ДОДАТОК А

Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії

5. Mazurova O., Syvolovskyi I., Syvolovska O. NoSQL database logic design methods for mongodb and Neo4j. Innovative Technologies and Scientific Solutions for Industries. 2022. No. 2 (20). P. 52–63. Doi: 10.30837/itssi.2022.20.052

38. Mazurova, O. Research of ACID transaction implementation methods for distributed databases using replication technology / Mazurova, O., Naboka, A., Shirokopetleva, M. Innovative technologies and scientific solutions for industries, 2 (16), pp. 19-31. Doi: 10.30837/ITSSI.2021.16.019

ДОДАТОК Б

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

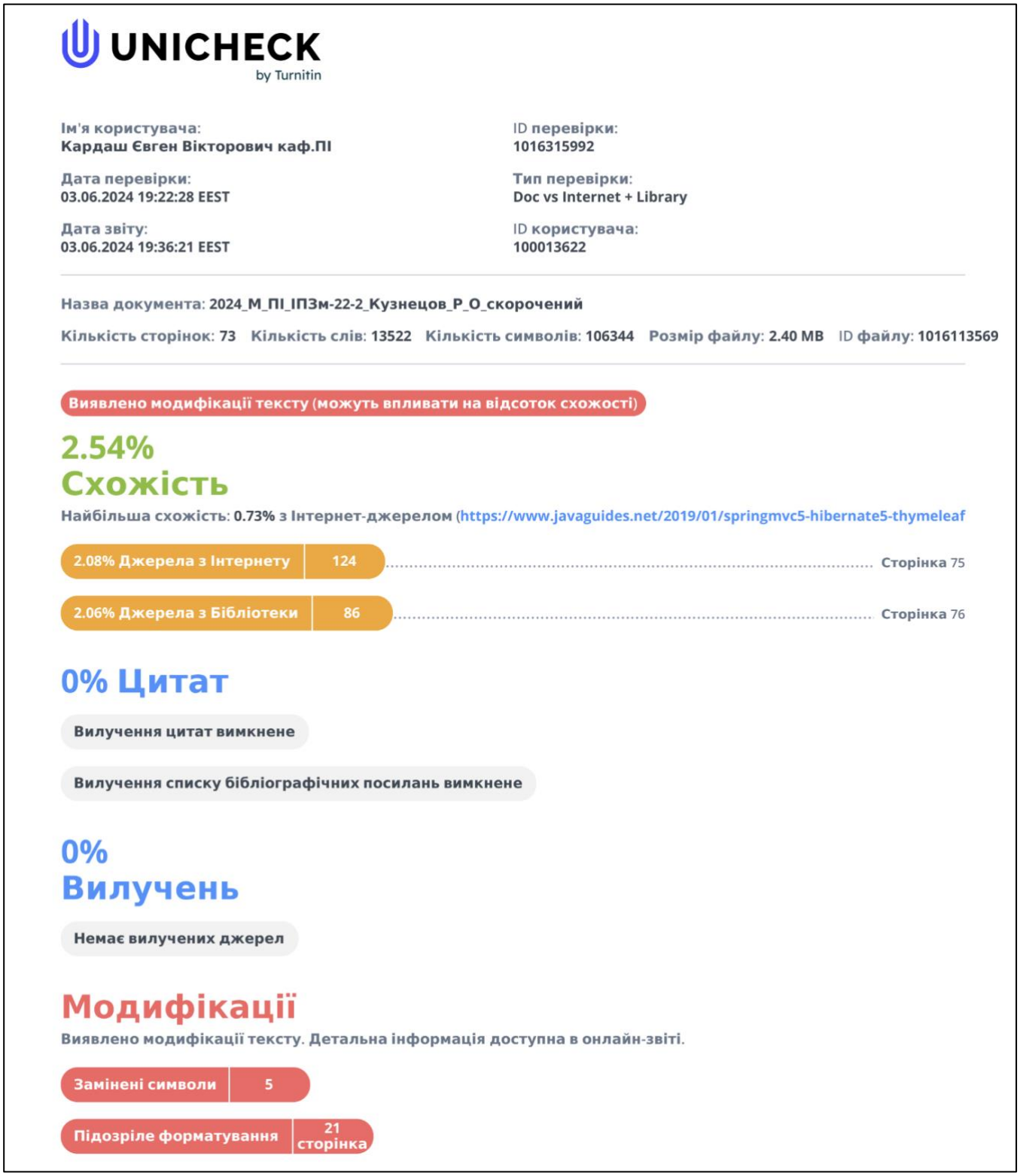


Рисунок Б.1 – Звіт з результатами перевірки роботи на академічну доброчесність

ДОДАТОК В

Слайди презентації

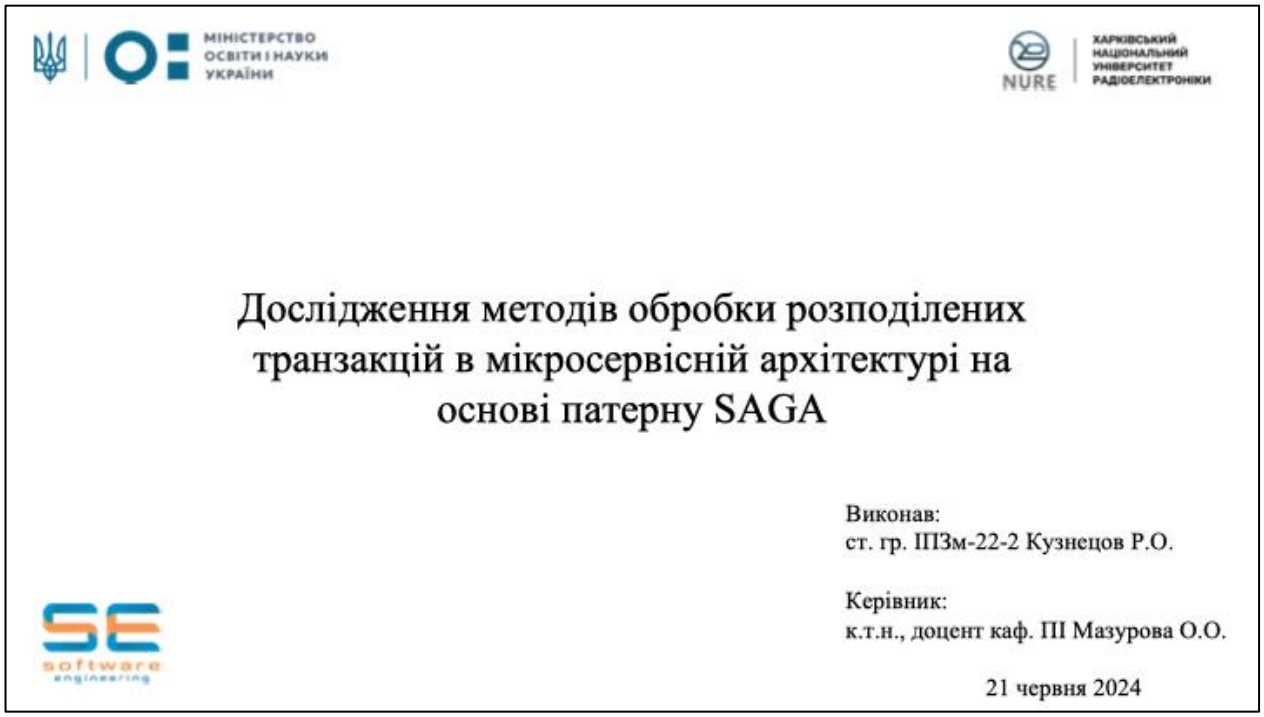


Рисунок В.1 – Титульний слайд

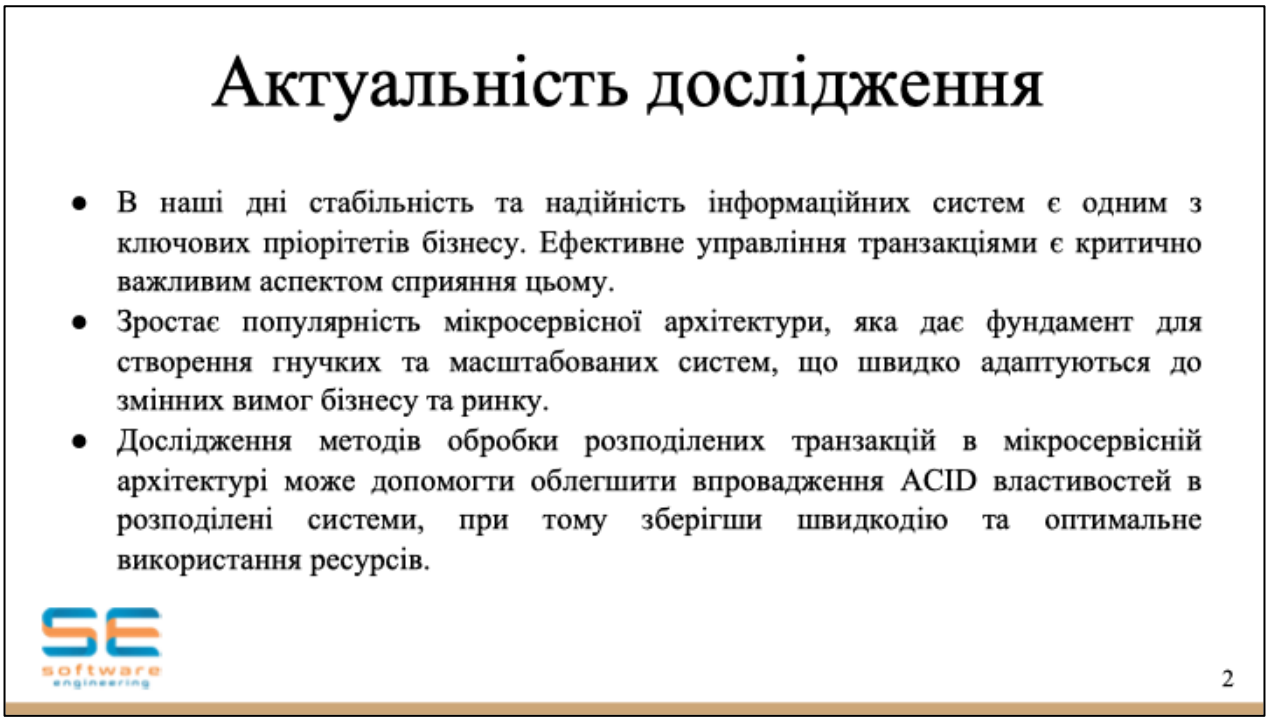


Рисунок В.2 – Слайд - Актуальність дослідження

Постановка задачі

- Провести аналіз та вибір методів обробки розподілених транзакцій для дослідження.
- Провести планування експериментального дослідження (обрати предметну область, обрати метрики для замірів під час експериментів, спроектувати архітектуру мікросервісного додатку).
- Спроектувати БД для обраної предметної області та описати запити для дослідження.
- Реалізувати програмну систему та обрані методи для спроектованих запитів, спроектувати та реалізувати допоміжне ПЗ для автоматизації експериментів.
- Провести експериментальне дослідження, оцінити якість обраних методів та виробити рекомендації стосовно їх вибору та використання.

SE software engineering

3

Рисунок В.3 – Слайд - Постановка задачі

Аналіз та вибір методів обробки розподілених транзакцій для дослідження

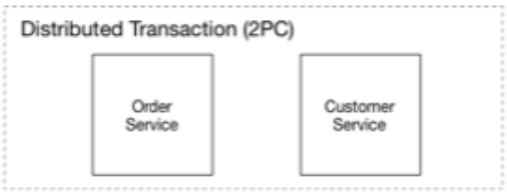
Лінійна адитивна згортка з ваговими коефіцієнтами

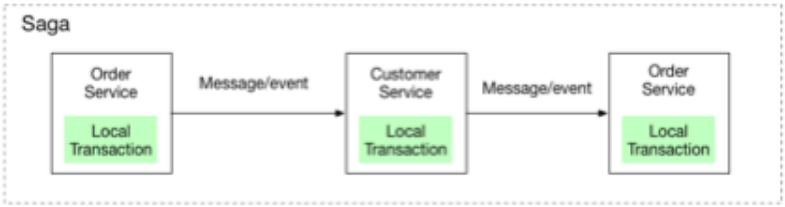
Метод / Критерій	Виконавче середовище	Сховища даних	Координація	Транзакційні властивості	Реалізації	Z*
Хореографія	1	1	1	1	0.67	0.9175
Паралельні пайплайни	1	1	1	1	0.33	0.8325
Оркестрація	1	1	0.5	1	0.67	0.7925
Двофазний коміт	1	1	0.5	1	0.67	0.7925
Модульний моноліт	0.5	0.5	0.5	1	1	0.75
Вагові коефіцієнти	0.2	0.1	0.25	0.2	0.25	4

Рисунок В.4 – Слайд - Аналіз та вибір методів обробки розподілених транзакцій для дослідження

Аналіз методів обробки розподілених транзакцій

Найбільш популярним патерном реалізації хореографії транзакцій є **SAGA** – послідовність локальних транзакцій.



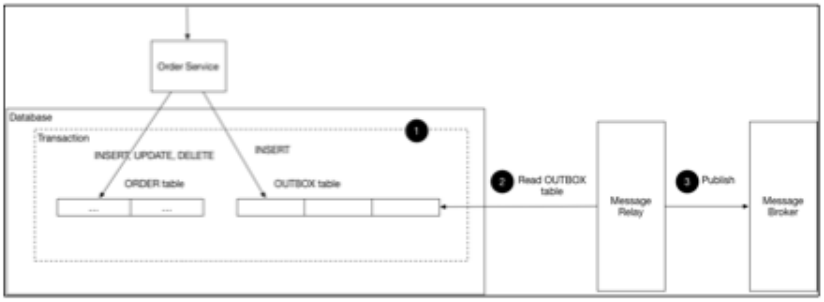


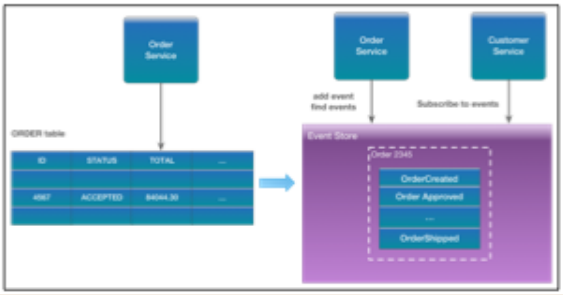
5

Рисунок В.5 – Слайд - Аналіз методів обробки розподілених транзакцій

Вибір шаблонів для дослідження і порівняння

Transactional Outbox - сервіс, який надсилає повідомлення, спочатку зберігає повідомлення в базі даних як частину транзакції, яка оновлює бізнес-об'єкти. Потім окремий процес надсилає повідомлення до брокера повідомлень.





Event Sourcing - бізнес-об'єкт зберігається шляхом зберігання послідовності подій, що змінюють його стан. Щоразу, коли стан об'єкта змінюється, до послідовності подій додається нова подія. Оскільки це одна операція, вона є атомарною за своєю суттю. Поточний стан об'єкта відновлюється шляхом відтворення його подій.

6

Рисунок В.6 – Слайд - Вибір шаблонів для дослідження і порівняння

Планування експериментального дослідження

- Обрати предметну область та побудувати моделі БД для дослідження.
- Спроектувати бізнес-транзакції, що будуть досліджені, та метрики порівняння.
- Налаштувати локальне середовище, інфраструктуру та контейнеризацію.
- Розробити програмний комплекс мікросервісів, реалізувавши кожну бізнес-транзакцію під кожен патерн, розробити додаткове ПЗ автоматизації експериментів та провести їх.
- Провести аналіз експериментів.

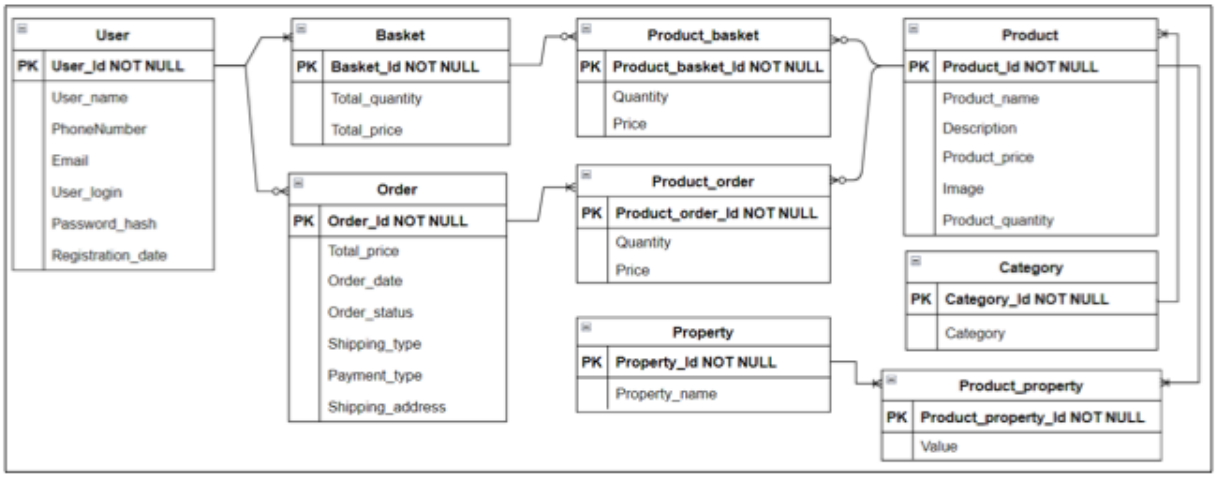
В якості метрик для дослідження були обрані:
-швидкість виконання запитів (мс);
-розмір журналу подій (Mb);
-кількість витраченної пам'яті (Mb);
-використання процесорного часу (%);
-пропускна здатність мережі I/O (Mb/Mb).

7

Рисунок В.7 – Слайд – Планування експериментального дослідження

Аналіз та моделювання предметної області

В якості предметної області була обрана область **електронної комерції** (e-commerce).



ER- діаграма предметної області

8

Рисунок В.8 – Слайд - Аналіз та моделювання предметної області

Рисунок В.9 – Слайд - Проектування БД

Рисунок В.10 – Слайд - Розробка архітектури системи

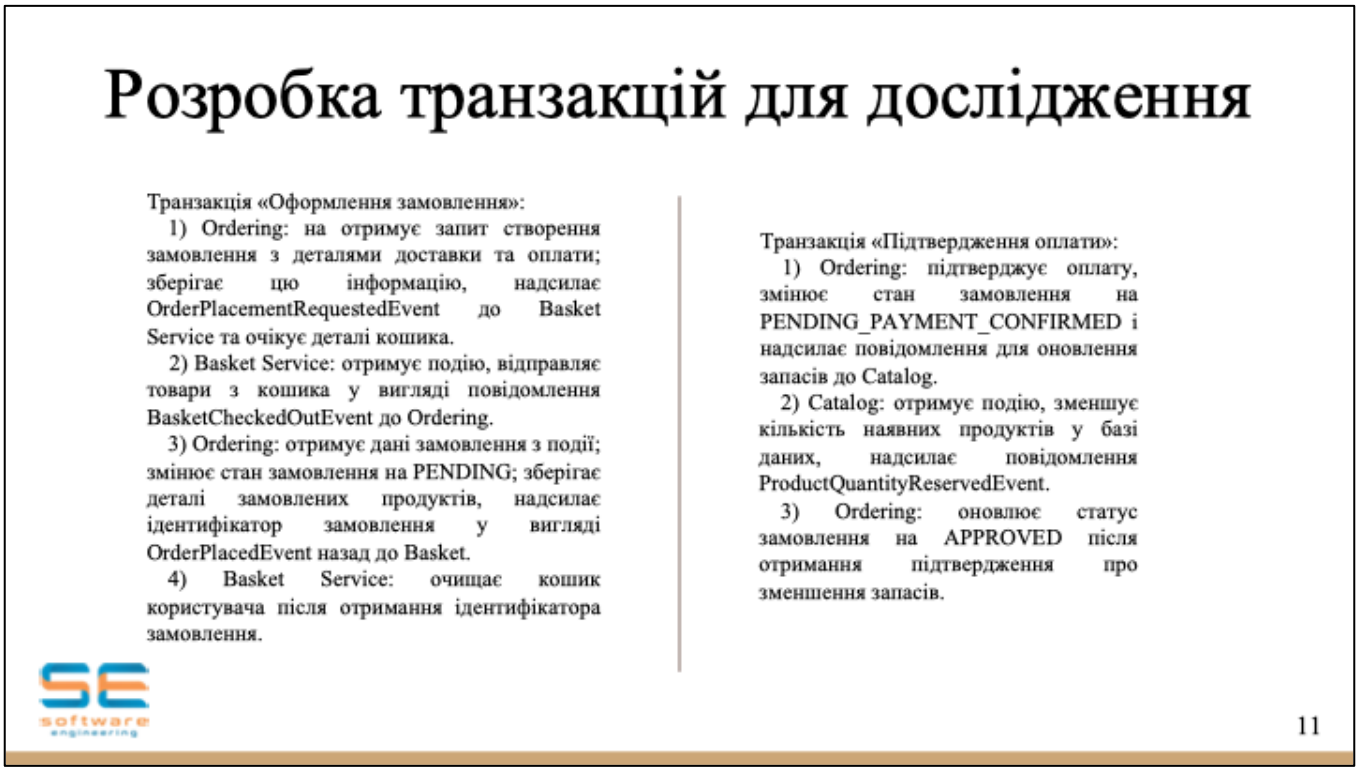


Рисунок В.11 – Слайд – Розробка транзакцій для дослідження

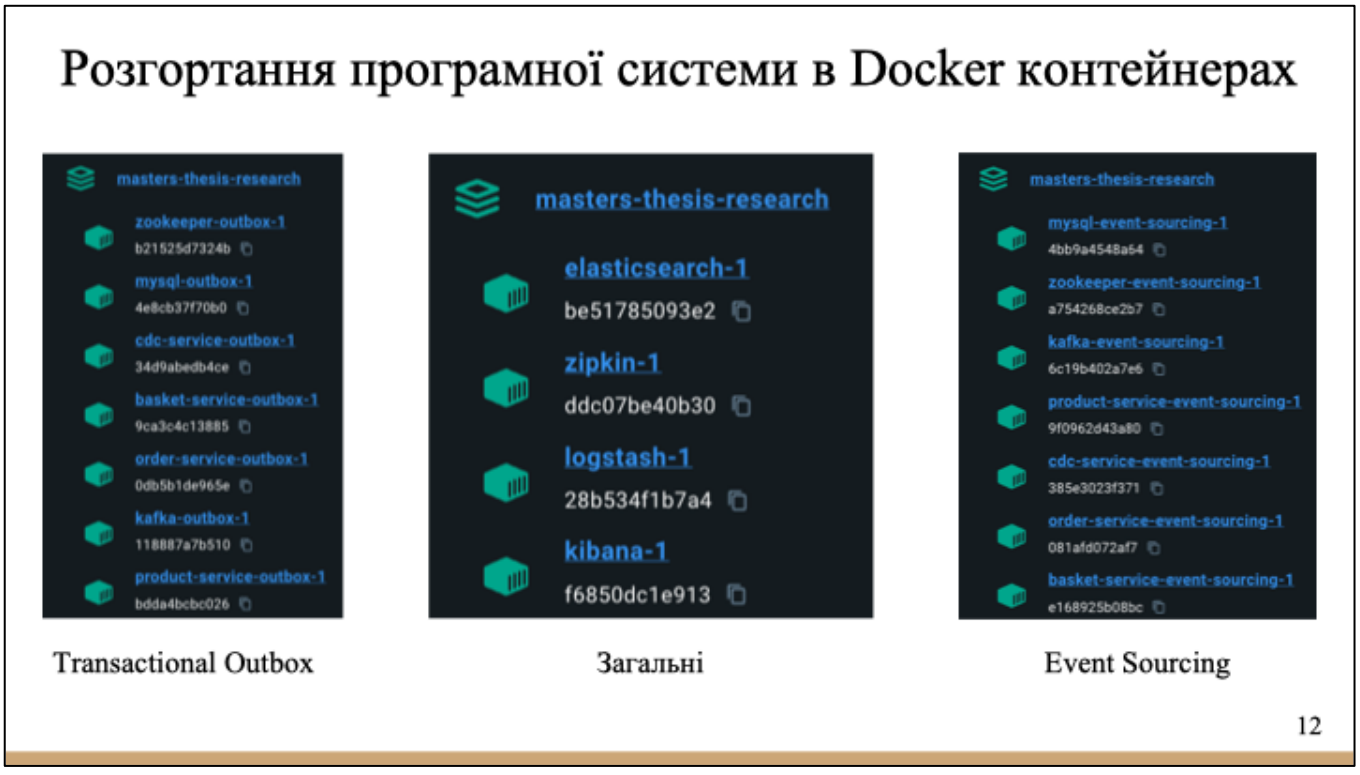


Рисунок В.12 – Слайд - Розгортання програмної системи в Docker контейнерах

Результати за метрикою «Час виконання», середній час для 100 послідовних запитів

Транзакція / Патерн	Transactional Outbox (мс)	Event Sourcing (мс)
«Додавання товару у кошик»	24,9	22,47
«Оформлення замовлення»	28,24	23,55
«Підтвердження оплати»	22,33	20,38
«Скасування замовлення»	23,11	19,64

13

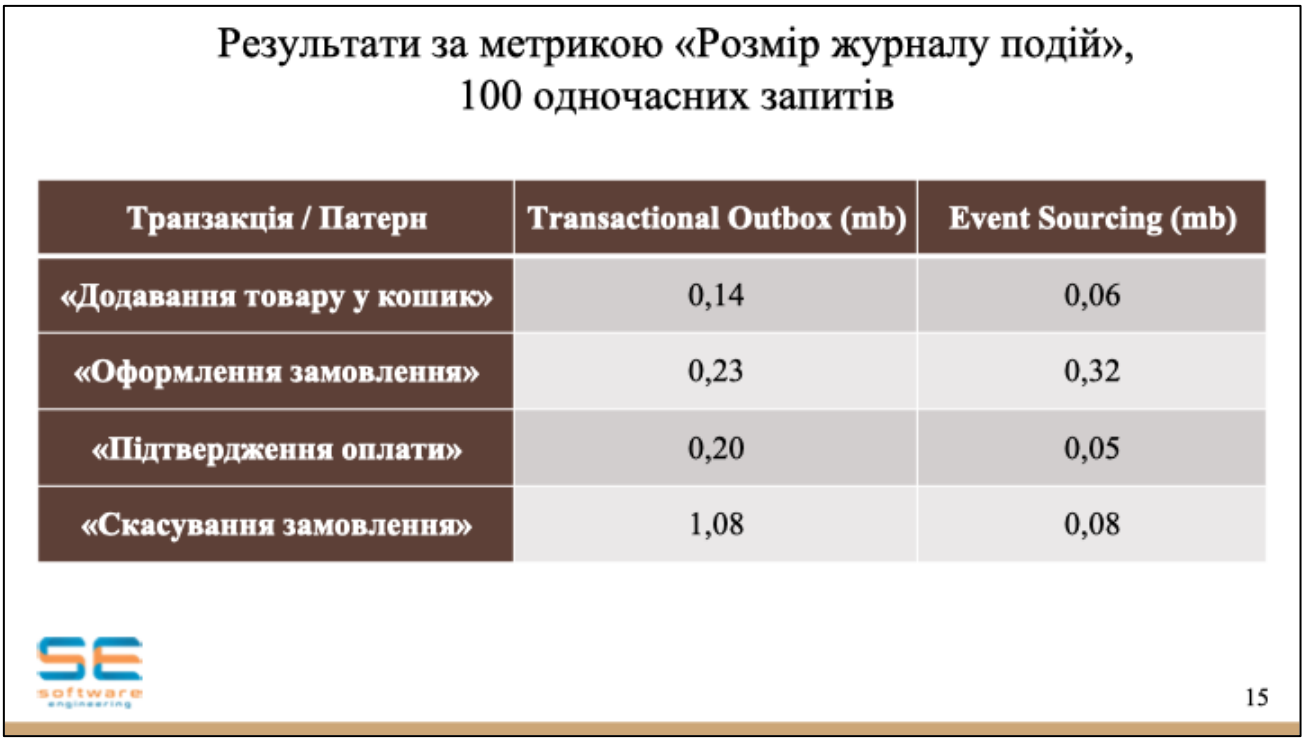
Рисунок В.13 – Слайд - Результати за метрикою «Час виконання», середній час для 100 послідовних запитів

Результати за метрикою «Час виконання», середній час для 100 одночасних запитів

Транзакція / Патерн	Transactional Outbox (мс)	Event Sourcing (мс)
«Додавання товару у кошик»	655,06	587,51
«Оформлення замовлення»	929,75	801,52
«Підтвердження оплати»	633,79	567,9
«Скасування замовлення»	557,8	571,9

14

Рисунок В.14 – Слайд - Результати за метрикою «Час виконання», середній час для 100 одночасних запитів



Результати за метрикою «Розмір журналу подій»,
100 одночасних запитів

Транзакція / Патерн	Transactional Outbox (mb)	Event Sourcing (mb)
«Додавання товару у кошик»	0,14	0,06
«Оформлення замовлення»	0,23	0,32
«Підтвердження оплати»	0,20	0,05
«Скасування замовлення»	1,08	0,08

15

Рисунок В.15 – Слайд - Результати за метрикою «Розмір журналу подій», 100 одночасних запитів

Результати за метрикою «Споживання оперативної пам’яті»,
100 одночасних запитів

Транзакція / Патерн	Transactional Outbox (mb)	Event Sourcing (mb)
«Додавання товару у кошик»	(B) + (C) = 3,31 + 1,97 = 5,28	(B) + (C) = 5,64 + 9,00 = 14,64
«Оформлення замовлення»	(O) + (B) = 4,94 + 2,80 = 7,74	(O) + (B) = 17,61 + 1,41 = 19,02
«Підтвердження оплати»	(O) + (C) = 2,14 + 1,25 = 3,39	(O) + (C) = 1,25 + 7,29 = 8,54
«Скасування замовлення»	(O) + (C) = 2,94 + 1,13 = 4,07	(O) + (C) = 6,06 + 11,02 = 17,08

16

Рисунок В.16 – Слайд - Результати за метрикою «Споживання оперативної пам’яті», 100 одночасних запитів

Результати за метрикою «Споживання процесорного часу», 100 одночасних запитів

Транзакція / Патерн	Transactional Outbox (%)	Event Sourcing (%)
«Додавання товару у кошик»	(B) + (C) = 33,3 + 9,7 = 43,0	(B) + (C) = 28,4 + 8,4 = 36,8
«Оформлення замовлення»	(O) + (B) = 41,9 + 12,8 = 54,8	(O) + (B) = 31,4 + 13,1 = 44,5
«Підтвердження оплати»	(O) + (C) = 21,7 + 9,2 = 30,9	(O) + (C) = 23,8 + 10,9 = 34,7
«Скасування замовлення»	(O) + (C) = 17,9 + 8,7 = 26,6	(O) + (C) = 23,3 + 8,5 = 31,8

17

Рисунок В.17 – Слайд - Результати за метрикою «Споживання процесорного часу», 100 одночасних запитів

Результати за метрикою «Пропускна здатність мережі», 100 одночасних запитів

Транзакція / Патерн	Transactional Outbox (mb)	Event Sourcing mb)
«Додавання товару у кошик»	1,07 / 1,20	1,22 / 1,13
«Оформлення замовлення»	0,91 / 0,99	1,35 / 1,14
«Підтвердження оплати»	0,83 / 1,23	1,69 / 1,35
«Скасування замовлення»	1,18 / 1.23	2,01 / 1,31

18

Рисунок В.18 – Слайд - Результати за метрикою «Пропускна здатність мережі», 100 одночасних запитів

Сформовані рекомендації

- В системах, де вимогою є швидкодія транзакцій, варто надати перевагу Event Sourcing. Кожна зміна стану сутності в базі є лише подією, яка швидко додається в кінець таблиці за константний час. Так само швидко можуть бути відтворені всі події для конкретної сутності при ефективній реалізації сховища подій.
- Event Sourcing добре підходить до транзакцій, де залучаються зв'язки 1:1; транзакції із залученням зв'язків 1:M відпрацьовували гірше у зв'язку з необхідністю публікації подій під кожен зв'язок, що було неоптимальним у контексті використання оперативної пам'яті та навантаження на мережу.
- Transactional Outbox - гарний вибір для систем, де є вимога в збереженні ресурсів виконання мікросервісу, а саме оперативної пам'яті віртуальної машини виконання та навантаження на мережу.
- При впровадженні шаблону Transactional Outbox, варто з обережністю ставитись до його використання в системах, де є вимога по оптимізації кількості місця, що займає БД на диску, та залучати додаткові зусилля на розробку механізму очищення непотрібних повідомлень з часом.
- Transactional Outbox рекомендується для систем, де важлива економія ресурсів оперативної пам'яті та мережевого трафіку; підходить для використання з реляційними СКБД, де критично важлива надійність збереження даних.
- Event Sourcing рекомендується використовувати в системах з високими вимогами до швидкодії транзакцій та необхідністю ретельного аудиту подій; підходить для систем з одноразовими зв'язками (один до одного) та ефективним використанням пам'яті.

19

Рисунок В.19 – Слайд - Сформовані рекомендації

Висновки

- Проведено аналіз та обрано методи обробки розподілених транзакцій для дослідження.
- Проведено планування експериментального дослідження.
- Спроектовано БД для області електронної комерції.
- Побудовано мікросервісну архітектуру, спроектовано бізнес-транзакції для експериментів.
- Розроблено середовище для проведення експериментів.
- Проведено експерименти, виконано аналіз їх результатів та розроблено рекомендації.
- Подано тези доповіді на 28 міжнародний молодіжний форум «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ В XXI ст»; підготовлено матеріали «Research of choreography methods for processing distributed transactions in a microservice architecture based on the SAGA pattern» (6 сторінок) для подачі на Скопус-конференцію «Problems of Infocommunications. Science and Technology» (PIC S&T'2024).

20

Рисунок В.20 – Слайд - Висновки

ДОДАТОК Г

Апробація результатів

УДК 004.45:004.65

ДОСЛІДЖЕННЯ МЕТОДІВ ОБРОБКИ РОЗПОДІЛЕНИХ ТРАНЗАКЦІЙ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Кузнецов Р. О.
Науковий керівник – к.т.н., доц. Мазурова О. О.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків,Україна
e-mail: roman.kuznetsov@nure.ua

Distributed systems more often become a robust architecture choice for a plenty of use cases in different domains. Despite of their advantages in improving scalability, reliability and maintainability of software systems, they provide additional level of complexity of handling business transactions in atomic and consistent way. This paper describes research that is aimed to analyze different practices in implementing distributed transactions handling patterns in microservices-based systems and provide recommendations what to use in specific situation.

Сьогодні все більша кількість програмних рішень в тих чи інших предметних галузях зводиться до використання мікросервісної архітектури через низку переваг: легше розподілення роботи між командами, незалежне розгортання сервісів, покращення відмовостійкості системи, облегшення показників масштабування тощо. Для більшості проектів впровадження мікросервісної архітектури стає де-факто стандартом при початковому плануванні на початку розробки.

Проте даний підхід несе з собою і ряд труднощів, зневага якими може нести катастрофічні наслідки для системи, один з яких це обробка розподілених транзакція та гарантія ACID властивостей. Все більше і більше розробників стикаються із труднощами в розумінні та виборі правильного підходу до опрацювання, фіксації та відкату транзакцій в розподілених масштабованих сервісах. Існують ряд патернів та підходів, які можуть бути по-різному використані в тих чи інших варіаціях мікросервісної архітектури та комунікації сервісів.

Отже, для отримання рішення цих проблем була поставлена задача виконати дослідження методів обробки розподілених транзакцій в мікросервісній архітектурі, що передбачає аналіз та вибір певних шаблонів реалізації розподілених транзакцій для порівняння, проектування запитів та транзакцій, та, відповідно, проведення експериментів та формування практично обґрунтованих рекомендацій.

Було проведено аналіз та обрано варіацію мікросервісної архітектури в комбінації синхронної та асинхронної взаємодії разом із патерном «DB per service» для ефективного моделювання розподілених транзакцій. Був обраний патерн Saga на основі хореографії як конкретний підхід для дослідження природи обробки розподілених транзакцій. Були обрані

340

Рисунок Г.1 – 28-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті» у Харкові, Україна. Перша сторінка

варіації реалізації атомарного оновлення бази та публікації повідомлень на основі Transactional Outbox та Event Sourcing.

Проведено планування експериментальних досліджень, а саме:

– в якості предметної області для побудови БД, на якій буде проводитися дослідження, обрано предметну галузь електронної комерції, яка легко може бути розкладена на мікросервіси із використанням розподілених бізнес транзакцій;

– розроблена строщена типова БД, яка включає в себе такі сутності, як Користувач, Корзина, Товар, Товар_Корзина, Замовлення, Товар_Замовлення, Категорія, Властивість, Товар_Властивість;

– в якості метрик для проведення замірів обрано час виконання, надійність, пропускну здатність, складність управління, консистентність даних, легкість розширення, обсяг оперативної пам'яті, затримку в мережі;

– заплановані також умови проведення експериментів, а саме проведення експериментів використовуючи локальну машину Mac Book Pro (M3 Pro, 12CPU, 36 Gb), використовуючи різні види навантаження, розмір даних, імітуючи затримку мережі, доступність сервісів.

На основі проаналізованих методів Saga, Transactional Outbox та Event Sourcing розроблено програмну системі в складі із трьох незалежних мікросервісів, кожен з яких має свою власну базу даних та комунікує з іншими сервісами, використовуючи хореографію транзакцій.

Розроблено програмні рішення для проведення експериментів, а саме допоміжний сервіс для імітації навантаження на систему, виконання транзакцій та фіксації статистики виконаних операцій.

Під час проведення експериментів очікується отримати результати, які дозволять глибше зрозуміти практичні аспекти застосування патернів Transactional Outbox і Event Sourcing та розробити критерії вибору найбільш підходящого методу в залежності від конкретних вимог.

Отримані в результаті рекомендації будуть корисні для розробників мікросервісних архітектур, архітекторів програмного забезпечення, інженерів, які працюють над проектами високої доступності і великого навантаження, де є потреба в забезпеченні консистентності даних.

В результаті чого будуть спрощені та прискорені такі процеси, як впровадження механізму забезпечення транзакційності в розподілених системах на основі мікросервісної архітектури.

Список використаних джерел:

1. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. 1st ed. O'Reilly Media, 2017. 611 p.

2. Richardson C. Microservices Patterns: With examples in Java. 1st ed. Manning. 2018. 520 p.

Рисунок Г.2 – 28-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті» у Харкові, Україна. Друга сторінка

Research of choreography methods for processing distributed transactions in a microservice architecture based on the SAGA pattern

Oksana Mazurova, Kuznetsov Roman, Perepychai Oleh
Department of Software Engineering
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine
oksana.mazurova@nure.ua, roman.kuznetsov@nure.ua, oleh.perepychai@nure.ua

***Abstract* — The paper presents the results of a study of choreography methods for processing distributed transactions for applications developed based on microservice architecture using the Transactional Outbox and Event Sourcing patterns of the SAGA pattern. An extended infological (ER) model is proposed for designing a microservice architecture and, accordingly, a distributed database. The results of designing a distributed database for microservice architecture in the field of e-commerce are presented. The results of experiments are analysed and recommendations for the use of the studied methods are formulated.**

Keywords — database, distributed transaction, choreography, ER-model, Event Sourcing, MySQL, SAGA, Transactional Outbox.

I. INTRODUCTION

Nowadays, with the active use of various Internet applications, including e-commerce systems, the issue of scaling such systems is important. In the direction of creating highly scalable systems, information systems based on microservice architecture (MA) are becoming increasingly common.

One of the important stages in the development of such systems is the design of distributed databases (DB) that underlie them, or the reengineering of existing DB [1]. The main issue when allocating parts of the database for separate microservices is to consider the data processing operations that make up the business logic of the software system. Thus, designing a distributed database (DDB) requires modelling queries and transactions that are necessary to support the business logic of a software system, which is not provided by the classical database design methodology [2]. The solution to this problem may be to extend the traditional ER model by adding elements such as transaction execution maps.

Aspects of physical integration and support of DB designed for MA, along with the correct processing of distributed transactions, also remain as important questions for developers. Developers of modern microservice information systems widely use the latest technologies and architectural patterns that facilitate the processing of distributed transactions [3]. The most popular approaches today are the use of event-driven architectures, Event Sourcing, CQRS, transaction orchestration and choreography, and the SAGA pattern [4]. The implementation of these technological solutions has a significant impact on the efficiency of the systems being developed.

Thus, to solve the problem of creating high-performance DDB, the task of studying methods of processing distributed transactions in the MA based on the use of certain patterns is relevant.

II. ANALYSIS OF BASIC RESEARCHES

In response to the challenges of processing rapidly growing volumes of data and transactions, today's MA is gaining popularity as a means of achieving scalability, flexibility, and speed of deployment. However, in addition to the advantages, developers face several difficulties and challenges caused by the peculiarities of this approach. It is necessary to adapt to specific approaches to infrastructure setup, centralised logging, communication, security, monitoring, distributed data processing, etc [5].

In the context of MA, distributed transaction management approaches play a key role in ensuring data integrity and consistency of business operations. In a distributed system, where each microservice may have its own database or even use different storage technologies, it is necessary to provide reliable mechanisms for data coordination and consistency between services [6]. Today, the choice of technologies and patterns for processing distributed transactions is often based on the principles expressed in the CAP and PACELC theorems [7] regarding the impossibility of simultaneously ensuring data consistency, availability, and partition tolerance in distributed systems.

Today, there are many approaches to event processing in Event-Driven architectures [8]: based on two-phase (2PC) or three-phase (3PC) commit [6], transaction choreography and orchestration, Event Sourcing and Transactional Outbox patterns.

Transaction orchestration and choreography mechanisms are widely used in distributed microservice systems. Both approaches can be implemented by the SAGA microservice system design pattern. Orchestration involves a centralised orchestrator that manages all microservices, but it involves additional synchronous calls that may not scale well under high load. Transaction choreography, on the contrary, has a decentralised model [9], where each microservice independently initiates and responds to events. Architectures that use transaction choreography patterns provide more flexible solutions because they allow each service to manage its transactions

978-x-xxxx-xxxx-x/20/$31.00 ©2024 IEEE

Рисунок Г.3 – Міжнародна Скопус-конференція «Problems of Infocommunications. Science and Technology» (PIC S&T’2024), Харків, Україна. Перша сторінка

autonomously, thereby increasing availability and resilience to network partitions [10].

Thus, the choice of microservices system architecture should be based on a detailed analysis of specific business requirements and technical capabilities, considering the specific goals of the project.

The importance of considering business requirements and modelling business processes is also noted by developers of distributed, in particular heterogeneous, DBs [11]. Existing approaches to modelling DDBs for MA, aimed at cutting off parts of the designed database monolith [5] for separate business logics, are rather informal and require the development of a single methodology like the methodology for designing relational databases [2]. This would allow to consider the peculiarities of existing business processes and to break down the database model monolith at earlier stages, and, accordingly, to consider the variety of data models that have emerged today, including NoSQL and NewSQL, during the logical design [12].

The appropriate stage for making a decision to divide a database into separate parts is the stage of infological (ER) modelling [2], which is based on modelling the entities and relationships of the subject area. The classical ER model is not aimed at modelling the elements of the business logic of the subject area, namely data processing operations (transactions, queries) that are important for DB partitioning. Extending such a model with data processing models would allow purposeful design of the DDB to implement efficient microservice transaction processing on its basis.

Therefore, designing and implementing an effective MA requires targeted design of the DDB for this particular architecture, as well as studying the effectiveness of methods for processing distributed transactions based on modern technologies and patterns such as SAGA.

III. Problem Statement

The purpose of this paper is to study the efficiency of Event Sourcing and Transactional Outbox distributed transaction processing templates based on the SAGA pattern in MA to develop qualitative recommendations for their use. To achieve this goal, the following tasks need to be solved:

- - to develop elements that would allow to extend the ER-model for use in the design of the DDB for MA;
- - design DDB for MA for the research;
- - plan experimental research, including the development of criteria for assessing the effectiveness of patterns and relevant transactions;
- - conduct an experimental research and develop recommendations for the use of selected patterns.

IV. Models and Methods

A. *Modelling data processing operations to extend the ER-model of DB*

In recent years, researchers have noted the need to use various semantic models [13], transaction execution maps, etc. to represent data processing operations when designing a database. Thus, a transaction execution map displays the sequence of CRUD operations that make up the essence of a transaction on database entities. It is proposed to extend transaction execution maps, starting now referred to as operation execution maps (OEM), to model queries, the distributed nature of which is also important to consider when designing a DDB.

Therefore, to ensure a cross-cutting approach to the design of the DDB for MA, it is proposed to extend the traditional ER model by integrating with the OEM, as well as taking into account the requirements of consistency, availability, partition tolerance, considered in the CAP-theorem [7], and other performance requirements for the following operations.

Thus, extended ER-model (EER) can be specified by a tuple $EER =< E, R, O >$, where E – a set of entities of a subject area; R – a set of relationships between entities; $O = \{O_k | k = \overline{1, p}\}$ – a set of data processing operations consisting of sets of queries and transactions.

Each operation O_k can be represented by a tuple $O_k =< OM_k, CAP_k, Fr_k, rT_k >$, where $OM_k = \|om_{ki}\|$ sets the execution map of the k-th operation; CAP_k sets the requirements for processing the operation in accordance with CAP theorem; Fr_k – estimated frequency of the operation; rT_k – requirements for the execution time of the operation.

Elements $om_{ki} = \{CRUD_{kij} | j = \overline{1, q}\}$ are specified by a sequence of CRUD-operators performed on database entities. The description of the CRUD-operator can be specified by a tuple

$$CRUD_{kij} =< poz_{kij}, CRUD\_Name_{kij}, ex\_\max_{kij} > \quad (1),$$

where poz_{kij} – the order in which the j-th operator is executed within the k-th operation; $CRUD\_Name_{kij} \in \{C, R, U, D\}$ – type of CRUD-operator; $ex\_\max_{kij}$ - the maximum number of instances of the i-th entity that can be processed by the corresponding operator.

The requirements for processing an operation can be specified by a vector $CAP_k = \|p_{kj}\|, j = \overline{1,3}$. Elements p_{kj} of the vector will take the value 1 if the requirements for processing the k-th operation contain, respectively, the requirement «(C)onsistency» for the component p_{k1}, requirement «(A)valability» for the component p_{k2}, requirement «(P)artitioning» for the component p_{k3}. In the absence of a relevant requirement, $p_{kj} = 0$.

For further analysis and design of the DDB model of individual transactions OM_k it is proposed to combine them into a common operations execution map (COEM), which can be presented in the form of a matrix $EOM = \|OM_k\| = \|om_{ki}\|$.

The proposed extension of the ER-model allows us to consider the process of designing a DDB in the world of traditional methodology with the stages of conceptual and

Рисунок Г.4 – Міжнародна Скопус-конференція «Problems of Infocommunications. Science and Technology» (PIC S&T’2024), Харків, Україна. Друга сторінка

infological modelling, which will be enhanced by the possibility of modelling the business logic of the subject area.

B. *Designing a distributed database for microservice architecture*

The e-commerce industry was chosen for the research, which is an area where the use of MA is quite popular. The analysis, conceptual and infological modelling of the subject industry were performed. A classic ER diagram was built using Burker and «Crow's foot» notations (see Fig. 1). For designing distributed, especially heterogeneous, databases, it is important not to display attributes on the diagram that can be used to model relationships, as is common for relational databases.

Based on the ER-model, an extended EER-model with operation maps was developed. The following data operations were considered for the business logic of working with the user's basket: O_1 – transaction of adding product to the basket; O_2 – transaction of order placement; O_3 – transaction for adjusting the amount of stock of products after payment of the order; O_4 – transaction to cancel an order in case of a payment error and others.

Figure 1 shows as EER with OEM O_1. O_1 consists of the following operators:

- operator <1, C, 1> adding (CREATE) a record of a new product to the basket (to the Product_Basket entity), which has a sequence number of 1, is essentially a CRUD-operator C (CREATE) and processes 1 instance of the entity (1 product);
- operator <2, R, 1> reading R (READ) product price from the Product entity for price validation.

The operation of adding an item to the basket can be performed up to 30,000 times per day. This is indicated by the corresponding number under the description of the first CRUD operator. Other characteristics of the operation, such as the requirements for its execution time and compliance with the CAP theorem, are not shown in the figure in order not to complicate the perception.

In general, for modelling and further analysis of operations that implement the business logic of an application, it will be more convenient to use the COEM, which can be visually represented in the form of a table shown in Figure 2. It contains a description of the operation (in this case, transaction) used during the study. It should be noted that the above transactions are designed in such a way that when studying patterns and templates for processing distributed transactions, it is possible to consider the features of processing transaction rollbacks and options when several microservices are involved in a transaction.

The extended EER, namely the COEM analysis (see Fig. 2), allowed us to distinguish the sets of entities most often used in joint operations into separate local models on which a separate microservice can be implemented, namely:

- the Basket and Product_Basket entities are involved in common transactions for adding, deleting and editing data, have the same requirements for their execution; accordingly, they can be allocated to the local Basket model;
- the Product, Category, Property and Product_Property entities are allocated to the Catalog model because they are involved in common operations (more often queries) to search and filter products and also have similar requirements for their execution;
- the User, Order and Product_order entities are allocated to the local Ordering model.

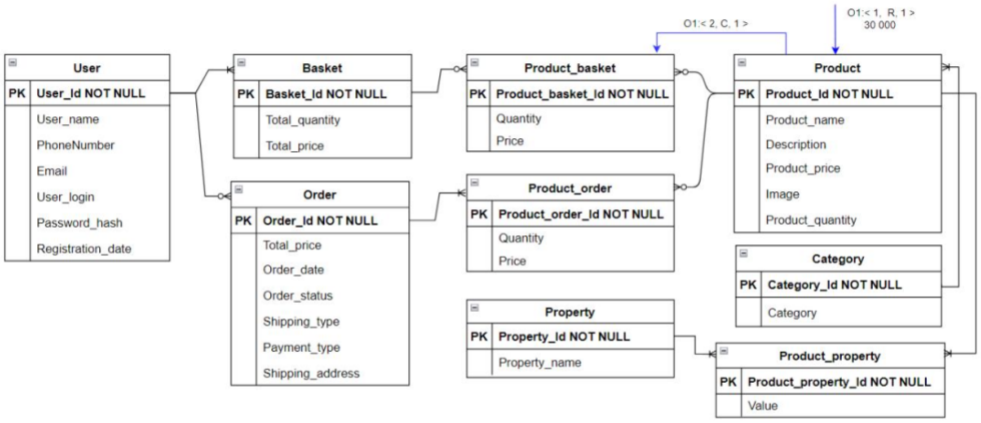


Fig. 1. The fragment of EER-diagram with operation execution map

Рисунок Г.5 – Міжнародна Скопус-конференція «Problems of Infocommunications. Science and Technology» (PIC S&T'2024), Харків, Україна. Третя сторінка

Operations	Characteristics of operations				Basket		Ordering			Catalog			
	C	A	P	Frequency (units/day)	Basket	Product_ Basket	Product_ Order	Order	User	Product	Category	Property	Product_ property
O1 - Adding a product to the basket	1	0	1	30 000		< 1, C, 1 >				< 2, R, 1 >			
O2 - Placing an order	1	1	0	5 000		< 2, R,10 > <5, D,10>	<3, C,10>	< 1, C, 1 > <4, U, 1 >					
O3 - Adjustment of the quantity of stock of goods after payment of the order	1	1	0	5000				< 1, U, 1 > < 3, U, 1 >		< 2, U, 10 >			
O4 - Order cancellation in case of payment error	1	1	0	1000			<1, R, 10>	<1, U, 1> < 4, U, 1>		<2, U, 10>			

Fig. 2. Fragment of common operation execution map

Thus, the analysis of the COEM allowed us to design a heterogeneous database for the e-commerce domain, the quality of which is confirmed by microservice solutions tested in practice. For this study, it was decided to use a homogeneous relational database. In accordance with the proposed DDB, the corresponding microservices Ordering, Basket and Catalog were implemented using the MySQL relational DBMS.

C. *Planning of the experimental research*

For the experimental research, two popular choreography implementations were chosen - the Transactional Outbox and Event Sourcing templates. They are used to ensure consistency through asynchronous data updates and event publishing, which allows the system to be fault-tolerant and efficiently distribute the load, significantly reducing the latency in data processing and improving system performance on a large amount of data [9].

Transactional Outbox is used in a fairly standard way: entities are stored as relational tables, but two infrastructure tables are added to publish event processing. Event Sourcing has a different approach to transaction processing, as any action on an entity is represented in the Event format. Fault tolerance and consistency between data updates is achieved through the use of events both for storing data and transmitting it in the form of messages to other services.

For the experimental research, an asynchronous interaction of business transactions (see Fig. 2) was designed. For example, the operation O1 "Adding a product to the basket" (for now and then, in the experiments, the transaction T1) consists of the following microservice operations:

- Basket: receives a request from the client to add a product to the basket, updates the data in its database, and then sends the ProductAddedToBasketEvent event to the Catalog to validate the product price;
- Catalog: receives the event, validates the price of the product, sends a response ProductBasketAdditionValidatedEvent if everything is fine; ProductBasketPriceHasChangedEvent if the price has changed, or ProductBasketAdditionValidationFailedEvent if the product does not exist at all back to the Basket;
- Basket: confirms the addition of an item to the user's basket, taking into account the data received, or compensates for changes by updating the value of items in the basket following the received data.

Transaction T2 «Placing an order»:

- Ordering: receives a request to create an order with delivery and payment details. Saves this information, sends an OrderPlacementRequestedEvent to the Basket Service to further receive the basket details;
- Basket Service: receives the event, sends the products from the basket as a message BasketCheckedOutEvent to Ordering;
- Ordering: receives order data from the event. Changes the order status to PENDING. Saves the details of the ordered products, sends the order identifier as an OrderPlacedEvent back to the Basket;
- Basket Service: clears the user's basket after receiving the order ID.

The transactions T3 of «Confirming the payment» and T4 of «Cancelling the order» were designed too (see Figure 2 for O3 and O4 operations, respectively).

To implement the transactions, Java, Spring, Kafka, Eventuate we chosed as programming tools, and a software package of designed e-commerce microservices was implemented. Each microservice is implemented in two variations - one using SAGA communications with the Transactional Outbox pattern, and the other using Event Sourcing.

For the experiments, a local Mac Book Pro machine was used with the following configuration: M3 Pro (12-core CPU with 6 performance cores up to 4.06 GHz and 6 efficiency cores up to 2.8 GHz), 36 Gb RAM. The service for the automated launch of a large number of transactions was launched on the local machine, but all microservices and their infrastructure (database servers, Kafka, Zookeeper, CDC service, Zipkin, Kibana) were deployed using Docker Compose for automated deployment of more than one container. The following configuration was used for Docker: CPU limit (cores) – 12; Memory limit (RAM, Gb) – 16; Swap (Gb) – 1; Virtual disk limit (Gb) – 64.

Separately, a JVM-level configuration was introduced for services and infrastructure components: Core Services Max Heap Size (Mb) – 64; Zookeeper Max Heap Size (Mb) – 64; Kafka Max Heap Size (Mb) – 192; CDC

Рисунок Г.6 – Міжнародна Скопус-конференція «Problems of Infocommunications. Science and Technology» (PIC S&T’2024), Харків, Україна. Четверта сторінка

service Max Heap Size (Mb) – 64. Limiting the maximum size of the JVM Heap is an important component for further experiments and their accuracy, because an unlimited Heap size means that as the number of objects in memory increases, the Heap size will grow until it reaches the host machine resource limit, which is suboptimal in the context of modern systems and can lead to Memory Leaks and OutOfMemoryError. In addition, with an unlimited Heap size, the Garbage Collector (GC) will be called much less frequently, which will also make the experiments more distant from real systems.

The service used for the experiments was designed to pre-populate with data and execute each transaction N times. For the experiments, it was decided to choose the value N = 100, which is a balance between peak load and low load.

During the experiments, we decided to collect data on the following metrics: T – the total execution time from the first request to the processing of the last event in the SAGA chain (ms); S – the size of the event log (Mb); M – the consumption of RAM (Mb); C – the consumption of CPU time (%); N – the network bandwidth, Network I/O (bytes).

All transactions were performed on "warmed-up" microservices and databases, i.e. with data allocated to the internal cache, pre-created framework components, a connection pools to the database, Kafka, etc.

V. Conducting Experiments and Its Conclusions

Let's consider the most interesting performance trends of the experiments conducted to study the designed transactions for Outbox and Event Sourcing. Measurements of the average execution time of 100 transactions in parallel mode are shown in Table 1.

TABLE 1. RESULTS FOR THE METRIC «EXECUTION TIME», AVERAGE TIME FOR THE 100 SIMULTANEOUS TRANSACTIONS

Transaction / Pattern	Transactional Outbox (ms)	Event Sourcing (ms)
T1	655,06	587,51
T2	929,75	801,52
T3	633,79	567,9
T4	557,8	571,9

We can see that in almost all cases, except for T4, Event Sourcing shows slightly better results, which is because when using this pattern, the event is intended to trigger the next handler and update the data. It is worth noting that Event Sourcing shows better results because it was the first cycle of filling the basket and creating an order based on it. All subsequent cycles would have to replay the events of all previous iterations, but by using the Snapshots mechanism, this problem of replaying unnecessary events would be solved [15].

An average transaction execution time of more than 500 ms can be a problematic factor for systems where low latency and high throughput are critical, where there is usually a requirement that a transaction should be executed in about 200-300 ms. This is due to the higher resource consumption in parallel processing, simultaneous running of processing threads, the cost of synchronising critical sections in the code, database and message broker, overheads on the locking mechanism and ensuring data consistency. But usually, for such systems, the SAGA pattern is not chosen because of the presence of Eventual Consistency.

Let's consider the results by the metric of the size of the generated event log (see Table 2).

TABLE 2. RESULTS FOR THE METRIC «EVENT LOG SIZE», 100 SIMULTANEOUS TRANSACTIONS

Transaction / Pattern	Transactional Outbox (мb)	Event Sourcing (мb)
T1	0,14	0,06
T2	0,23	0,32
T3	0,20	0,05
T4	1,08	0,08

As you can see, a larger event log in all cases except T2 is observed for Outbox, because Event Sourcing stores the entire state of entities in these events, and Outbox needs additional meta-information.

Let us consider the specifics of RAM consumption in the involved services (see Table 3). For the respective values, the sum of the RAM consumption of the microservices involved in the execution of the transaction was used. The abbreviations are as follows: (B) – Basket, (C) – Catalog, (O) – Ordering.

TABLE 3. RESULTS FOR THE METRIC «RAM CONSUMPTION», 100 SIMULTANEOUS TRANSACTIONS

Transaction / Pattern	Transactional Outbox (мb)	Event Sourcing (мb)
T1	(B) + (C) = 3,31 + 1,97 = 5,28	(B) + (C) = 5,64 + 9,00 = 14,64
T2	(O) + (B) = 4,94 + 2,80 = 7,74	(O) + (B) = 17,61 + 1,41 = 19,02
T3	(O) + (C) = 2,14 + 1,25 = 3,39	(O) + (C) = 1,25 + 7,29 = 8,54
T4	(O) + (C) = 2,94 + 1,13 = 4,07	(O) + (C) = 6,06 + 11,02 = 17,08

As seen from the results, Event Sourcing uses more RAM. When events are replayed, empty objects are created in memory on which these changes are performed. You can also see that in all transactions, Catalog uses much more memory. This is because when working with products, we need to create an event for each individual product, because an individual product is an aggregate and is also updated through a set of events.

Let's take a look at how the CPU time was consumed during transactions (see Table 4).

TABLE 4. RESULTS FOR THE METRIC «CPU CONSUMPTION», 100 SIMULTANEOUS TRANSACTIONS

Transaction / Pattern	Transactional Outbox (%)	Event Sourcing (%)
T1	(B) + (C) = 33,3 + 9,7 = 43,0	(B) + (C) = 28,4 + 8,4 = 36,8
T2	(O) + (B) = 41,9 + 12,8 = 54,8	(O) + (B) = 31,4 + 13,1 = 44,5
T3	(O) + (C) = 21,7 + 9,2 = 30,9	(O) + (C) = 23,8 + 10,9 = 34,7
T4	(O) + (C) = 17,9 + 8,7 = 26,6	(O) + (C) = 23,3 + 8,5 = 31,8

Рисунок Г.7 – Міжнародна Скопус-конференція «Problems of Infocommunications. Science and Technology» (PIC S&T’2024), Харків, Україна. П’ята сторінка

It can be seen that the most costly transactions in the previous metrics, such as T1 and T2, use fewer CPU instructions for Event Sourcing.

Regarding the last metric, network bandwidth, or the ratio of bytes received and sent by the service (I/O) through the network, the following observations were made: less load on the network was performed by transactions when using the Outbox pattern. The transmitted data was considered in REST requests to launch a transaction, database queries, and message transmission via a message broker. A larger number of events in Event Sourcing also leads to duplication of certain meta-information, such as entity identifiers, which is used more efficiently in Outbox and allows for less payload to be transmitted in the network.

Thus, we can draw conclusions and develop certain recommendations on the use of a particular method in a particular situation. In systems where transaction speed is a requirement, Event Sourcing should be preferred. Each change in the state of an entity in the database is just an event that is quickly added to the end of the table in a constant time. In addition, Event Sourcing is suitable for systems where there is a requirement for a clear audit of events – this requirement is easily covered by the fact that we have the ability to track to the smallest detail how the state of the entity has changed over time. At the same time, the pattern optimises the use of database memory, as it does not require a separate place to separate business data and published events.

Event Sourcing is well suited to transactions involving one-to-one relationships. As seen in the research results, transactions involving one-to-many connections performed worse due to the need to publish events for each connection, which was suboptimal in terms of memory usage and network load. Therefore, for systems where the use of memory resources is critical, it is worth choosing a different pattern.

Transactional Outbox is well suited for using relational DBMSs and will be a good choice for systems where there is a requirement to save microservice execution resources, namely the virtual machine's RAM and less network load. However, you should be cautious about using it in systems where there is a requirement to optimise the amount of space occupied by the database on the disc and engage in additional efforts to develop a mechanism for clearing unnecessary messages over time.

Both patterns showed approximately the same CPU usage, but they are still somewhat high. This is due to the large number of queries and events to process entity relationships. In this case, it is worth considering alternatives to transaction processing that will allow you to use not only relational databases, but also NoSQL databases such as MongoDB and Redis, which can significantly reduce CPU costs and improve system throughput.

VI. Conclusion

A number of scientific and practical results have been obtained in the field of designing and implementing DDB for MA, namely:

- the extension of the ER-model by means of maps of data processing operations was proposed, which allowed to use them as a basis for designing a DDB for MA;
- in the field of e-commerce, a DDB was designed for microservice architecture, which showed the practical value of the proposed approach;
- an experimental study of the effectiveness of choreography methods for processing distributed transactions for MA based on the use of the Transactional Outbox and Event Sourcing templates of the SAGA pattern was carried out, and recommendations for their use were formed.

References

[1] Filatov, V., Semenets, V. (2018), "Methods for Synthesis of Relational Data Model in Information Systems Reengineering Problems", In 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T). IEEE.

[2] Date C.J. Database Design and Relational Theory: Normal Forms and All That Jazz. – Apress, 2019. – 470 P. – ISBN 978-148-425-539-1.

[3] Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, 1st ed. – O'Reilly Media, 2017. – 611 p.

[4] Richardson C. Microservices Patterns: With examples in Java, 1st ed. – Manning, 2018. – 520 p.

[5] Newman S. Building Microservices: Designing Fine-Grained Systems, 2d ed. – O'Reilly Media, 2021. – 616 p.

[6] Chockler G., Gotsman A. Multi-shot distributed transaction commit. Distributed Computing. 2021. Vol. 34, no. 4. pp. 301–318. DOI: https://doi.org/10.1007/s00446-021-00389-4

[7] E. Brewer A certain freedom: thoughts on the CAP theorem / E. Brewer. - in Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing, New York, NY, USA. - 2010. - c. 335.

[8] Farias K., Lazzari L. Event-driven Architecture and REST Architectural Style: An Exploratory Study on Modularity. Journal of Applied Research and Technology. 2023. Vol. 21, no. 3. P. 338–351.

[9] Event-sourced, observable software architectures: An experience report / F. Alongi et al. Software: Practice and Experience. 2022

[10] Aydin S., Cebi C. B. Comparison of Choreography vs Orchestration Based Saga Patterns in Microservices. 2022 International Conference on Electrical, Computer and Energy Technologies (ICECET), Prague, Czech Republic, 20–22 July 2022

[11] Tanyansky S.S. Models, methods and information technologies for the integration of heterogeneous distributed databases. - Manuscript. Dissertation for the degree of Doctor of Technical Sciences for speciality 05.13.06 - Information Technologies. - Kharkiv National University of Radio Electronics, Kharkiv, 2011.

[12] Kuzochkina A., Shirokopetleva M., Dudar Z., (2018), Analyzing and Comparison of NoSQL DBMS. International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), pp. 560-564. DOI: https://doi.org/10.1109/INFOCOMMST.2018.8632133

[13] Filatov, V., Semenets, V., & Zolotukhin, O. (2020). Data mining in relational systems. Innovative Technologies and Scientific Solutions for Industries, (3 (13)), 65–76. https://doi.org/10.30837/itssi.2020.13.065

[14] Knock Out 2PC with Practicality Intact: a High-performance and General Distributed Transaction Protocol / Z. Lai et al. 2023 IEEE 39th International Conference on Data Engineering (ICDE), Anaheim, CA, USA, 3–7 April 2023. 2023. URL: https://doi.org/10.1109/icde55515.2023.00179

[15] Consistent retrospective snapshots in distributed event-sourced systems / B. Erb et al. 2017 International Conference on Networked Systems (NetSys), Gottingen, 13–16 March 2017. 2017. URL: https://doi.org/10.1109/netsys.2017.7903947

Рисунок Г.8 – Міжнародна Скопус-конференція «Problems of Infocommunications. Science and Technology» (PIC S&T'2024), Харків, Україна. Шоста сторінка

ДОДАТОК Д

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008:2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-22-2 (група)

Кузнецов Роман Олександрович

(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

12.06.2024

Рисунок Д.1 – Експертний висновок нормоконтролю пояснювальної записки