УДК 004.9
DOI https://doi.org/10.15589/znp2024.3(496).15

ENSURING FAULT TOLERANCE AND STABILITY OF BANKRUPTCY RISK ASSESSMENT INFORMATION SYSTEMS

ЗАБЕЗПЕЧЕННЯ ВІДМОВОСТІЙКОСТІ ТА СТАБІЛЬНОСТІ РОБОТИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОЦІНЮВАННЯ РИЗИКУ БАНКРУТСТВА

Artem P. Sinkovskyi
a.sinkovskyi@chdtu.edu.ua
ORCID: 0009-0009-8877-7351

Yurii V. Tryus
tryus@chdtu.edu.ua
ORCID: 0000-0002-0739-2065

А. П. Сіньковський,
аспірант

Ю. В. Триус,
д-р. пед. наук, канд. фіз.-мат. наук, проф.

Cherkasy State Technological University, Cherkasy
Черкаський державний технологічний університет, м. Черкаси

Abstract. This paper presents a comprehensive approach to ensuring the fault tolerance and stability of an information system for assessing the risk of enterprise bankruptcy. The research focuses on the development and implementation of advanced technologies and methods aimed at improving the reliability, scalability, and efficiency of the system under high loads and potential failures. A key aspect of the work is the introduction of automatic scaling, which allows the system to dynamically adapt to changes in load. This ensures optimal resource utilization and high performance, even with significant fluctuations in the number of requests. Special attention is paid to the implementation of an event-driven architecture with asynchronous logic. This approach allows for a significant increase in system throughput and reduces delays in request processing. The paper also presents the implementation of the Circuit Breaker pattern, which ensures the system's resilience to cascading failures. This allows the system to remain functional even with the partial unavailability of individual services. Considerable attention is paid to load balancing using Kubernetes. Different strategies for distributing traffic between the tiers, automatic disaster recovery mechanisms, and cluster configuration features to ensure high availability are considered. Practical recommendations for optimizing the deployment and management of containerized microservices are presented. The paper also covers the aspects of monitoring and logging in a distributed system, which are critical for rapid problem detection and troubleshooting. Approaches to centralized log collection and analysis, as well as methods for visualizing performance metrics, are presented. The practical significance of the work lies in the development of specific recommendations and technical solutions that can be used to improve the fault tolerance and stability of information systems for assessing bankruptcy risk. The proposed approaches can significantly improve the reliability and efficiency of such systems, which is critical for making accurate and timely financial decisions.
Key words: system design; fault-tolerance; microservices; information-system; bankruptcy; risk evaluation.

Анотація. У даній роботі представлено комплексний підхід до забезпечення відмовостійкості та стабільності роботи інформаційної системи оцінювання ризику банкрутства підприємств. Дослідження зосереджено на розробці та впровадженні передових технологій та методів, спрямованих на підвищення надійності, масштабованості та ефективності системи в умовах високих навантажень та потенційних збоїв. Ключовим аспектом роботи є впровадження можливості автоматичного масштабування, що дає можливість системі динамічно адаптуватися до змін у навантаженні. Особлива увага також приділена реалізації event-driven архітектури з використанням асинхронного підходу, що реалізований на стороні мікросервісів. Даний підхід дозволяє значно підвищити пропускну здатність системи та зменшити latency (затримки) при обробці запитів. У роботі також представлено впровадження паттерну Circuit Breaker, використання якого дозволяє системі залишатися функціональною навіть при частковій недоступності окремих сервісів. Значну увагу приділено балансуванню навантаженнь з використанням Kubernetes. Представлено практичні рекомендації щодо оптимізації розгортання та управління контейнеризованими мікросервісами. Робота також висвітлює аспекти моніторингу та логування в розподіленій системі, що є важливим для швидкого виявлення та усунення проблем. Представлено підходи до централізованого збору та аналізу логів, а також методи візуалізації метрик продуктивності. Практична значимість роботи полягає у розробці конкретних рекомендацій та технічних рішень, які можуть бути застосовані для підвищення відмовостійкості та стабільності роботи інформаційних систем оцінювання ризику банкрутства. Запропоновані підходи дозволяють значно підвищити надійність та ефективність таких систем, що є критичним для прийняття точних та своєчасних рішень.
Ключові слова: проектування; відмовостійкість; мікросервіси; інформаційна системи; банкрутство; оцінка ризику.

ПОСТАНОВКА ЗАДАЧІ

Забезпечення відмовостійкості та стабільності роботи інформаційної системи оцінювання ризику банкрутства є комплексною задачею, тому для вирішення даної задачі можна виокремити наступні кроки.

Крок 1. Проаналізувати архітектуру та компоненти існуючої інформаційно-аналітичної системи оцінювання ризику банкрутства з точки зору забезпечення відмовостійкості.

Крок 2. Визначити ключові фактори, що впливають на стабільність роботи системи, визначивши потенційні слабкі місця системи.

Крок 3. Реалізувати автоматичне масштабування мікросервісів в залежності від навантаження, з урахуванням визначених граничних значень для кожного мікросервісу, таких як завантаження центрального процесора, пам'ять та інші ресурси.

Крок 4. Використати event-driven архітектуру у комбінації з асинхронною логікою для проектування системи.

Крок 5. Спланувати механізми балансування навантаження між компонентами системи використовуючи технології Kubernetes [1].

Крок 6. Визначити механізми автоматичного відновлення після збоїв для критично важливих компонентів використовуючи патерн Circuit Breaker [2].

Крок 7. Налаштувати систему моніторингу та сповіщення, що буде інформувати про збої в роботі компонентів.

Крок 8. Розробити план забезпечення безперервності бізнес-процесів у випадку масштабних збоїв.

АНАЛІЗ ОСТАННІХ ДОСЛІДЖЕНЬ І ПУБЛІКАЦІЙ

Модель Альтмана, реалізована в Bloomberg Terminal [3], відома своєю надійністю та широким застосуванням. Однак, з точки зору відмовостійкості, система Bloomberg Terminal використовує розподілену архітектуру з множинними центрами обробки даних, що забезпечує високу доступність сервісу. Це дозволяє мінімізувати ризики простоїв та втрати даних при оцінці ризику банкрутства.

CreditEdge від Moody's Analytics [4] використовує хмарну інфраструктуру для забезпечення масштабованості та надійності. Система реалізує автоматичне балансування навантаження та резервне копіювання даних, що є критичним для стабільної роботи при оцінці кредитних ризиків.

Dun & Bradstreet [5] у своїх рішеннях з управління ризиками застосовує передові технології забезпечення безперервності бізнесу. Їхня система включає механізми швидкого відновлення після збоїв, що особливо важливо при роботі з великими обсягами даних про підприємства.

ВІДОКРЕМЛЕННЯ НЕВИРІШЕНИХ РАНІШЕ ЧАСТИН ЗАГАЛЬНОЇ ПРОБЛЕМИ

Незважаючи на прогрес у розробці відмовостійких систем оцінювання ризику банкрутства, залишається ряд невирішених проблем:

1. Забезпечення стабільної інтеграції та синхронізації даних з різних джерел в умовах нестабільного з'єднання.

2. Розробка та використання алгоритмів, здатних динамічно реагувати на зміни в структурі запитів та обсягах даних.

3. Підтримка узгодженості даних між різними елементами системи.

4. Розробка та використання методів швидкого та безшовного відновлення системи.

5. Визначення можливих сценаріїв поведінки системи при навантаженнях на основі історичних даних та поточного стану системи.

МЕТА ДОСЛІДЖЕННЯ

Мета дослідження полягає у розробці комплексного підходу до забезпечення відмовостійкості та стабільності роботи інформаційної системи оцінювання ризику банкрутства підприємств. Основна увага зосереджена на створенні механізмів, здатних ефективно протидіяти різноманітним викликам, таким як нестабільність мережевого з'єднання, високе навантаження, потенційні збої в роботі системи та інше.

Дослідження спрямоване на розробку рішень для оптимізації інтеграції даних, впровадження динамічного балансування навантаження, забезпечення консистентності даних та створення ефективних механізмів відновлення після збоїв. Кінцевою метою є підвищення надійності, продуктивності та стійкості системи оцінювання ризику банкрутства, що дозволить забезпечити безперервність бізнес-процесів та підвищити точність фінансових прогнозів.

МЕТОДИ, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об'єктом дослідження є інформаційна система оцінювання ризику банкрутства підприємств. Предметом дослідження є методи та технології забезпечення відмовостійкості та стабільності роботи інформаційної системи оцінювання ризику банкрутства. Методи дослідження включають системний аналіз, моделювання, методи розподілених обчислень а також теорії надійності та оптимізації.

Основний матеріал

Автоматичне масштабування мікросервісів є критично важливим для забезпечення стабільної роботи системи під високим навантаженням. Це дозволяє динамічно адаптувати ресурси відповідно до поточних потреб, забезпечуючи ефективне їх використання та підтримку високої продуктивності. Основним підходом до автоматичного масштабування є використання технології контейнеризації та оркестрації,

такої як Kubernetes. Kubernetes забезпечує автоматичне масштабування на основі метрик використання ресурсів, таких як CPU, RAM та інші. Це дозволяє системі швидко реагувати на зміни навантаження, автоматично додаючи або видаляючи екземпляри (Instance) мікросервісів залежно від потреб системи в конкретний момент часу [6]. Це також дозволяє розробленій інформаційній системі адаптуватися до умов, які постійно змінюються , що забезпечує високу продуктивність системи в цілому.

На рис. 1 представлена схема автоматичного масштабування в кластері Kubernetes (k8s cluster). Ось ключові елементи цієї системи масштабування [1]:

– master node – містить ETCD secrets та Helm values, які керують загальною конфігурацією кластера;

– worker nodes – вузли, які забезпечують розгортання та виконання pods (контейнерів);

– deployment – представляє розгортання додатку, яке включає pod 1 і pod 2, з можливістю масштабування до pod Max;

– pod Autoscaler – компонент, який автоматично регулює кількість pods у розгортанні;

– metrics service – служба, яка збирає метрики з pods та надає їх Autoscaler'y;

– metrics target utilization – визначає цільові показники використання ресурсів, на основі яких приймаються рішення про масштабування.

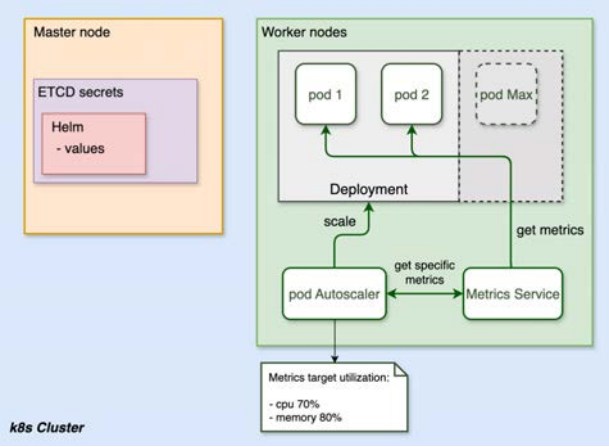

Рис. 1. Діаграма автоматичного масштабування системи оцінювання банкрутства підприємства

Також, одним із основних елементів балансування навантаження є налаштування pod health checks (перевірки стану контейнера), адже Kubernetes регулярно перевіряє стан контейнерів і автоматично виключає з балансування ті, які не проходять перевірку [7].

Реалізація механізму масштабування представлена на рис. 2, який показує процес масштабування для одного з ключових мікросервісів системи, що відповідає за формування опитування (Survey Builder Service). У даному випадку максимальна кількість контейнерів дорівнює 6, а мінімальна — 1. Дані про використання ресурсів контейнерами збирає саме Metrics Service (рис. 1). Основні цільові показники, на які pod Autoscaler звертає увагу, це CPU та memory. Якщо використання ресурсів перевищує або не досягає цільових показників, Autoscaler дає команду на масштабування Deployment (рис. 1). Deployment створює або видаляє pods відповідно до команд Autoscaler'a, підтримуючи оптимальне використання ресурсів. У результаті це забезпечує ефективне автоматичне масштабування, дозволяючи кластеру адаптуватися до змін навантаження та оптимізувати використання ресурсів.

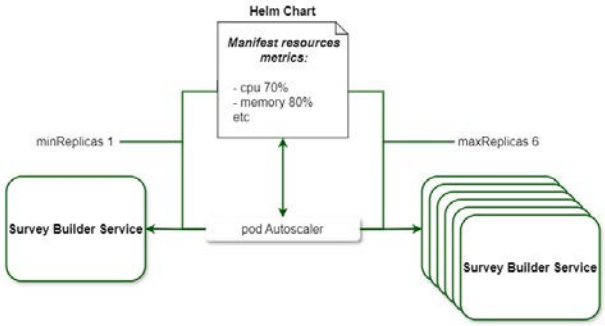

Рис. 2. Діаграма автоматичного масштабування Survey Builder Service

Для забезпечення контролю над помилками, що можуть виникати в процесі експлуатації програми, використовується Micrometer, Prometheus та Grafana. Ці інструменти дозволяють досліджувати та аналізувати метрики системи, забезпечуючи оперативну інформацію, представлену у зручному вигляді при виникненні проблем, пов'язаних, як з бізнес логікою, так і з ресурсами, які необхідні для функціонування системи.

Крім того, для швидкого виявлення та реагування на помилки використовується інструмент Sentry. Його використання дозволяє в реальному часі відслідковувати та реєструвати помилки, що виникають у різних частинах системи. Це сприяє швидкому виявленню та виправленню проблем.

Для ефективного та швидкого реагування на помилки було інтегровано в систему ELK stack. В свою чергу, використання trace id, яке було додане до системи для більш якісного відслідковування інформації між мікросервісами, дозволяє прискорити пошук помилок, надаючи можливість відстежувати шлях та причини їх виникнення, незалежно від того, в якому мікросервісі вони виникли. Це забезпечує комплексний аналіз даних від одного мікросервісу до іншого, сприяючи швидкому пошуку та, в результаті, усуненню проблем і покращенню стійкості системи в цілому [8].

Для забезпечення високої пропускної спроможності основних мікросервісів, в яких було виявлено

зниження продуктивності при високих навантаженнях, було здійснено перехід від синхронного до асинхронного підходу (рис. 3). Цей перехід був реалізований шляхом впровадження технології WebFlux для мікросервісів, написаних на Java. Для мікросервісів, розроблених на Kotlin, використовувався фреймворк Kotlin Coroutines [9]. Даний підхід є обгорткою event-driven архітектури [10].

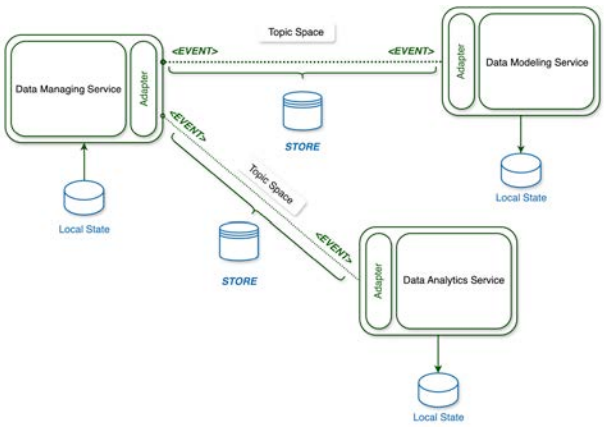

Рис. 3. Фрагмент event-driven архітектури сервісів системи

Kotlin Coroutines представляє собою потужний інструмент для асинхронного програмування, який дозволяє писати асинхронний код у послідовному стилі [9]. Це значно спрощує розробку та підтримку складних асинхронних операцій, роблячи код більш читабельним та менш схильним до помилок. Coroutines дозволяють ефективно управляти потоками виконання, що особливо важливо в умовах високонавантажених систем.

Однією з ключових переваг Kotlin Coroutines є можливість легко масштабувати додаток без значного збільшення споживання ресурсів. На відміну від традиційних потоків, coroutines дозволяють створювати тисячі паралельних операцій без суттєвого навантаження на систему. Це особливо важливо для мікросервісної архітектури (рис. 4), де ефективне управління ресурсами є критичним фактором. Використання Kotlin Coroutines також дозволило реалізувати більш гнучкі механізми обробки помилок та затримок. Завдяки вбудованим можливостям структурованої конкурентності, стало можливо більш ефективно керувати життєвим циклом асинхронних операцій, що призвело до підвищення стабільності та надійності системи в цілому.

Це дозволило значно розвантажити обчислювальні ресурси системи, зменшити затримки в обробці запитів та підвищити загальну ефективність використання процесорного часу. Використання асинхронних методів обробки дозволило більш ефективно керувати паралельними процесами, що призвело до зниження часу очікування для користувачів та покращення часу відгуку системи.

Крім того, перехід на асинхронні механізми обробки дозволив краще масштабувати мікросервіси, забезпечуючи стабільну роботу навіть при значному збільшенні навантаження. Завдяки Kotlin Flow [9] і WebFlux [11], система отримала можливість обробляти більшу кількість запитів одночасно, що значно підвищило її пропускну здатність і надійність.

Використання Kotlin Flow також спростило реалізацію механізмів кешування та буферизації даних [9]. Це дозволило оптимізувати використання ресурсів системи та зменшити навантаження на базу даних, що особливо важливо при роботі з великими обсягами фінансової інформації.

Впровадження паттерну Circuit Breaker є ключовим елементом у забезпеченні стійкості інформаційної системи оцінювання ризику банкрутства до каскадних відмов [2]. Цей паттерн дозволяє системі залишатися функціональною навіть при частковій недоступності окремих сервісів.

Circuit Breaker постійно відстежує успішність запитів до кожного мікросервісу. Для контролю ввімкнення механізму circuit breaker були визначені порогові значення для кількості невдалих запитів або часу відповіді, після досягнення яких ініціюється процес його активації.

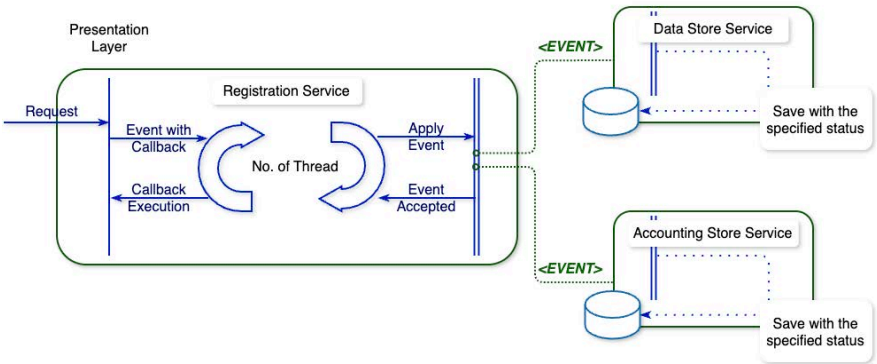

Рис. 4. Діаграма асинхронної взаємодії між сервісами на прикладі процесу реєстрації користувача в системі

Реалізація circuit breaker в розробленій системі базується на бібліотеці Resilience4j, яка добре інтегрується з Spring Boot та надає гнучкі можливості налаштування. Для кожного мікросервісу налаштовано окремий екземпляр Circuit Breaker з параметрами, оптимізованими під специфіку роботи цього сервісу.

Наприклад, для сервісу побудови опитування (Survey Builder Service) встановлено наступні параметри:

– Поріг відмов – 50% запитів протягом 1 хвилини.
– Час очікування у відкритому стані – 1 хвилина.
– Кількість дозволених запитів у напіввідкритому стані – 5.

Використання Circuit Breaker патерну дозволило значно підвищити стійкість системи до збоїв. Під час тестування імітувалися різні сценарії, включаючи повну недоступність окремих сервісів та мережеві затримки. У всіх випадках система змогла продовжити роботу, надаючи користувачам базову функціональність та поступово відновлюючись при нормалізації ситуації.

Впровадження асинхронного підходу дозволило підвищити продуктивність окремих сервісів, проте це також ускладнило процес виявлення помилок (Debugging Process). Для розв'язання цих проблем було внесено важливі покращення в систему та впроваджено інструменти, що запропоновані в [11]:

Крок 1. Використання ELK (Elasticsearch, Logstash, Kibana) з кореляційними ідентифікаторами (trace id) для кожного запиту допомагає відстежувати повний шлях виконання запиту через всі мікросервіси та ідентифікувати проблемні місця.

Крок 2. Інтеграція Zipkin надає можливість візуалізувати виконання запитів в асинхронному середовищі та виявляти слабкі місця або помилки в логіці обробки.

Крок 3. Створення спеціалізованих тестових середовищ, де можливо відтворювати реальні сценарії роботи системи з асинхронною обробкою, допомагає проводити глибинне тестування без впливу на продуктивну систему.

Крок 4. Інтеграція Sentry дозволила в реальному часі відслідковувати та реєструвати помилки, що виникають в різних частинах системи. Це сприяє швидкому виявленню та виправленню проблем.

Крок 5. Використання Jprofiler дозволяє глибше аналізувати продуктивність мікросервісів для оптимізації їх роботи.

Крок 6. Інтеграція Micrometer з Prometheus і Grafana надає можливість проводити детальний моніторинг і візуалізацію метрик продуктивності, що додатково сприяє оптимізації системи.

Розподілені транзакції є критичним аспектом у забезпеченні цілісності даних в мікросервісній архітектурі. В рамках забезпечення цілісності даних було впроваджено механізм управління розподіленими транзакціями на основі патерну Saga, зокрема його хореографічної варіації. Патерн Saga дозволяє координувати складні бізнес-процеси, які охоплюють кілька мікросервісів, забезпечуючи при цьому можливість скасування змін у разі виникнення помилок на будь-якому етапі [12]. Хореографічний підхід дозволяє ефективно керувати транзакціями, що охоплюють кілька мікросервісів, забезпечуючи при цьому можливість відкату змін у разі виникнення помилок.

Хореографічний патерн Saga не має центрального оркестратора, кожен мікросервіс самостійно виконує свою частину транзакції та ініціює наступний крок через події. Це дозволяє зменшити залежність від центрального компонента, підвищуючи надійність та масштабованість системи.

Основні етапи роботи хореографічної Saga:

– перший мікросервіс виконує свою частину бізнес-логіки та генерує подію для наступного мікросервісу;
– кожен наступний мікросервіс, отримавши подію, виконує свою частину транзакції та генерує подію для наступного мікросервісу;
– у разі помилки будь-який мікросервіс може ініціювати виконання компенсаційних дій для відкату змін, виконаних попередніми.

Хореографічний патерн Saga особливо корисний для систем, де обробка даних відбувається у географічно розподілених центрах. Такий підхід дозволяє координувати складні бізнес-процеси, забезпечуючи узгодженість даних навіть у разі збоїв.

Система оцінювання ризику банкрутства підприємства включає роботу з кількісними та якісними показниками. Saga координує наступні кроки:

– отримання кількісних та якісних показників за допомогою опитувань;
– збір фінансових показників;
– агрегація та попередня обробка даних;
– застосування моделей для оцінки ризиків банкрутства;
– перевірка результатів;
– формування звіту.

Якщо на будь-якому етапі виникає помилка, Saga ініціює компенсаційні дії для вже виконаних кроків, забезпечуючи узгодженість даних у всій системі.

Saga є досить гарним вибором для забезпечення консистентності транзакцій. Відсутність централізованого оркестратора зменшує точку відмови та підвищує надійність. Кожен мікросервіс незалежно виконує свою частину транзакції, що дозволяє ефективно масштабувати систему, а також легко адаптується до змін у бізнес-логіці та інфраструктурі.

Для системи оцінювання ризику банкрутства спроектовано оптимальну хореографію розподілених транзакцій (рис. 5).

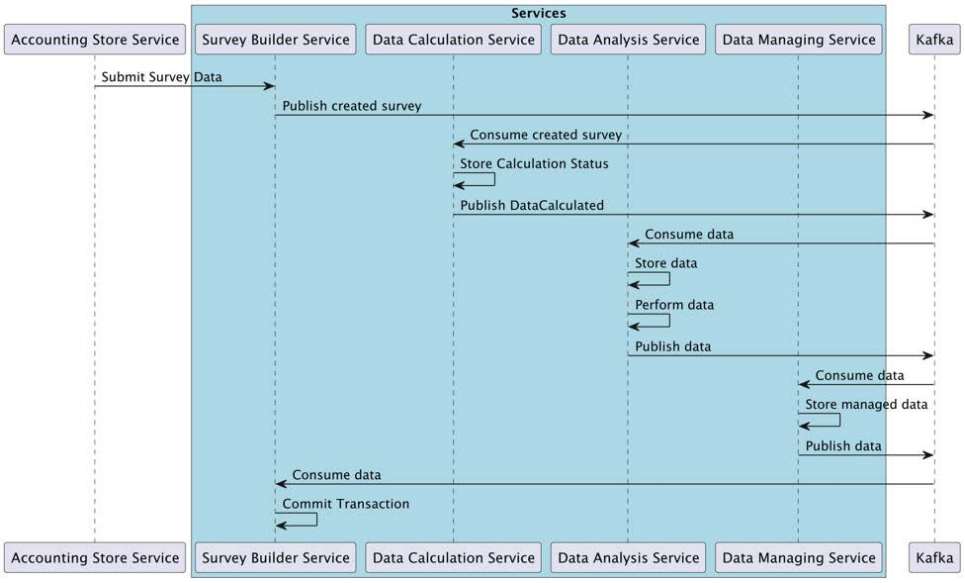

Рис. 5. Спроектована діаграма розподілених транзакцій системи за підходом Saga хореографічної

ОБГОВОРЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Реалізація асинхронного підходу ускладнила процес виявлення помилок. Для подолання цих труднощів було здійснено кілька важливих удосконалень та впроваджено інструменти, серед яких використання ELK (Elasticsearch, Logstash, Kibana) з кореляційними ідентифікаторами (trace id) для кожного запиту, інтеграція Zipkin для візуалізації виконання запитів в асинхронному середовищі, створення спеціалізованих тестових середовищ, інтеграція Sentry для відстеження та логування помилок у реальному часі, використання JProfiler для аналізу продуктивності мікросервісів, а також інтеграція Micrometer з системами Prometheus та Grafana для моніторингу та візуалізації показників ефективності.

Реалізація патерну Circuit Breaker є ключовим аспектом у забезпеченні стійкості інформаційної системи оцінки ризиків банкрутства до каскадних збоїв. Ця модель дозволяє системі залишатися функціональною навіть тоді, коли окремі сервіси частково недоступні.

Наявність розподілених транзакцій є важливим аспектом забезпечення цілісності даних в архітектурі мікросервісів. В рамках цього було впроваджено механізм управління розподіленими транзакціями на основі патерну Saga, а саме його хореографічної варіації. Патерн Saga дозволяє координувати складні бізнес-процеси, що охоплюють декілька мікросервісів, забезпечуючи при цьому можливість відкату змін у разі виникнення помилок на будь-якому етапі.

ВИСНОВКИ

Проведене дослідження дозволило досягти значного прогресу у розвитку інформаційних систем оцінювання ризику банкрутства. Впровадження різних стратегій та інструментів, а саме, таких як автоматичне масштабування, керована подіями архітектура з асинхронною логікою, патерн Circuit Breaker та балансування навантаження за допомогою Kubernetes, значно підвищило надійність, продуктивність та відмовостійкість системи вцілому.

Всі ці елементи сприяють підвищенню надійності, продуктивності та стійкості системи оцінювання ризику банкрутства, що забезпечує безперервність критично важливих бізнес-процесів і покращує точність фінансових прогнозів.

REFERENCES

[1] Luksa, M. (2018). Kubernetes in Action. First Edition. Manning. 624 p.

[2] Richardson, C. (2018). Microservices Patterns: With Examples in Java. First Edition. Manning. 520 p.

[3] Altman Model (Bloomberg Terminal). Отримано з https://www.bloomberg.com/professional/solution/bloomberg-terminal.

[4] CreditEdge (Moody's Analytics), отримано з https://moodysanalytics.com/product-list/creditedge.

[5] Dun & Bradstreet (Рішення з управління ризиками), отримано з https://www.dnb.com/.

[6] Hossen, M. R., & Islam, M. A. (2023). Practical efficient Microservice Autoscaling. ACM SIGMETRICS Performance Evaluation Review, 50(4), 50–52. https://doi.org/10.1145/3595244.3595262

[7] Lercher, A., Glock, J., Macho, C., & Pinzger, M. (2024). Microservice API evolution in practice: A study on strategies and challenges. Journal of Systems and Software, 215, 112110. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85194951657&doi=10.1016%2fj.jss.2024.112110&partnerID=40&md5=2870b557e40e129d4298150788d62b1d

[8] Sabharwal, N., & Pandey, P. (2020). Monitoring Microservices and Containerized Applications. https://doi.org/10.1007/978-1-4842-6216-0

[9] Elizarov, R., Belyaev, M., Akhin, M., & Usmanov, I. (2021). Kotlin Coroutines: Design and implementation. Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. https://doi.org/10.1145/3486607.3486751

[10] Zuki, S., Mohamad, R., & Saadon, N. (2024). Containerized event-driven Microservice architecture. Baghdad Science Journal, 21(2(SI)), 0584. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85186251434&doi=10.21123%2fbsj.2024.9729&partnerID=40&md5=c12fff81fae6a4d5985a484cb504dcdd

[11] Heckler, M. (2021). Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications. 1st Edition. O'Reilly Media. 486 p.

[12] Soshi, M., & Maekawa, M. (1997). The saga security system: A security architecture for open distributed systems. Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems. https://doi.org/10.1109/ftdcs.1997.644703

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Luksa, M. (2018). Kubernetes in Action. First Edition. Manning. 624 p.

[2] Richardson, C. (2018). Microservices Patterns: With Examples in Java. First Edition. Manning. 520 p.

[3] Модель Альтмана (Bloomberg Terminal). Отримано з https://www.bloomberg.com/professional/solution/bloomberg-terminal.

[4] CreditEdge (Moody's Analytics), отримано з https://moodysanalytics.com/product-list/creditedge.

[5] Dun & Bradstreet (Рішення з управління ризиками), отримано з https://www.dnb.com/.

[6] Hossen, M. R., & Islam, M. A. (2023). Practical efficient Microservice Autoscaling. ACM SIGMETRICS Performance Evaluation Review, 50(4), 50–52. https://doi.org/10.1145/3595244.3595262

[7] Lercher, A., Glock, J., Macho, C., & Pinzger, M. (2024). Microservice API evolution in practice: A study on strategies and challenges. Journal of Systems and Software, 215, 112110. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85194951657&doi=10.1016%2fj.jss.2024.112110&partnerID=40&md5=2870b557e40e129d4298150788d62b1d

[8] Sabharwal, N., & Pandey, P. (2020). Monitoring Microservices and Containerized Applications. https://doi.org/10.1007/978-1-4842-6216-0

[9] Elizarov, R., Belyaev, M., Akhin, M., & Usmanov, I. (2021). Kotlin Coroutines: Design and implementation. Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. https://doi.org/10.1145/3486607.3486751

[10] Zuki, S., Mohamad, R., & Saadon, N. (2024). Containerized event-driven Microservice architecture. Baghdad Science Journal, 21(2(SI)), 0584. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85186251434&doi=10.21123%2fbsj.2024.9729&partnerID=40&md5=c12fff81fae6a4d5985a484cb504dcdd

[11] Heckler, M. (2021). Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications. 1st Edition. O'Reilly Media. 486 p.

[12] Soshi, M., & Maekawa, M. (1997). The saga security system: A security architecture for open distributed systems. Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems. https://doi.org/10.1109/ftdcs.1997.644703

Дата надходження статті до редакції: 21.06.2024
Дата затвердження статті до друку: 04.07.2024