку, що призводить до зв’язування мікросервісів під час виконання, що значно впливає на доступність додатку; якщо один з учасників виходить з ладу, система в цілому стає недоступною;

– вузол координатора транзакції може стати єдиною точкою відмови;
– можливі взаємоблокування між транзакціями;
– протокол 2PC не підтримується базами даних NoSQL.

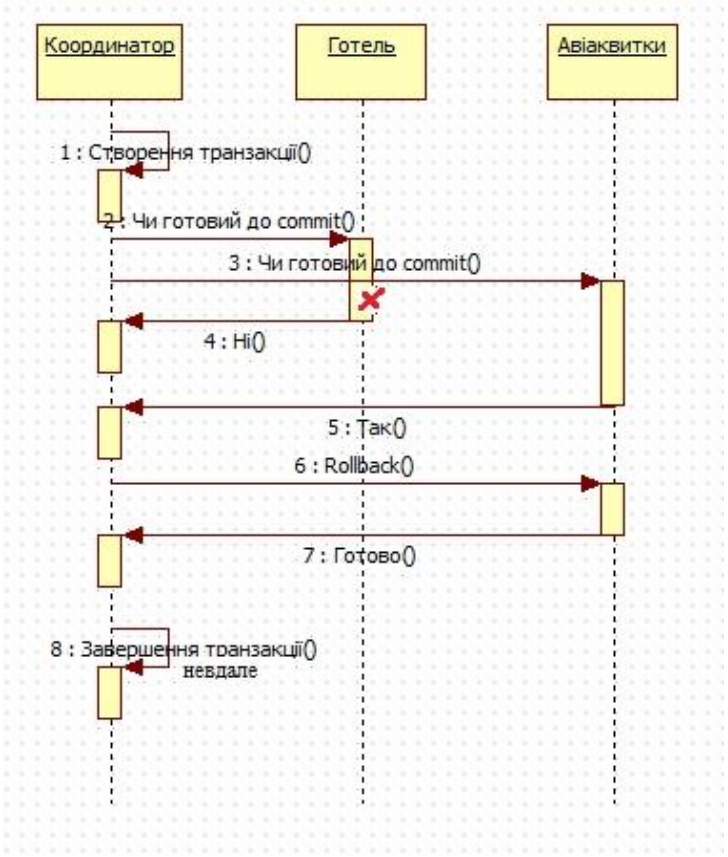

Рис. 3. Невдале виконання розподіленої транзакції з використанням двофазної фіксації

Інший підхід для управління транзакціями у мікросервісах полягає у використанні шаблону проєктування Saga [7]. Сага – це послідовність локальних транзакцій у кожній з служб, що беруть у ній участь. Кожна така транзакція оновлює базу даних сервісу та публікує повідомлення, щоб ініціювати наступну локальну транзакцію. Якщо якась з них зазнає невдачі, сага виконує серію компенсуючих транзакцій, які скасовують зміни, внесені попередніми локальними транзакціями. Кожен крок саги, за яким слідує крок, який може завершитися невдало, повинен мати відповідну компенсуючу транзакцію.

Є два типи саг: хореографія та оркестрація. Суть шаблону хореографії полягає в тому, що кожен мікросервіс приймає рішення про перебіг бізнес-транзакції, не покладаючись на центральну точку управління. Кожна локальна транзакція у службі публікує події, які запускають локальні транзакції на інших сервісах.

Нижче наведено діаграми, що ілюструють послідовність виконання локальних транзакцій у сервісах «Готель» та «Авіабілети», що беруть участь у сазі на основі хореографії. На рис. 4 представлено діаграму послідовності для варіанту успішного виконання транзакції, що охоплює сервіси «Готель» та «Авіаквитки».

При оновленні локальної бази даних у службі «Готель» публікується повідомлення, яке ініціює виконання локальної транзакції у наступному сервісі. Якщо сервісу «Авіаквитки» не вдалося виконати необхідні операції, то сервіс «Готель» повинен запустити компенсуючу транзакцію (рис. 5).

У шаблоні Saga на основі оркестрації мікросервіси, що беруть участь, не взаємодіють безпосередньо один з одним. Натомість є сервіс-оркестратор, який відповідає за управління загальним станом транзакції. На рис. 6 наведено діаграму послідовності успішного виконання розподіленої транзакції.

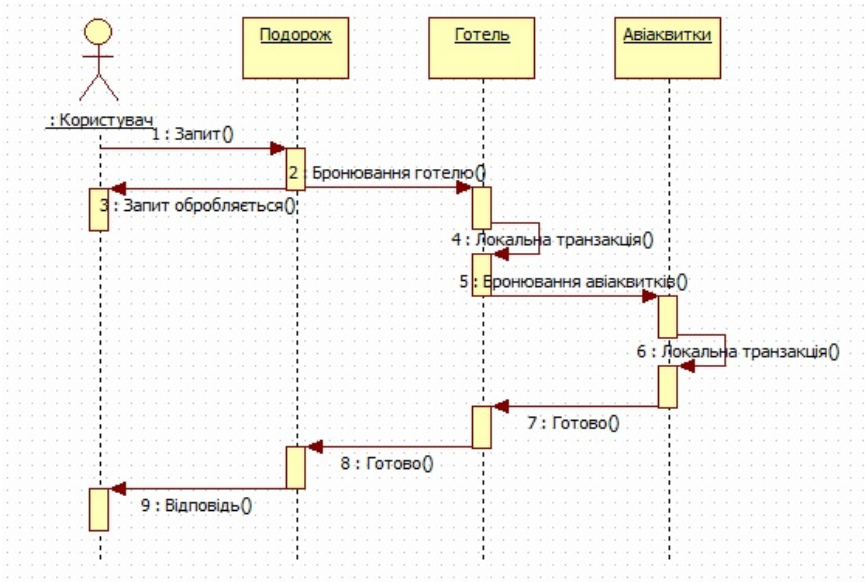

Рис. 4. Успішне виконання розподіленої транзакції з використанням шаблону Saga на основі хореографії

У разі збою при виконанні локальної транзакції у сервісі «Авіаквитки» оркестратор відправляє команду сервісу «Готель» виконати компенсуючу транзакцію (рис. 7).

З використанням шаблону Saga виникає проблема забезпечення узгодженості даних у сервісах. У межах своєї бізнес-логіки мікросервіси часто оновлюють свої локальні сховища даних. Одночасно їм потрібно повідомляти про зміни, що відбулися, іншим службам. Без використання протоколу 2PC виконання цих двох операцій у межах однієї транзакції не гарантовано. Сервіс може вийти з ладу на будь-якому з цих етапів. Але база даних або брокер повідомлень, який використовується для публікування повідомлень, можуть не підтримувати протокол двофазної фіксації. Для вирішення проблеми атомарного оновлення стану об’єкта та надсилання повідомлень повинен застосовуватися один з наступних шаблонів – Event sourcing або Transactional Outbox.

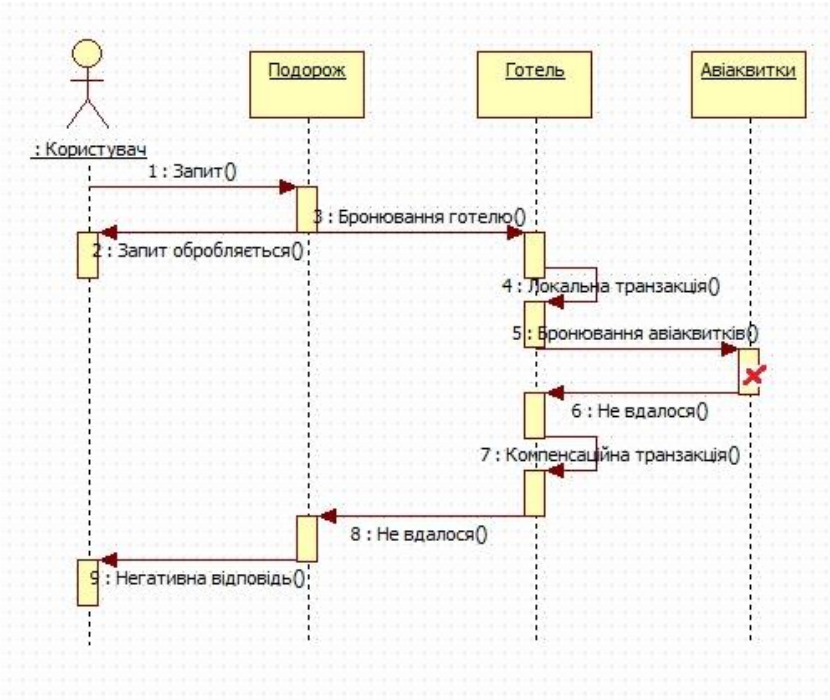

Рис. 5. Невдале виконання розподіленої транзакції з використанням шаблону Saga на основі хореографії

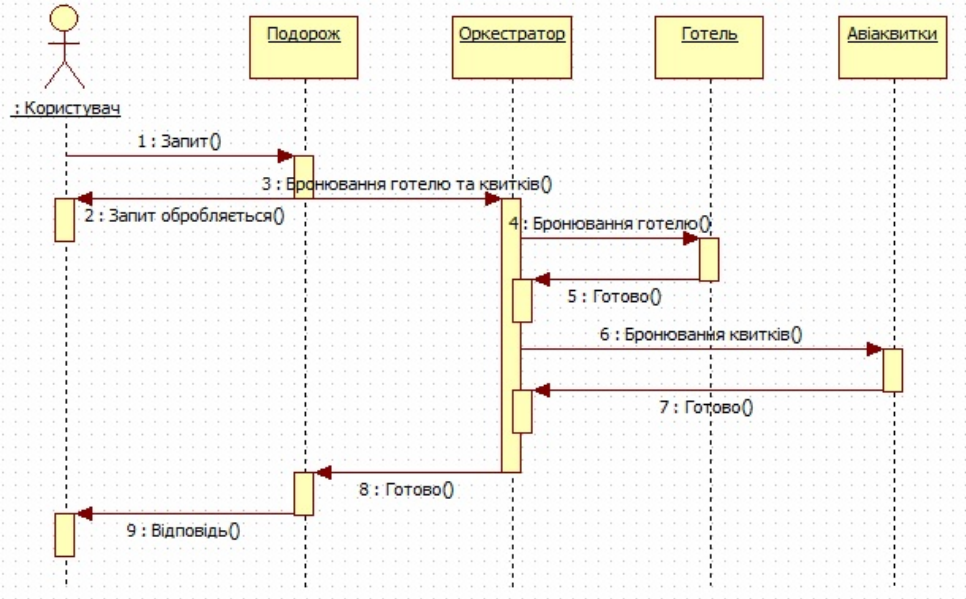

Рис. 6. Успішне виконання розподіленої транзакції з використанням шаблону Saga на основі оркестрації

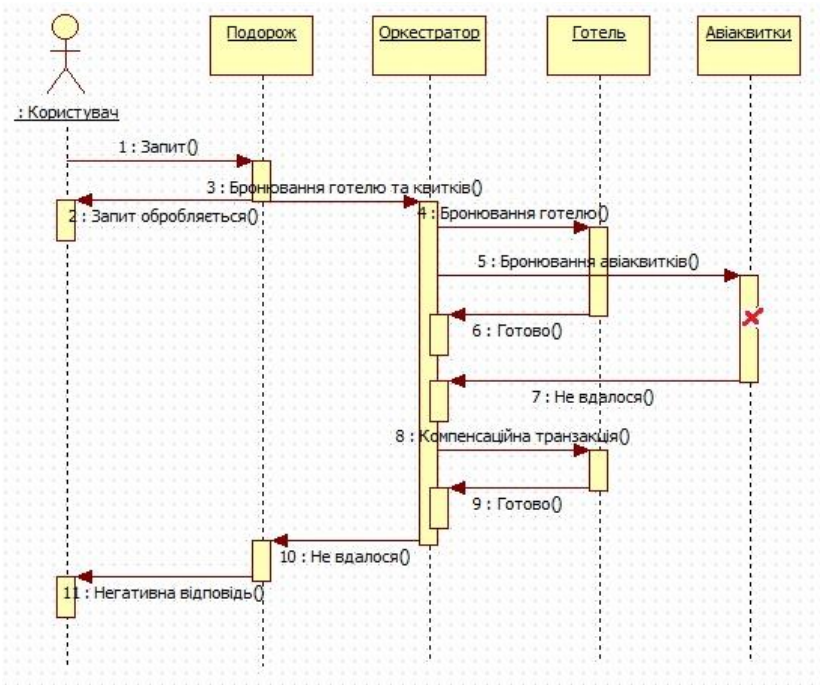

Рис. 7. Невдале виконання розподіленої транзакції з використанням шаблону Saga на основі оркестрації

У разі використання шаблону Outbox мікросервіс, що використовує реляційну базу даних, виконує додавання повідомлення в таблицю вихідних повідомлень Outbox як частину локальної транзакції під час оновлення бази даних. Служба, яка використовує базу даних NoSQL, додає повідомлення до атрибута запису, що оновлюється. Потім окремий процес ретрансляції повідомлень зчитує їх з таблиці Outbox та публікує у брокер повідомлень, який повинен бути опитаний наступною службою, яка є учасником саги. Шаблон Outbox використовується разом з Debezium – сервісом для захоплення змін у базах даних (Change Data Capture) та надсилання їх на обробку іншим системам. Debezium заснований на Apache Kafka, що дозволяє використовувати переваги цієї платформи – стійкість до відмов, масштабованість та надійна обробка великих обсягів даних у реальному часі.

На рис. 8 схематично представлений процес запису даних у рамках однієї локальної транзакції у таблиці Hotels та Outbox бази даних сервісу «Готель». Сервіс Debezium зчитує дані з таблиці Outbox та відправляє їх до брокеру повідомлень Kafka. Сервіс «Авіаквитки» опитує брокер і з появою нових повідомлень оновлює свою базу даних.

Шаблон Transactional Outbox має такі переваги:

– повідомлення гарантовано надсилаються тоді і лише тоді, коли транзакція бази даних фіксується;

– повідомлення надсилаються брокеру повідомлень у порядку, в якому вони були надіслані додатком.

Інший спосіб вирішення проблеми атомарного оновлення стану об’єкта та публікації подій – використання шаблону Event sourcing. Він зберігає стан

бізнес-об'єкта як послідовність подій. Програма може відновити стан об'єкта, відтворюючи події. Події зберігаються у сховищі, яке є базою даних і поводиться як брокер повідомлень (рис. 9). Воно надає API, який дозволяє службам підписуватися на події. Коли сервіс зберігає подію у такому сховищі, вона доставляється всім зацікавленим передплатникам.

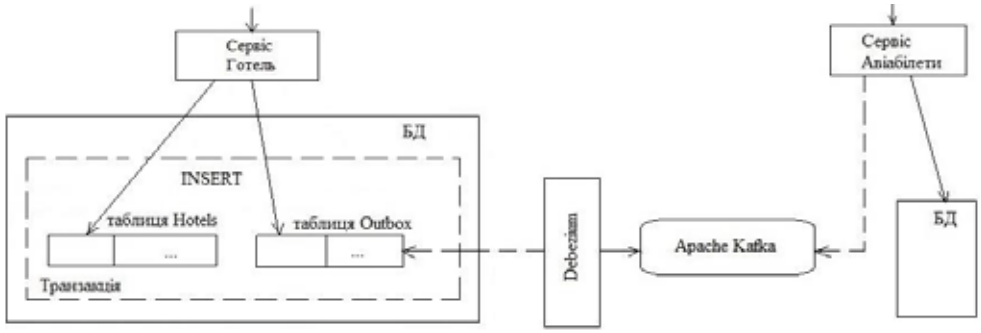

Рис. 8. Використання шаблону Outbox для забезпечення узгодженості даних у сервісах

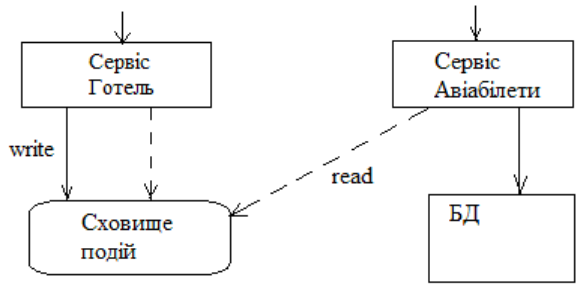

Рис. 9. Використання шаблону Event sourcing для забезпечення узгодженості даних

Шаблон Event sourcing дозволяє надійно публікувати події при кожній зміні стану бізнес-об'єкта.

Висновки. У статті розглянуто два основні підходи в реалізації управління розподіленими транзакціями у додатках з мікросервісною архітектурою: використання протоколу двофазної фіксації 2PC та шаблону проєктування Saga.

2PC є надійним протоколом узгодженості даних у мікросервісах. Але це форма синхронного зв'язку, що призводить до зв'язування мікросервісів під час виконання, що значно впливає на доступність додатку.

Якщо пріоритетною вимогою є висока доступність застосування, то краще обрати як основу для розробки шаблон проєктування Saga. Він вирішує проблему виконання транзакцій, що охоплюють декілька сервісів. Для забезпечення узгодженості даних Saga використовується разом з одним з шаблонів – Event sourcing або Transactional Outbox. До переваг шаблону Saga належить також підтримка довготривалих транзакцій. Під час виконання транзакції інші служби не блокуються, якщо одна з них працює протягом тривалого часу. До недоліків слід

віднести складнішу модель програмування. Шаблон Saga важко налагоджувати, і складність зростає зі збільшенням числа мікросервісів, що беруть участь у розподіленій транзакції.

Використання шаблону саги на основі хореографії забезпечує більш високу продуктивність сервісів порівняно з оркестрацією. Його краще застосовувати для сценаріїв, в яких час відповіді критичний, кількість сервісів, що беруть участь у розподіленій транзакції, невелика. Шаблон на основі оркестрації повільніший за хореографію, але він є найкращим вибором для обробки складних сценаріїв транзакцій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Richardson Ch. Microservices patterns. Manning Publications, 2019. 520 p.
2. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015. 280 p.
3. Бугаєва І.Г., Розум М.В. Реалізація міжпроцесної взаємодії в мікросервісній архітектурі. *Вісник Одеського національного морського університету: Зб. наук. праць.* 2022. № 1(67). С. 81–89.
4. Rudrabhatla Ch. K. Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture. *International Journal of Advanced Computer Science and Applications*. 2018. Vol. 9, No. 8.
5. Kristianto H., Zahra A. Performance analysis of choreography and orchestration in microservices architecture. *Journal of Theoretical and Applied Information Technology*. 2021. Vol. 99, No.18.
6. Harding R., Aken D. V., Pavlo A., Stonebraker M. An Evaluation of Distributed Concurrency Control. *Proceedings of the VLDB Endowment*. 2017. Vol. 10, No. 5.
7. Garcia-Molina H., Salem K. SAGAS. *Proceedings of the 1987 ACM SIGMOD international conference on Management of data*. 1987. P. 249–259.

REFERENCES:

1. Richardson, Ch. (2019). Microservices Patterns. Manning Publications, 520.
2. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 280.
3. Buhaieva, I.G., Rozum, M.V. (2022). Realizatsiia mizhprotsesnoi vzaiemodii v mikroservisnii arkhitekturi [Implementing interprocess communication in microservice architecture]. *Visnyk Odeskoho natsionalnoho morskoho universytetu – Herald of the Odessa national maritime university*, 1(67), 81–89 [in Ukrainian].
4. Rudrabhatla, Ch. K. (2018). Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture. *International Journal of Advanced Computer Science and Applications*, 9(8).
5. Kristianto, H., Zahra, A. (2021). Performance analysis of choreography and orchestration in microservices architecture. *Journal of Theoretical and Applied Information Technology*, 99(18).
6. Harding, R., Aken, D. V., Pavlo, A., Stonebraker, M. (2017). An Evaluation of Distributed Concurrency Control. *Proceedings of the VLDB Endowment*, 10(5).
7. Garcia-Molina, H., Salem, K. (1987). SAGAS. *Proceedings of the 1987 ACM SIGMOD international conference on Management of data*, 249–259.