УДК 004.415.2 DOI: 10.31673/2412-4338.2023.012941

Корнага Я.І., Губарєв О.М.

Національний Технічний Університет України "Київський політехнічний інститут ім. Ігоря Сікорського", Київ

МОДЕЛІ ЗАБЕЗПЕЧЕННЯ КОНСИСТЕНТНОСТІ ДАНИХ В ДОДАТКАХ З МІКРОСЕРВІСНОЮ АРХІТЕКТУРОЮ

Анотація: *Окремі транзакції бази даних можуть легко відповідати вимогам ACID і забезпечувати надійну узгодженість, однак для розподілених транзакцій необхідно вирішити ряд обмежень.*

Жорсткі транзакції повністю відповідають функції ACID, тоді як при використанні гнучких транзакцій ізоляція повністю не гарантується. На практиці, від вимоги до ізоляції певною мірою відмовляються, щоб забезпечити високу пропускну здатність і продуктивність системи.

Гнучкі транзакції зазвичай дотримуються теорії базової доступності, гнучкого стану, узгодженості в кінцевому підсумку (BASE). Теорія BASE є розширенням теореми CAP, це баланс між узгодженістю та доступністю в CAP. Згідно теорії BASE, ми не можемо досягти сильної узгодженості, однак кожна програма може досягти кінцевої узгодженості, використовуючи відповідний метод відповідно до своїх власних характеристик.

В свою чергу, CAP показує, що розподілена система не може досягти узгодженості, доступності та стійкості до розподілення одночасно. На це варто звернути увагу на етапі проектування системи. Виявлено, що жорсткі транзакції прагнуть до стійкої узгодженості, і тому вони жертвують високою доступністю. А гнучкі транзакції жертвують узгодженістю в обмін на високу доступність системи.

Ключові слова: *мікросервісна архітектура, консистентність даних, розподілені системи, база даних, моноліти, транзакції.*

Kornaga Ya.I., Hubariev O.M.
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

DATA CONSISTENCY MODELS IN APPLICATIONS WITH MICROSERVICE ARCHITECTURE

Abstract: *Individual database transactions can easily meet ACID requirements and ensure strong consistency, but distributed transactions have a number of limitations to address.*

Hard transactions are are found to be fully ACID compliant, while using flexible transactions isolation is not fully guaranteed. In practice, the isolation requirement is waived to some extent to ensure high throughput and system performance.

Flexible transactions generally adhere to the Basic Availability, Flexible State, Eventually Consistency (BASE) theory.

The BASE theorem is an extension of the CAP theorem. This is the balance between consistency and availability in CAP. According to the BASE theory, we cannot achieve strong consistency, however, each program can achieve ultimate consistency by using an appropriate method according to its own characteristics.

CAP shows that a distributed system cannot achieve consistency, availability, and distribution resilience at the same time. It is worth to pay attention at the stage of system design. It was found that

hard transactions are strive for persistent consistency, and so they sacrifice high availability. Flexible transactions sacrifice consistency in exchange for high system availability.

Keywords: *microservice architecture, data consistency, distributed systems, database, monoliths, transactions.*

1. Постановка проблеми.

Процес використання розподілених систем стрімко розвивається. Основною причиною такого розвитку та поширення є прагнення створити систему, яка є надійною, легко масштабованою, стійкою до відмови. Розподілені системи можуть забезпечити виконання цих вимог та перехід від більш традиційних монолітних систем, коли вся система сконцентрована в одному блоці і всі частини додатку взаємодіють між собою в одному проєкті, до систем з мікросервісною архітектурою, коли кожен сервіс виконує одну бізнес-функцію і розміщений в окремому проєкті.

Перехід від монолітної архітектури до архітектури мікросервісів є складним завданням, яке складається з таких проблем, як ідентифікація мікросервісів, декомпозиція коду, побудова комунікації між мікросервісами, незалежне розгортання тощо. Одним із ключових питань є адаптація сховища даних до архітектури мікросервісів. Монолітна архітектура взаємодіє з єдиною базою даних, тоді як в архітектурі мікросервісів зберігання даних децентралізоване, кожен мікросервіс працює незалежно та має власне приватне сховище даних.

2. Мета і задачі дослідження.

Провести порівняльний аналіз існуючих підходів до вирішення проблем розподілених транзакцій, запропонувати критерії вибору того чи іншого підходу при побудові систем з мікросервісною архітектурою.

3. Аналіз останніх досліджень і публікацій.

В останні кілька років зростає інтерес до аналізу моделей і методів обробки даних в системах з мікросервісною архітектурою.

Існуючі дослідження в основному описують характеристики та визначають основні вимоги для проектування мікросервісної архітектури, та найважливішою частиною архітектури мікросервісів є консистентність даних.

Оскільки деякі бізнес-операції охоплюють кілька сервісів, потрібен механізм для забезпечення узгодженості даних. Такі механізми вже існують, але мають ряд недоліків при застосуванні в різних архітектурах.

4. Результати дослідження.

Головним показником добре структурованої програмної системи є її архітектура. Монолітні системи вже не здатні задовольнити вимоги сучасних додатків, які є досить великими, складними і потребують багато ресурсів, тому все частіше їх замінюють мікросервісами.

Такі компанії як Amazon, Netflix, Uber та Etsy з часом розібрали свої монолітні програми та переробили їх на архітектури на основі мікросервісів, що призвело до отримання прибутку та значних переваг на ринку.

Мікросервіси легко масштабувати - будь-якої миті можливо додати або видалити модуль. Якщо мікросервіс досягне граничного навантаження, можна буде розгортати нові екземпляри сервісу у сусідньому кластері. Вони більш стійкі до збоїв, оскільки при відмові одного з мікросервісів порушується робота тільки тих функцій, за які він відповідає. Будь-якої миті можна змінити мікросервіс, або технології які він використовує, незалежно від інших сервісів системи, не порушуючи загальну роботу додатку. Мікросервісна архітектура полегшує тестування системи, оскільки можна тестувати кожен сервіс окремо. Також, команди можуть

працювати незалежно, кожна над своїм сервісом, що дозволяє пришвидшити впровадження нового функціоналу.

Рис.1. Виконання транзакції в моноліті
Складено автором на основі джерела [1]

Проте, цей архітектурний підхід також має ряд недоліків - зростають інфраструктурні витрати (тести, інструкції з розгортання, інфраструктура хостингу, інструменти моніторингу), збільшуються витрати на комунікацію між сервісами та базою даних. Якщо система має високе навантаження це може стати проблемою. Також потрібно ретельно продумати протоколи і стандарти обміну інформацією між сервісами. Збільшуються витрати на підтримку системи, оскільки замість супроводження однієї програми, у випадку мікросервісів, потрібно підтримувати багато програм і, часто, серверів.

Рис.2 Виконання транзакції в мікросервісах
Складено автором на основі джерел [1,12,13]

При використанні даного підходу, відсутня центральна база даних і бізнес-логіка охоплює декілька локальних транзакцій. Це означає, що неможливо використовувати одну робочу одиницю транзакції серед баз даних в архітектурі мікросервісів. Але при цьому існує необхідність виконання ACID принципів у створеному додатку:

- Атомарність (Atomicity) – незалежно від кількості кроків в середині транзакції, всі вони мають завершитись успішно, або транзакція має бути відкочена.
- Узгодженість (Consistency) – усі дані в базі даних мають бути узгодженими наприкінці транзакції.

- Ізоляція (Isolation) – транзакція не може змінювати дані, які змінюються іншою транзакцією.

- Довговічність (Durability) – дані мають бути збережені в базі даних наприкінці транзакції [5].

Також кожна програма може використовувати різні технології для керування своїми даними, наприклад базами даних, відмінними від SQL. Тому, з точки зору управління даними, у мікросервісному підході до побудови додатків існує ряд критичних моментів, таких як керування транзакціями та узгодженість/цілісність даних.

На сьогоднішній день існує декілька підходів вирішення цієї проблеми, а саме

1. Двофазний коміт (Рис.3).
2. TCC (Рис.4).
3. SAGA (Рис.5).
4. Локальний обмін повідомленнями (Рис.6).
5. Транзакційний обмін повідомленнями (Рис.7).

При використанні першого підходу транзакції виконуються на двох або більше ресурсах (наприклад, базах даних, чергах повідомлень). При цьому, цілісність даних у кількох базах даних гарантує менеджер розподілених транзакцій або координатор. Розподілена транзакція є дуже складним процесом, оскільки задіяно кілька ресурсів. Для забезпечення коректного завершення розподіленої транзакції широко застосовується алгоритм двофазної фіксації (2PC). Як правило, цей алгоритм застосовується для оновлень, які фіксуються за короткий проміжок часу. Всі мікросервіси, які використовуються в роботі, готуються до фіксації та повідомляють менеджера розподілених транзакцій про готовність до завершення транзакції. На наступному етапі або відбувається фіксація, або менеджер транзакцій відправляє всім мікросервісам команду виконати повернення до попереднього стану.

Фаза підготовки. Менеджер транзакцій (МТ) реєструє запуск транзакції та запитує кожного менеджера ресурсів (МР) про те, чи готовий він до виконання підготовчих операцій.

Після того як МР отримує замовлення, він оцінює свій власний стан і намагається виконати підготовчі операції для локальної транзакції, такі як резервування ресурсів, блокування ресурсів і виконання операцій. Потім МР чекає наступних запитів від МТ без фіксації транзакції. Якщо попередня спроба виявляється невдалою, МР інформує МТ про те, що виконання цієї фази не вдається, і відкочує виконані операції. В цьому випадкуМР більше не бере участі в транзакції.

МТ збирає відповіді МР і реєструє фазу підготовки транзакції як завершену.

Фаза фіксації або відкату. Ця фаза ініціює операцію фіксації або відкату транзакції на основі результату попередньої фази.

Якщо всі МР відповіли успішно на попередньому кроці, тоді виконуються такі дії: МТ реєструє транзакцію як зафіксовану та відправляє команду про здійснення транзакції всім МР.

Після того, як МР отримають команду, вони фіксують транзакції, звільняють ресурси та відповідають МТ «коміт завершено».

Якщо МТ отримує відповіді від усіх МР, він реєструє транзакцію як завершену. Якщо будь-який МР відповідає помилкою виконання або не відповідає вчасно на попередньому кроці, МТ вважатиме транзакцію невдалою. В такому випадку виконуються наступні дії: МТ реєструє транзакцію як перервану та відправляє команду про відкат транзакції всім МР.

Отримавши команду, МР відкочують транзакцію, звільняють ресурси та відповідають МТ «відкат завершено».

Якщо МТ отримує відповіді від усіх цих МР, він реєструє транзакцію як завершену [12].

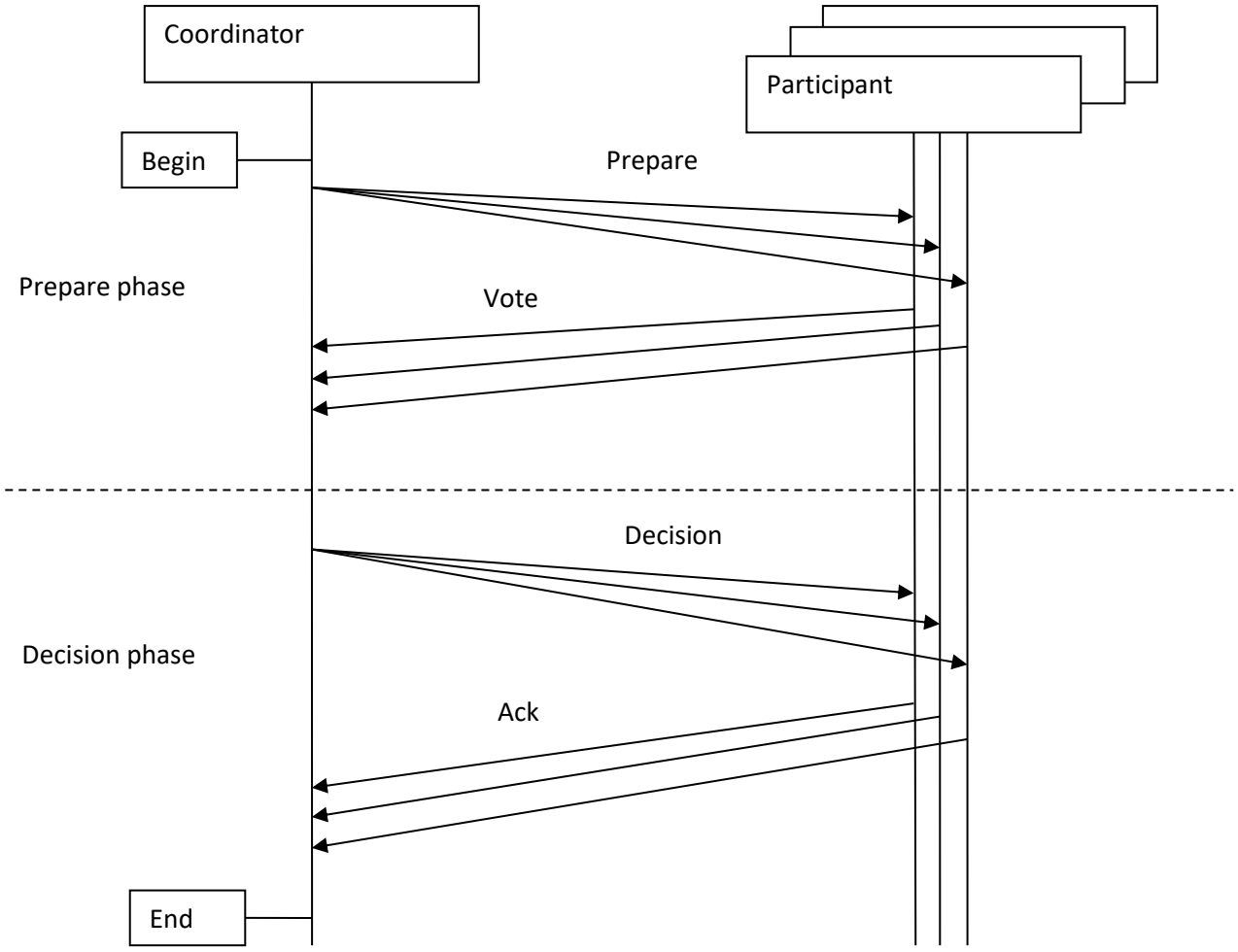

Рис.3 Модель двофазного коміту
Складено автором на основі джерел [1,12]

До переваг використання підходу двохфазного коміту можна віднести:

1. Гарантування атомарності транзакції. Транзакція буде зафіксована лише у випадку, коли всі мікросервіси, які приймають участь у процесі, спрацюють успішно, в іншому випадку зміни не будуть зафіксовані.
2. Дозволяє ізолювати читання від запису, оскільки змін в об'єктах не видно до тих пір, поки координатор транзакцій зафіксує ці зміни.
3. Забезпечується узгодженість даних.
4. Забезпечується довговічність, оскільки транзакції базуються на локальних транзакціях.

До недоліків даної моделі можна віднести:

1. Дуже складний процес для обслуговування.
2. Висока затримка та низька пропускна здатність, оскільки це процес блокування (не підходить для сценаріїв високого навантаження).
3. Можливі взаємоблокування між транзакціями.
4. Координатор транзакцій є єдиною точкою відмови [2].

Try, Commit, and Cancel (TCC) — це модель компенсаційних транзакцій. Ця модель вимагає, щоб кожна служба програми надавала три інтерфейси- інтерфейси спроби, фіксації та скасування. Основна ідея цієї моделі полягає в тому, щоб якнайшвидше зняти блокування

ресурсів шляхом резервування ресурсів (забезпечення проміжних станів). Якщо транзакцію можна зафіксувати, зарезервовані ресурси підтверджуються. Якщо транзакцію потрібно відкотити, зарезервовані ресурси звільняються.

TCC також є протоколом двофазної фіксації, і його можна вважати варіантом XA з двофазною фіксацією, який не підтримує блокування ресурсів протягом тривалого часу.

Модель TCC поділяє транзакції на дві фази:

Фаза перша. На першому етапі TCC виконує перевірку бізнесу (на консистентність) і резервує бізнес-ресурси (для квазіізоляції). Це визначає операцію спроби TCC.

Фаза друга. Якщо резервування всіх бізнес-ресурсів успішне на етапі спроби, виконується операція підтвердження. В іншому випадку виконується операція скасування.

Підтвердження: на етапі підтвердження фактично виконується бізнес – операція і використовуються ресурси, що були зарезервовані на етапі спроби. Фаза підтвердження має бути ідемпотентною. Крім того, спроба виконання бізнес-операції може бути повторена, якщо попередня була невдалою.

Скасування: операція скасування скасовує виконання бізнес-операції, звільняє зарезервовані ресурси та повторює спробу у разі невдачі [2].

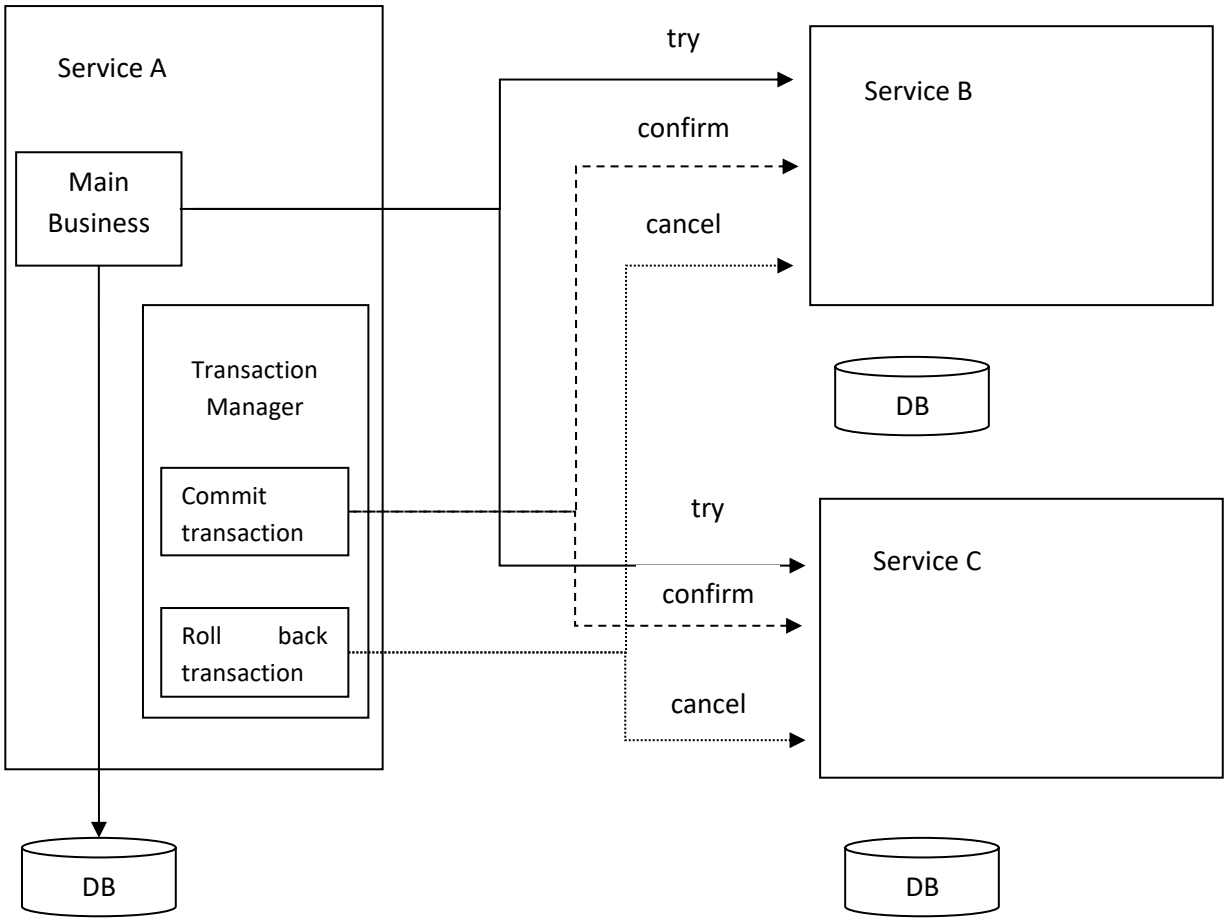

Рис.4. Модель TCC

Складено автором на основі джерел [1,13]

Основними характеристиками використання даного підходу є:

1. Високий рівень паралелізму.
2. Відсутність довгострокового блокування ресурсів.

3. Забезпечується: атомарність (ініціатор транзакції координує та гарантує, що всі транзакції зафіксовані, або всі вони відкочуються), узгодженість, ізоляція(TCC реалізує ізоляцію даних шляхом попереднього розподілу ресурсів на етапі спроби), довговічність(TCC реалізує довговічність, координуючи кожну транзакцію).

4. Більші витрати на розробку, оскільки додатково необхідні інтерфейси Try, Confirm, Cancel.

5. Краща узгодженість, ніж SAGA, де може бути проміжний стан, але можуть виникати помилки при передачі.

Інший підхід - це узгодженість у загальному підсумку. При використанні даного підходу кожен сервіс публікує подію щоразу, коли оновлює свої дані. Інші сервіси підписуються на події. При отриманні події сервіси оновлюють свої дані. Таким чином, розподілена транзакція виконується як сукупність асинхронних локальних транзакцій на кількох відповідних мікросервісах. Мікросервіси обмінюються інформацією через шину подій.

SAGA забезпечує керування транзакціями за допомогою послідовності локальних транзакцій мікросервісів. Кожен мікросервіс має власну базу даних і може керувати локальними транзакціями атомарним способом із суворою узгодженістю.

Таким чином, шаблон SAGA групує ці локальні транзакції та послідовно викликає одну за одною. Кожна локальна транзакція оновлює базу даних і публікує подію для запуску наступної локальної транзакції.

Якщо один із кроків не вдається, SAGA запускає транзакції відкату, які є набором компенсаційних транзакцій, які відкочують зміни в попередніх мікросервісах і відновлюють узгодженість даних.

Існує два способи реалізації саги: «хореографія» та «оркестрація».

Хореографія передбачає узгодження SAGA’s із застосуванням принципів публікації-підписки. За допомогою хореографії кожен мікросервіс запускає власну локальну транзакцію та публікує події в системі посередника повідомлень, які запускають локальні транзакції в інших мікросервісах.

Ще один алгоритм SAGA — оркестрація. Оркестрація забезпечує координацію SAGA“s із централізованим мікросервісом контролера. Ця централізована мікрослужба контролера керує робочим процесом SAGA та викликає послідовне виконання транзакцій локальних мікрослужб [9].

Мікросервіси оркестратора виконують SAGA-транзакції та централізовано керують ними, а якщо один із кроків не вдасться, виконуються кроки відкату з компенсуючими транзакціями.

Результати проведеного аналізу підтверджують, що основними перевагами використання підходу узгодженості у є:

1. Кожен мікросервіс виконує локальну атомарну транзакцію. При цьому, оскільки це асинхронний процес, то робота мікросервісів не блокується. А це також означає, що не блокується і база даних.

2. Оскільки дане рішення є асинхронним і реалізоване на роботі з подіями, то можна забезпечити масштабованість системи при роботі під високим навантаженням.

3. Відсутні взаємоблокування між транзакціями.

4. Відсутня єдина точка відмови.

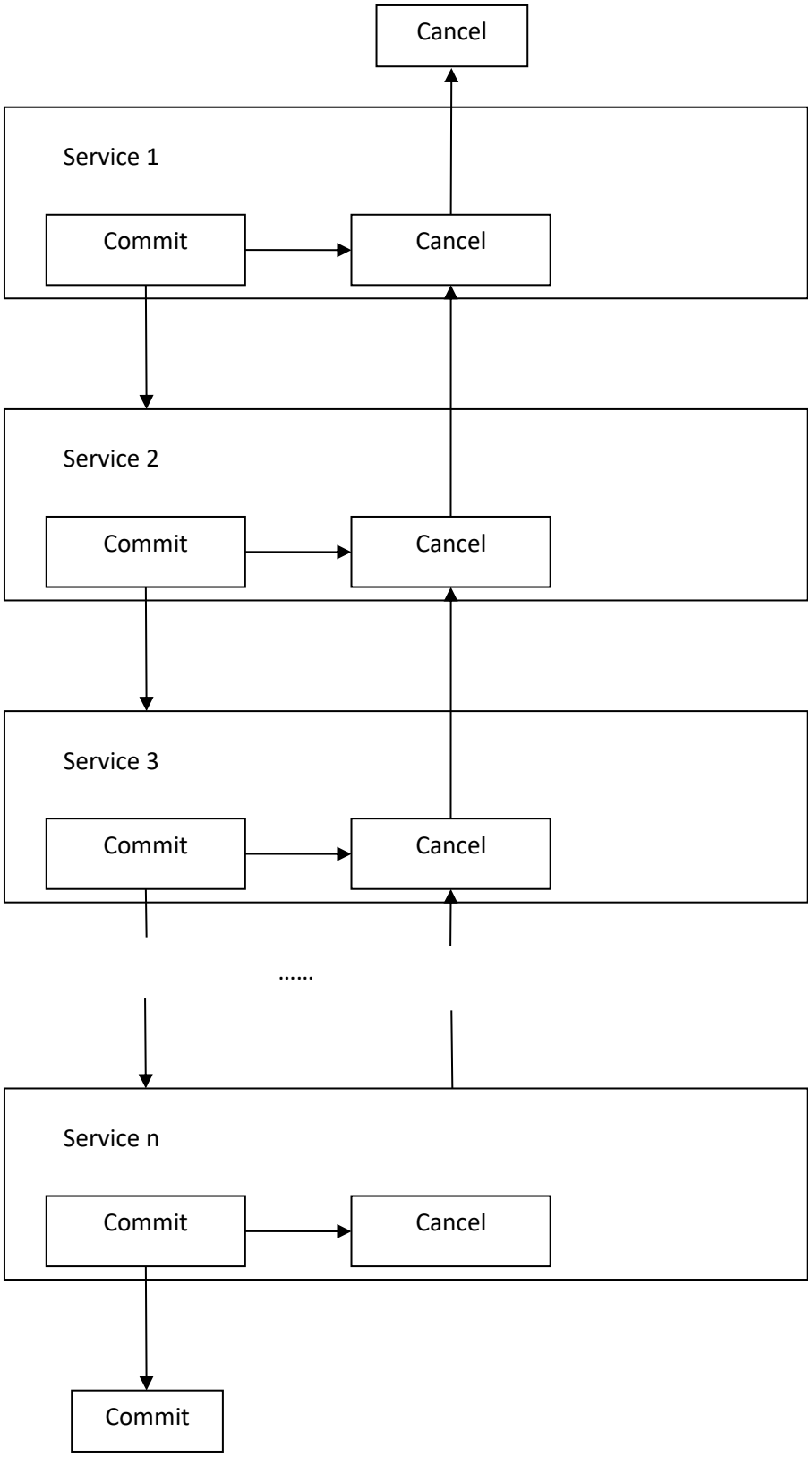

Рис.5. Модель SAGA

Складено автором на основі джерел [1, 9, 12]

Основними недоліками використання даного підходу є :

1. Не забезпечується ізоляція при читанні.
2. При збільшенні кількості мікросервісів, ускладнюється їх налагодження та підтримка.
3. Збільшується вартість розробки.
4. Складність дизайн.
5. Підтримувати узгодженість даних між розподіленими сховищами даних складно.

Наступний підхід — це алгоритм розподіленої транзакції на основі повідомлень. Основна концепція даного підходу полягає в інформуванні інших учасників транзакції про власний статус виконання через систему повідомлень. Впровадження систем повідомлень може більш ефективно відокремити учасників транзакцій і дозволити кожному учаснику діяти асинхронно. Складність у реалізації цього алгоритму полягає в тому, щоб забезпечити узгодженість між виконанням локальних транзакцій і відправленням повідомлень. Тому що обидві дії повинні бути успішними або скасованими.

Існує два основних рішення для алгоритму розподілених транзакцій на основі повідомлень:

1. Рішення на основі транзакційних повідомлень
2. Рішення на основі локальних повідомлень [12].

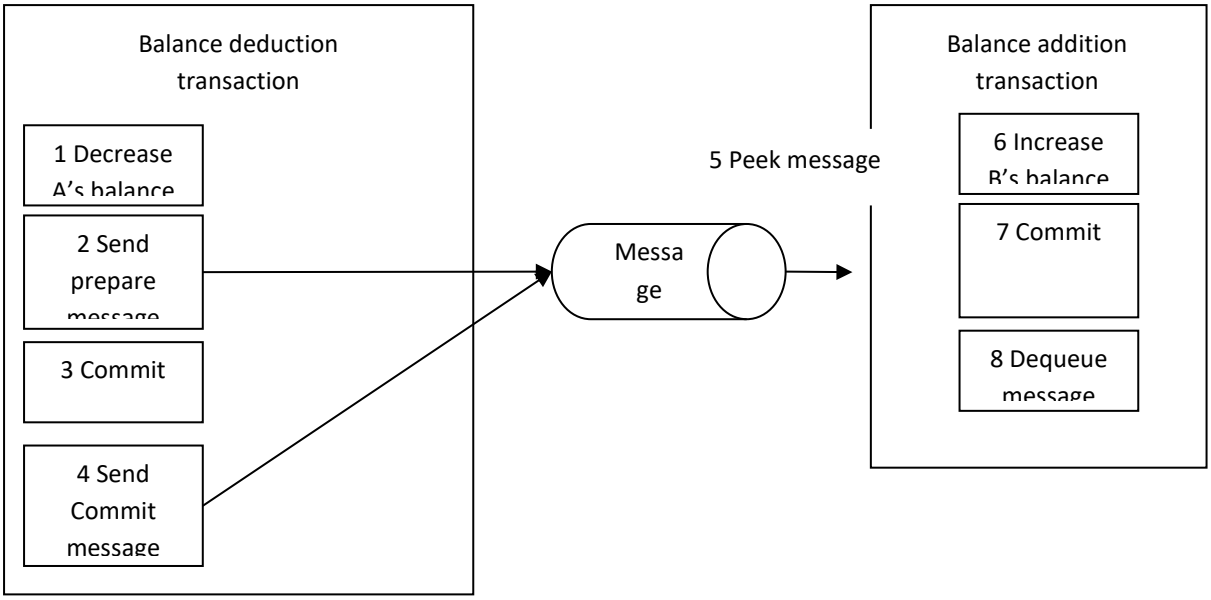

Рис.6. Модель транзакційного обміну повідомленнями
Складено автором на основі джерел [9,12]

Алгоритм роботи моделі транзакційного обміну повідомленнями. Ініціатор транзакції заздалегідь надсилає транзакційне повідомлення. Після того, як менеджер черги (МЧ) отримує транзакційне повідомлення, він його зберігає, оновлює його стан на «буде надіслано» та надсилає підтвердження відправнику.

Якщо ініціатор транзакції не отримує підтвердження, виконання локальної транзакції скасовується. Якщо ініціатор транзакції отримує підтвердження, локальна транзакція виконується, а інше повідомлення надсилається МЧ для сповіщення про виконання локальних транзакцій.

Після того як МЧ отримує сповіщення, він змінює стан транзакційного повідомлення на основі результату виконання локальної транзакції. Якщо виконання пройшло успішно, МЧ

змінює стан повідомлення на «для споживання» і доставляє його споживачам. Якщо виконання не вдається, повідомлення видаляється.

Під час виконання локальної транзакції сповіщення, надіслане до МЧ, може бути втрачено. Тому МЧ, який підтримує транзакційні повідомлення, має регулярну логіку сканування. Шляхом сканування МЧ визначає повідомлення, які залишаються в стані «для надсилання», і ініціює запит до відправника повідомлення щодо остаточного стану повідомлення. На основі результату запиту МЧ оновлює стан повідомлення. Отже, ініціатор транзакції повинен надати системі МЧ інтерфейс для запиту стану транзакційного повідомлення.

Якщо стан транзакційного повідомлення «готовий до надсилання», МЧ надсилає повідомлення учасникам нижчого рівня. Якщо надсилання буде невдалим, система продовжить повторювати спроби.

Після отримання повідомлення наступні учасники виконують локальні транзакції. Якщо виконання локальних транзакцій проходить успішно, в систему МЧ надсилається підтвердження. Якщо виконання не вдається, підтвердження не надсилається. У цьому випадку МЧ постійно надсилає повідомлення наступному учасникам, які не надсилають підтвердження [2].

Наступним підходом є локальний обмін повідомленнями. Ключова ідея полягає в тому, щоб запровадити постійну чергу повідомлень для виконання завдань, які вимагають розподіленої обробки в асинхронному режимі.

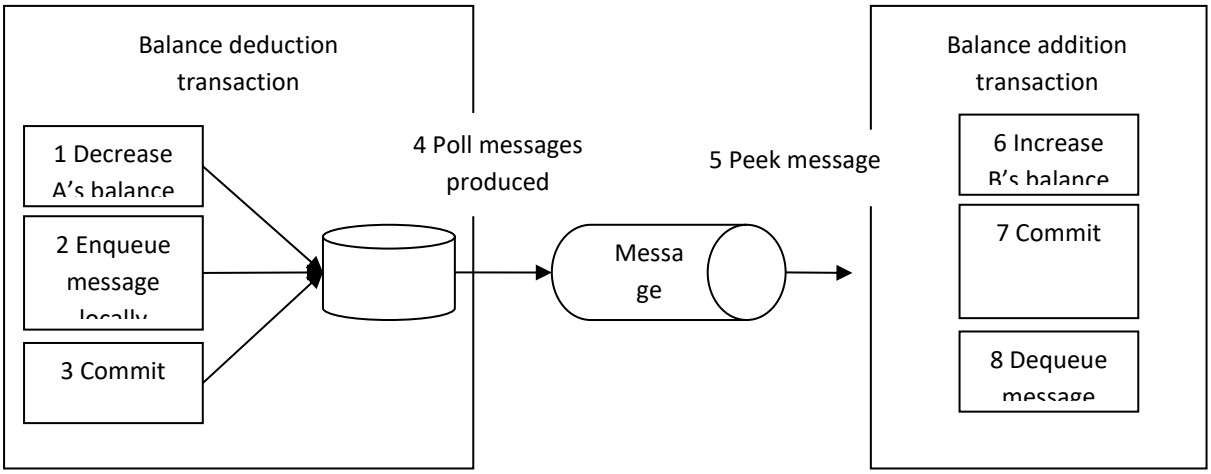

Рис.7 Модель локального обміну повідомленнями
Складено автором на основі джерел [9,12]

Основна концепція цього режиму полягає в тому, що ініціатор транзакції підтримує локальну таблицю повідомлень. Бізнес-операції та операції з локальною таблицею повідомлень виконуються в одній локальній транзакції. Якщо завдання виконано успішно, у локальній таблиці повідомлень також записується повідомлення зі станом «для надсилання». Система запускає заплановане завдання для регулярного сканування локальної таблиці повідомлень на наявність повідомлень у стані «для надсилання» та надсилає їх до менеджера черги (МЧ). Якщо надсилання не вдається або минув час очікування, повідомлення надсилатиметься повторно, доки воно не буде успішно надіслано. Потім завдання видалить запис стану з локальної таблиці повідомлень. Подальший процес споживання та підписки подібний до режиму транзакційного повідомлення [9].

При використанні даного підходу повідомлення ставиться в чергу в тій самій транзакції, що й бізнес-операція,що гарантує атомарність цієї операції та обміну повідомленнями. Вони або всі завершуються, або обидва зазнають невдачі .

До особливостей моделі транзакційного обміну повідомленнями можна віднести:

- транзакції необхідно розділити на кілька завдань;
- розробникам необхідно створити додаткові таблиці повідомлень;
- необхідний механізм для опитування кожної локальної таблиці повідомлень;
- логіка на стороні споживача вимагає додаткових механізмів для відміни операцій, якщо вони не вдаються після повторної спроби.
-

Таблиця 1

Порівняльна характеристика моделей виконання розподілених транзакцій

Функція	Двофазний коміт	TCC	SAGA	Обмін повідомленнями
Відсутня єдина точка відмови	Так	Так	Так	Так
Відсутнє брудне читання	Так	Так	Ні	Ні
Підтримує гетерогенні системи	Ні	Так	Так	Так
Автоматичне відкочування після таймауту	Так	Так	Ні	Ні

Складено автором на основі джерел [5, 9 ,12]

Коли ми підбираємо розподілене рішення для системи, слід виходити з вимог щодо узгодженості. Для сценаріїв, які вимагають сильної узгодженості, рішення двофазного коміту повинні бути пріоритетними. Для сценаріїв, які вимагають остаточної узгодженості, перевага надається гнучким транзакціям.

5. Висновки і перспективи подальших досліджень.

Якщо під час виконання додатка потрібно оновити дані одразу на декількох мікросервісах, то підхід з узгодженістю у загальному підсумку матиме перевагу у порівнянні з використанням двофазного підходу, оскільки може масштабуватися у розподіленому середовищі. Але при використанні підходу з узгодженістю у загальному підсумку потрібно вирішувати проблеми з атомарним оновленням бази даних та породженням подій.

Модель локального обміну повідомленнями підходить для асинхронних завдань, які не потребують операцій відміни. Розподілені транзакції на основі повідомлень мають високі вимоги до системи МЧ і вносять певне втручання в бізнес. Для таких транзакцій необхідно або надати інтерфейс для запиту статусу повідомлення транзакції, або підтримувати локальну таблицю повідомлень. Крім того, розподілені транзакції на основі повідомлень не підтримують відкат транзакцій. Якщо транзакція завершується невдачею, її потрібно повторювати, доки вона не буде успішною. Ця функція робить її застосовною до обмежених бізнес-сценаріїв, які менш чутливі до остаточної узгодженості.

Коли ми вибираємо розподілене рішення для системи, ми можемо прийняти рішення на основі вимог узгодженості. Для сценаріїв, які вимагають високої узгодженості, слід віддавати пріоритет рішенням із двофазною фіксацією. Для сценаріїв, які вимагають лише остаточної узгодженості, перевагу надають гнучким рішенням для транзакцій.

Архітектура мікросервісів має багато переваг, такі як висока доступність, масштабованість, автоматизація, автономні команди тощо. Але для досягнення максимальної ефективності мікросервісного архітектурного стилю потрібна низка змін у традиційних методах. І управління даними та узгодженістю є однією з тем, які потрібно ретельно досліджувати.

Список використаних джерел:

1. Nadareishvili I. Microservice Architecture: Aligning Principles, Practices, and Culture / I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen – O'Reilly Media, 2016 – 146 p.
2. Saga distributed transactions pattern [Електронний ресурс] //- Режим доступу:https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga
3. Compensating Transaction pattern [Електронний ресурс]//-Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction
4. Christian PostaThe hardest part about microservices: your data [Електронний ресурс]//- Режим доступу: https://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/
5. A Guide to Transactions Across Microservices [Електронний ресурс] //- Режим доступу:https://www.baeldung.com/transactions-across-microservices
6. Stephen WattsACID explained: Atomic, Consistent, Isolated & Durable [Електроннийресурс]//-Режим доступу: https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/
7. Vincent Bushong, Amr S. Abdelfattah, Abdullah A. Maruf (2011). On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study. https://www.mdpi.com/2076-3417/11/17/7856
8. Miloš Milić, Dragana Makajić-Nikolić (2022). Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. https://www.mdpi.com/2073-8994/14/9/1824
9. Eman Daraghmi, Cheng-Pu Zhang, Shyan-Ming Yuan (2022). Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture. https://www.mdpi.com/2076-3417/12/12/6242
10. Michael J. Donahoo, Michal Trnka, Tomas Cerny (2018). Contextual Understanding of Microservice Architecture: Current and Future Directions. https://www.researchgate.net/publication/322842819_Contextual_understanding_of_microservice_architecture_current_and_future_directions
11. Isak Shabani, Endrit Mëziu, Blend Berisha (2021). Design of Modern Distributed Systems based on Microservices Architecture. https://thesai.org/Publications/ViewPaper?Volume=12&Issue=2&Code=IJACSA&SerialNo=20
12. Aleksandra Stoykov, Zeliko Stojanov (2021). Review of methods for migtating software systems to microservices architecture. https://pdfs.semanticscholar.org/3303/14b9a3e729f1f496a3424654bae46ff81b42.pdf?_ga=2.178782564.1728312785.1671746531-553667512.1671746531
13. Mehmet Söylemez ,Bedir Tekinerdogan, Ayça Kolukısa Tarhan (2022). Feature-Driven Characterization of Microservice Architectures: A Survey of the State of the Practice. https://www.mdpi.com/2076-3417/12/9/4424
14. Freddy Tapia, Miguel Ángel Mora, Walter Fuertes (2020). From Monolithic Systems to Microservices: A Comparative Study of Performance. https://www.mdpi.com/2076-3417/10/17/5797

References:

1. Nadareishvili I. Microservice Architecture: Aligning Principles, Practices, and Culture / I. Nadareishvili, R. Mitra, M. McLarty, M. Amundsen – O'Reilly Media, 2016 – 146 p.
2. Saga distributed transactions pattern [Електронний ресурс] //- Режим доступу:https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga
3. Compensating Transaction pattern [Електронний ресурс]//-Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction

4. Christian PostaThe hardest part about microservices: your data [Електронний ресурс]//- Режим доступу: https://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/

5. A Guide to Transactions Across Microservices [Електронний ресурс] //- Режим доступу:https://www.baeldung.com/transactions-across-microservices

6. Stephen WattsACID explained: Atomic, Consistent, Isolated & Durable [Електроннийресурс]//-Режим доступу: https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/

7. Vincent Bushong, Amr S. Abdelfattah, Abdullah A. Maruf (2011). On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study. https://www.mdpi.com/2076-3417/11/17/7856

8. Miloš Milić, Dragana Makajić-Nikolić (2022). Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. https://www.mdpi.com/2073-8994/14/9/1824

9. Eman Daraghmi, Cheng-Pu Zhang, Shyan-Ming Yuan (2022). Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture. https://www.mdpi.com/2076-3417/12/12/6242

10. Michael J. Donahoo, Michal Trnka, Tomas Cerny (2018). Contextual Understanding of Microservice Architecture: Current and Future Directions. https://www.researchgate.net/publication/322842819_Contextual_understanding_of_microservice_architecture_current_and_future_directions

11. Isak Shabani, Endrit Mëziu, Blend Berisha (2021). Design of Modern Distributed Systems based on Microservices Architecture. https://thesai.org/Publications/ViewPaper?Volume=12&Issue=2&Code=IJACSA&SerialNo=20

12. Aleksandra Stoykov, Zeliko Stojanov (2021). Review of methods for migtating software systems to microservices architecture. https://pdfs.semanticscholar.org/3303/14b9a3e729f1f496a3424654bae46ff81b42.pdf?_ga=2.178782564.1728312785.1671746531-553667512.1671746531

13. Mehmet Söylemez ,Bedir Tekinerdogan, Ayça Kolukısa Tarhan (2022). Feature-Driven Characterization of Microservice Architectures: A Survey of the State of the Practice. https://www.mdpi.com/2076-3417/12/9/4424

14. Freddy Tapia, Miguel Ángel Mora, Walter Fuertes (2020). From Monolithic Systems to Microservices: A Comparative Study of Performance. https://www.mdpi.com/2076-3417/10/17/5797