УДК 004.65+004.72 DOI: 10.31673/2412-9070.2024.023941

В. М. МОЛЯВЧИК, аспірант,
Державний університет інформаційно-комунікаційних технологій, Київ

ВИКОРИСТАННЯ ШАБЛОНУ ПРОЄКТУВАННЯ SAGA ДЛЯ КЕРУВАННЯ ТРАНЗАКЦІЯМИ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Використання мікросервісної архітектури в розробленні програмного забезпечення, у межах якого кожен сервіс є самодостатнім учасником зі своєю зоною відповідальності, своїм окремим сховищем даних, набором технологій та робочим оточенням, кидає виклик у контексті роботи транзакцій у розподіленому режимі роботи сервісу. Адже з погляду надійності й цілісності системи вкрай важливо забезпечити консистентність даних, що передбачає відповідність визначеним обмеженням і правилам, відсутність протиріч та узгодженість даних між компонентами системи. Для забезпечення консистентності даних може бути використано шаблон проєктування SAGA, що передбачає набір інструкцій для коректного керування транзакціями в розподіленій архітектурі.

Ключові слова: мікросервіс; сховище даних; розподілена транзакція; SAGA; консистентність.

Вступ

Постановка проблеми. Архітектура, побудована на мікросервісній взаємодії, надає спроможності з декомпозиції комплексних задач, оперативної імплементації нового функціонала, ізоляції критичних складових та масштабування окремих компонентів програмного забезпечення. Переважно мікросервісна архітектура реалізує принцип «Database per service» — підхід, за яким кожен мікросервіс використовує свою власну базу даних замість єдиної спільної бази даних, як це зазвичай реалізовано в монолітній архітектурі.

У ситуації децентралізованого сховища даних (рис. 1) для коректної роботи сервісу постає потреба в механізмі, що забезпечить консистентність (Consistency відповідно до ACID) даних між усіма складовими.

Консистентність передбачає, що стан даних за будь-яких помилок чи збоїв у роботі одного з мікросервісів завжди відповідає визначеному набору правил чи запобіжників. Узгодженість даних у системі вкрай важлива з таких причин:

- уникнення конкурентності та конфліктів;
- відповідність бізнес-логіці застосунку;
- запобігання суперечливих станів пов'язаних абстракцій;
- відновлення до коректного стану після збоїв одного чи кількох компонентів.

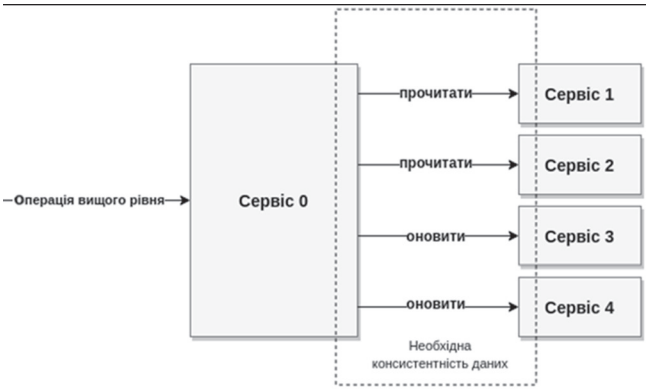

Рис. 1

Основна частина

Аналіз предметної сфери. Забезпечення узгодженості даних у разі оперування кількома сховищами може бути реалізовано із застосуванням розподілених транзакцій. Для дотримання властивостей ACID розподілені транзакції можуть використовувати такі протоколи: двофазний коміт (2PC), трифазний коміт (3PC), розподілені блокування.

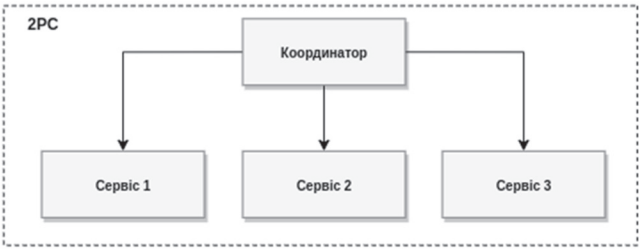

Рис. 2

Проте на практиці розподілені транзакції накладають певні обмеження: використання єдиної СКБД, падіння продуктивності в процесі масштабування, збій координатора (рис. 2) розподіленої транзакції, що призводить до втрати її поточного стану.

Тому для архітектури з міжсервісною комунікацією більш оптимальним рішенням є використання шаблону проєктування SAGA, що забезпечує глобальну узгодженість даних за допомогою наборів локальних транзакцій (рис. 3), окремих для кожного мікросервісу, а також відповідні бази даних. Суть

шаблону полягає в тому, що кожна локальна транзакція здійснює оновлення сховища, після чого публікує подію чи повідомлення, яке ініціює наступну локальну транзакцію.

Рис. 3

За таких умов конкретна локальна транзакція відповідає властивостям ACID, оскільки працює виключно з одним сховищем. У разі, якщо будь-яка локальна транзакція виявилась невдалою, то шаблоном SAGA передбачається запуск компенсувальних транзакцій, які скасовують внесені зміни попередніми успішними транзакціями.

Отже, операція вищого рівня ініціює першу локальну транзакцію для Сервіс 1 (див. рис. 3), після завершення якої Сервіс 1 публікує повідомлення про завершення локальної транзакції, активізуючи виконання наступної транзакції для Сервіс 2, і далі за схемою.

Координація всіх етапів імплементується за допомогою асинхронного обміну повідомленнями, важливою властивістю якого є гарантування виконання всіх етапів SAGA через буферизацію повідомлень доти, доки мікросервіс їх не прочитає.

Традиційні транзакції ACID можна автоматично скасувати за допомогою вбудованих можливостей СКБД чи ОРМ, завдяки яким стан бази даних повернеться в початкове положення. У разі використання SAGA це неможливо, оскільки на кожному етапі у визначеному порядку здійснюється оновлення в локальній базі даних відповідного сервісу. Тому для скасування внесених змін SAGA використовує компенсувальні транзакції, які мають бути завчасно реалізовані в бізнес-логіці сервісу та виконані у зворотному порядку для коректного анулювання всіх попередніх етапів (рис. 4).

Рис. 4

Розглянемо реалізацію SAGA за способами координації. Особливістю SAGA з координацією ***хореографія*** є можливість кожної транзакції публікувати події, що ініціюють транзакції в інших мікросервісах. Такий спосіб дає змогу учасникам обмінюватись подіями без додаткового програмного компонента для моніторингу транзакцій, використовуючи Pub/Sub-підхід (рис. 5).

Цей спосіб доречний для систем із невеликою кількістю учасників і лінійною міжсервісною взаємодією. Також перевагою є те, що він не додає єдиної точки збою, оскільки відповідальність за транзакції покладено на самі мікросервіси.

Недоліки хореографії полягають у можливості виникнення циклічних залежностей та імовірних проблемах щодо модифікації наявної системи через складність відстеження підписок мікросервісів.

SAGA з координацією ***оркестрування*** — це наявність окремого програмного компонента — оркестратора, що виступає як контролер, обробляє всі транзакції та вказує мікросервісам, які саме транзакції вони мають далі виконати (рис. 6).

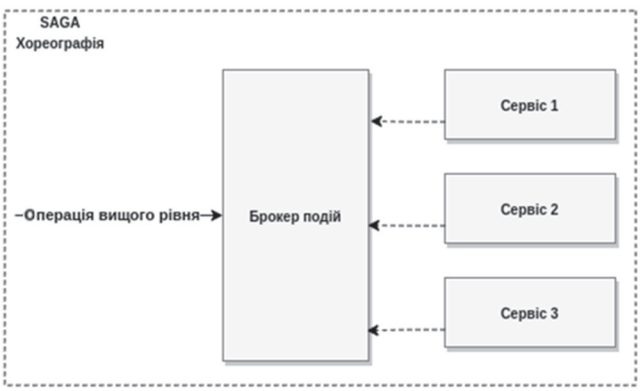

Рис. 5

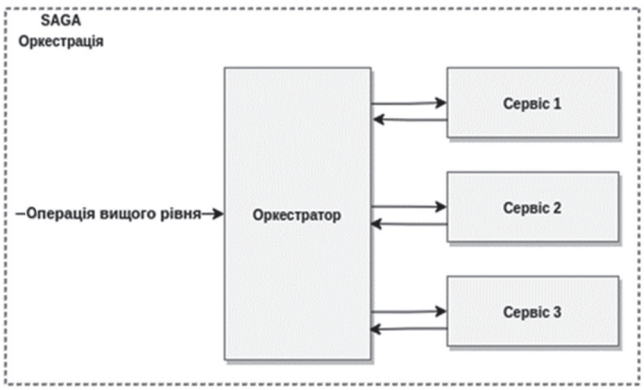

Рис. 6.

Перевагами оркестрування є унеможливлення циклічних залежностей, структурованість, легкість та очевидність у керуванні скасуваннями транзакцій. Оркестрування рекомендовано реалізовувати для складних систем із великою кількістю учасників.

Недоліки полягають у додаванні додаткової єдиної точки збою, оскільки повну відповідальність дій покладено на оркестратора, також будь-яке масштабування логіки потребує модифікації самого оркестратора.

Висновки

Розроблення програмного забезпечення з мікросервісною архітектурою передбачає імплементацію-механізму забезпечення консистентності даних. Для уникнення використання розподілених транзакцій рекомендовано використовувати шаблон проєктування SAGA, що дає змогу забезпечити глобальну узгодженість даних за допомогою наборів локальних та компенсувальних транзакцій. Шаблон стійкий до збоїв та забезпечує надійність та цілісність системи.

Переваги SAGA в імплементації ***хореографія*** полягають у легкості горизонтального масштабування, відсутності сильної зв'язності між компонентами.

Переваги імплементації ***оркестрування*** такі: уникнення циклічних залежностей, структурованість, легкість та очевидність у керуванні скасуваннями транзакцій.

Проте шаблон не варто застосовувати для циклічних залежностей, а також для трансформації наявної монолітної архітектури на мікросервісну.

Список використаної літератури

*1. **SAGA** Pattern Briefly [Електронний ресурс]. URL: https://medium.com/trendyol-tech/saga-pattern-briefly-5b6cf22dfabc*

*2. **What** is a Distributed Transaction? [Електронний ресурс]. URL: https://hazelcast.com/glossary/distributed-transaction/*

*3. **Saga** Pattern in Microservices [Електронний ресурс]. URL: https://www.baeldung.com/cs/saga-pattern-microservices*

*4. **Saga** Pattern [Електронний ресурс]. URL: https://www.dremio.com/wiki/saga-pattern/*

*5. **SAGA** Design Pattern [Електронний ресурс]. URL: https://www.geeksforgeeks.org/saga-design-pattern/*

*6. **Tushych A. M.** Analysis of problems of production modernization through implementation of IT technologies // Economic trends: new opportunities and threats: International scientific conference. Le Mans, France: Le Mans University, 2021. 2021. November 19-20. P. 58–61.*

*7. **Microservice** Architecture, Pattern Saga [Електронний ресурс]. URL: https://microservices.io/patterns/data/saga.html*

V. Moliavchyk

USING THE SAGA DESIGN PATTERN FOR TRANSACTION MANAGEMENT IN A MICROSERVICE ARCHITECTURE

The use of microservice architecture in software development, in which each service is a self-sufficient participant with its own area of responsibility, its own data store, set of technologies, and work environment, is challenging in the context of transactions in a distributed service mode. After all, from the point of view of reliability and integrity of the system, it is extremely important to ensure data consistency, which implies compliance with defined restrictions and rules, absence of contradictions and consistency of data between system components. The architecture built on microservice interaction provides capabilities for decomposition of complex tasks, operational implementation of new functionality, isolation of critical components and scaling of individual software components. As a rule, the microservice architecture implements the principle «Database per service» — an approach according to which each microservice uses its own database, instead of a single shared database, as is usually implemented in a monolithic architecture. To ensure data consistency, the SAGA design pattern can be used, which provides a set of instructions for the correct management of transactions in a distributed architecture. To avoid the use of distributed transactions, it is recommended to use the SAGA design pattern, which allows for global data consistency through sets of local and compensating transactions. The template is fault-tolerant and ensures system reliability and integrity.

Keywords: microservice; data store; distributed transaction; SAGA; consistency.