УДК

Олег Кузьмич[1], Максим Сенів[2]

[1] Національний Університет Львівська Політехніка, Львів, Україна, ORCID: 0000-0002-9749-6385

[2] Національний Університет Львівська Політехніка, Львів, Україна, ORCID: 0000-0003-1044-4628

Аналіз наявних методів і засобів забезпечення відмовостійкості мікросервісного програмного забезпечення

Анотація: Проаналізовано літературні джерела, в яких досліджено методи та засоби забезпечення відмовостійкості мікросервісного програмного забезпечення. Встановлено, що сучасна індустрія програмного забезпечення стрімко розвивається, що призводить до збільшення складності систем та вимог до їх відмовостійкості. З'ясовано, що мікросервісна архітектура пропонує переваги у масштабованості та гнучкості, але водночас створює нові виклики у забезпеченні безперебійної роботи через розподілений характер та численні потенційні точки відмови. Оцінено вплив різних класифікацій відмов (миттєві, переривчасті, постійні) на вибір та ефективність методів забезпечення відмовостійкості. Охарактеризовано закономірності розвитку методів забезпечення відмовостійкості, включаючи реактивні методи (повторні спроби, вимикачі ланцюгів), які спрямовані на відновлення після збою, проактивні методи (балансування навантаження, реплікація), що прагнуть запобігти збоям шляхом розподілу навантаження та створення резервних копій, а також методи тестування (функціональне тестування, інженерія хаосу), які дозволяють виявляти потенційні проблеми до їх виникнення в робочому середовищі. Детально проаналізовано переваги та недоліки кожного з цих методів, виявивши їх сильні та слабкі сторони, а також області застосування. Досліджено, що хоча ці методи ефективні в певних ситуаціях, жоден з них не є

універсальним рішенням для всіх типів відмов. Це підкреслює необхідність у комплексному підході, який би враховував специфіку мікросервісної архітектури, вимоги до системи та типи можливих відмов. Визначено ключові проблеми, що потребують подальшого дослідження, такі як розробка комплексних підходів, що враховують специфіку мікросервісної архітектури, автоматизація процесів виявлення та виправлення помилок, а також зниження вартості впровадження методів забезпечення відмовостійкості. Запропоновано перспективні напрями досліджень, зокрема розробку нових методів та інструментів, які враховують особливості мікросервісної архітектури та забезпечують високий рівень відмовостійкості за оптимальної вартості, а також дослідження методів раннього виявлення потенційних проблем, таких як безперервне функціональне з використанням інтеграції помилок.

Ключові слова: мікросервісна архітектура; надійність; розподілені системи; тестування; методи забезпчення відмовостійкості

Oleh Kuzmych[1], Maksum Seniv[2]

[1] Lviv Polytechnic National University, Lviv, Ukraine, ORCID: 0000-0002-9749-6385

[2] Lviv Polytechnic National University, Lviv, Ukraine, ORCID: 0000-0003-1044-4628

Analysis of existing methods and tools for ensuring the fault tolerance of microservice software

Abstract: It was found that the rapid evolution of the software industry has led to increased system complexity and heightened demands for fault tolerance, particularly in microservice architectures that offer scalability and flexibility but introduce new challenges due to their distributed nature and numerous potential failure points. This study analyzes existing literature on methods and tools for ensuring fault tolerance in

microservice software. We examine various failure classifications, including transient failures (short-lived and self-correcting), intermittent failures (recurring sporadically), and permanent failures (requiring intervention to resolve), and how these classifications influence the choice and efficacy of different fault tolerance strategies. The analysis encompasses reactive methods (retries, circuit breakers) aimed at recovery after failure, proactive methods (load balancing, replication) that strive to prevent failures through load distribution and redundancy, and testing methods (functional testing, chaos engineering) that identify potential issues before they manifest in production. We delve into the strengths, weaknesses, and applicability of each method, revealing that while effective in specific scenarios, no single method provides a universal solution for all failure types. Consequently, a comprehensive approach considering microservice architecture specifics, system requirements, and potential failure types is essential. Key areas for further research include developing integrated approaches tailored to microservice architectures, automating error detection and correction processes, and reducing the cost of implementing fault tolerance methods. This study contributes to the field by highlighting promising research directions, such as creating novel methods and tools that address microservice-specific challenges and provide high fault tolerance at optimal cost, as well as investigating early detection methods for potential issues, such as continuous functional testing with error integration.

Keywords: microservice architecture; reliability; distributed systems; testing; methods of ensuring fault tolerance

Вступ/Introduction

Сучасна індустрія програмного забезпечення стрімко еволюціонує, реагуючи на експоненціальне зростання обсягів інформації та вимог до функціональності, масштабованості та відмовостійкості систем. Мікросервісна архітектура стала відповіддю на ці виклики, дозволяючи створювати складні

програми як сукупність незалежних сервісів. Однак, розподілений характер мікросервісів і збільшення кількості потенційних точок відмови суттєво ускладнюють забезпечення їхньої безперебійної роботи. У науковій літературі активно досліджуються різноманітні підходи до забезпечення відмовостійкості програмного забезпечення. Проте, більшість існуючих методів та інструментів не враховують специфіку мікросервісної архітектури, а ті, що адаптовані, часто виявляються надмірно ресурсоємними. Це створює актуальну проблему пошуку ефективних та економічних рішень для забезпечення відмовостійкості мікросервісів. Актуальність дослідження зумовлена критичною важливістю безперебійної роботи програмного забезпечення в сучасному цифровому світі. Простої систем можуть призводити до значних фінансових втрат, втрати даних, порушення роботи підприємств та навіть загрожувати життю людей. Тому розробка нових підходів та удосконалення існуючих до забезпечення відмовостійкості мікросервісів має не лише наукову, а й важливу практичну цінність.

Об'єктом дослідження є процес забезпечення відмовостійкості мікросервісного програмного забезпечення.

Предметом дослідження є методи та засоби забезпечення відмовостійкості мікросервісного програмного забезпечення.

Метою роботи є провести систематичний огляд останніх публікацій, які мають відношення до теми методів забезпечення відмовостійкості мікросервісного програмного забезпечення, визначити їх переваги та недоліки, що дасть змогу розробити нові чи вдосконалити існуючі методи, які б дозволили знизити накладні витрати та підвищити ефективність процесу.

Для досягнення зазначеної мети визначено такі основні *завдання дослідження:* описати й охарактеризувати особливості забезпечення відмовостійкості в мікросервісному програмному забезпеченні, проаналізувати та визначити переваги і недоліки наявним методів забезпечення відмовостійкості, виявити перспективні напрями майбутніх досліджень для

вдосконалення процесу забезпечення відмовостійкості мікросервісного програмного забезпечення.

Аналіз останніх досліджень та публікацій

Розподілений характер мікросервісної архітектури та велика кількість потенційних точок відмови значно ускладнюють забезпечення безперебійної роботи програмного забезпечення, адже відомо, що чим більше в системі елементів, тим більше можливих комбінацій дефектів, які спричиняють виникнення помилок, які в свою чергу призводять до відмов [23]. цьому дослідженні було проведено аналіз 20 наукових робіт, присвячених різним аспектам забезпечення відмовостійкості мікросервісів.

У роботах [1, 2] розглянуто концепцію реактивних мікросервісів, які використовують асинхронну комунікацію та неблокуючий ввід/вивід для підвищення масштабованості та відмовостійкості. Хоча цей підхід демонструє потенціал у підвищенні продуктивності та стійкості до збоїв, його впровадження вимагає ретельного проектування та використання спеціалізованих інструментів.

Згідно з роботою [3], методи забезпечення відмовостійкості можна розділити на реактивні, проактивні та стійкі. Реактивні методи, такі як повторні спроби [4] та вимикачі ланцюгів [6, 7], спрямовані на відновлення після виникнення збою. Проактивні методи, такі як балансування навантаження [8] та реплікація [5, 9], прагнуть запобігти збоям шляхом розподілу навантаження та створення резервних копій. Стійкі методи, що використовують машинне навчання та штучний інтелект, динамічно прогнозують та запобігають збоям у мінливих середовищах.

Робота [9] пропонує більш детальну класифікацію механізмів відмовостійкості, виділяючи чотири ключові області: надлишковість даних та захист, стійкість системи і сервісу, моніторинг і відновлення, а також експлуатаційні та проектні практики. Ця класифікація дозволяє систематизувати

різноманітні підходи та інструменти, що використовуються для забезпечення відмовостійкості мікросервісів.

У роботах [12, 13] розглядаються питання тестування мікросервісів. Автори [12] пропонують фреймворк MicroHive для автоматизованого функціонального та надійнісного тестування, який дозволяє виявляти помилки та причинно-наслідкові зв'язки у ланцюгах збоїв. У статті [13] досліджується проблема інтеграційного тестування безсерверних систем та пропонується удосконалений підхід, який дозволяє значно скоротити час тестування.

Інженерія хаосу (chaos engineering) [14] є перспективним підходом до забезпечення відмовостійкості, який передбачає навмисне внесення помилок у систему для перевірки її здатності до відновлення. Цей підхід дозволяє виявити приховані проблеми та покращити стійкість системи до збоїв, проте вимагає ретельного планування та контролю.

У дослідженнях [15, 16] вивчається використання платформи Kubernetes та методу вимикачів для підвищення надійності мікросервісів. Автори вказують на необхідність розробки моделі, яка б дозволила визначати оптимальні методи забезпечення відмовостійкості для конкретних мікросервісних застосунків.

У роботі [18] порівнюються два підходи до внесення помилок у систему: підхід посіву помилок (модель Міллса) та підхід ін'єкції помилок. Автори дослідження [19] пропонують набір інструментів FaultSee для відтворюваного внесення помилок у розподілені системи, що дозволяє краще оцінити вплив несправностей. У статті [20] аналізуються переваги та недоліки використання контрольних точок в розподілених обчисленнях, а також проводиться порівняння існуючих алгоритмів контрольних точок.

Аналіз літератури показав, що проблема відмовостійкості мікросервісного програмного забезпечення є комплексною та багатогранною. Незважаючи на значний прогрес у цій області, залишається багато відкритих питань, зокрема щодо розробки комплексних підходів, які враховують специфіку мікросервісної

архітектури та вимоги до системи, а також щодо створення інструментів для автоматизованого виявлення та виправлення помилок.

Результати дослідження та їх обговорення/Research result and their discussion

Було визначено, що мікросервісна архітектура, незважаючи на свої переваги, ускладнює забезпечення відмовостійкості через розподілений характер та велику кількість потенційних точок відмови. Це призводить до виникнення різноманітних відмов, що можуть суттєво вплинути на роботу системи.

Відмовостійкість - здатність програмного продукту або компонента працювати, як передбачено, незважаючи на наявність апаратних або програмних помилок. [21]

Забезпечення відмовостійкості є критично важливим завданням, оскільки відмови можуть призвести до значних фінансових втрат, втрати даних, порушення роботи підприємств та навіть загрожувати життю людей. Проте робитип це складно, оскільки відмов є велика кількість і вони всі різні. Їх можна прокласифікувати за тривалістю відмови на:

Миттєві відмови (Transient faults):

- Це короткочасні збої, які швидко зникають самі по собі.
- Приклади: тимчасова втрата мережевого з'єднання, короткочасне перевантаження сервера, збій в роботі кешу.
- Такі відмови зазвичай можна вирішити за допомогою повторних спроб (retries) або механізмів автоматичного відновлення.

Короткочасні відмови (Intermittent faults):

- Це збої, які виникають періодично, але не постійно.

- Причини можуть бути різними: програмні помилки, неправильна конфігурація, апаратні проблеми.
- Виявлення та усунення таких відмов може бути складнішим, оскільки вони не завжди легко відтворюються.

Постійні відмови (Permanent faults):

- Це збої, які зберігаються протягом тривалого часу і не зникають самі по собі.
- Приклади: апаратний збій, критична помилка в коді, некоректні дані.
- Такі відмови зазвичай вимагають втручання людини для усунення, наприклад, перезапуску сервісу, відновлення даних з резервної копії або виправлення коду [22].

Вирішенням проблеми з відмовостійкості в мікросервісних системах мали слугувати методи забезпечення відмовостійкості. Вони поділяються на:

Реактивні методи: Спрямовані на відновлення системи після виникнення збою. До цієї категорії належать такі методи, як повторні спроби (retries) та вимикачі ланцюгів (circuit breakers). Повторні спроби дозволяють обробляти тимчасові збої, автоматично повторюючи запит до сервісу, який не відповів. Вимикачі ланцюгів запобігають каскадним збоям, тимчасово відключаючи доступ до проблемного сервісу.

Проактивні методи: Прагнуть запобігти збоям шляхом розподілу навантаження та створення резервних копій. До цієї категорії належать такі методи, як балансування навантаження (load balancing) та реплікація. Балансування навантаження розподіляє запити між кількома екземплярами сервісу, запобігаючи перевантаженню окремих екземплярів. Реплікація створює резервні копії даних та сервісів, що дозволяє швидко відновити роботу системи у разі збою.

Методи тестування: Дозволяють виявляти потенційні проблеми та вразливості до їх виникнення в робочому середовищі. До цієї категорії належать

такі методи, як функціональне тестування, тестування навантаження та інженерія хаосу (chaos engineering). Функціональне тестування перевіряє коректність роботи окремих функцій та сервісів. Тестування навантаження дозволяє оцінити продуктивність системи під високим навантаженням. Інженерія хаосу передбачає навмисне внесення помилок у систему для перевірки її здатності до відновлення.

Наведу переваги та недоліки кожного з цих методів в таблитчці нижче.

Переваги та недоліки підходів забезпечення відмовостійкості

Публікація	Назва підходу	Переваги	Недоліки
4	Retries (Повторні спроби)	● Простота: Легко реалізувати та налаштувати. ● Гнучкість: Може використовуватися з будь-якими типами мікросервісів.	● Реактивність: Проблеми виявляються лише після їх виникнення, що може призвести до простою. ● Непередбачуваність: Неможливо гарантувати, що відновлення буде успішним
7	Circuit Breakers (Вимикачі ланцюгів)	● Локалізація: Обмежує вплив збою на інші мікросервіси. ● Захист даних: Запобігає поширенню проблем на інші частини системи	● Складність: Потребує додаткової логіки та інфраструктури. ● Можливість втрати даних: Якщо не синхронізовані дані, може призвести до втрати даних
2	Bulkheads (Заслони)	Локалізація: Обмежує вплив збою на інші мікросервіси. Захист даних: Запобігає поширенню проблем на інші частини системи	● Складність: Потребує додаткової логіки та інфраструктури. ● Можливість втрати даних: Якщо не синхронізовані дані, може призвести до втрати даних
8	Load Balancing (Балансування навантаження)	● Покращена доступність: шляхом розподілу трафіку балансування навантаження запобігає єдиній точці збою. Якщо екземпляр мікросервісу виходить з ладу, інші можуть впоратися з навантаженням, мінімізуючи час простою. ● Масштабованість: балансування навантаження дозволяє легко	● Додаткова складність: впровадження та підтримка балансувальника навантаження ускладнює систему. ● Єдина точка відмови: хоча це покращує відмовостійкість у мікросервісах, сам балансувальник навантаження може стати єдиною точкою відмови. Для самого балансувальника навантаження

		горизонтально масштабувати. У міру збільшення трафіку можна додавати додаткові екземпляри мікросервісу, розподіляючи навантаження та зберігаючи продуктивність. ● Підвищена продуктивність: розподіляючи трафік, балансувальники навантаження можуть запобігти перевантаженню окремих екземплярів, що призводить до швидшого часу відповіді та більш плавної взаємодії з користувачем. ● Моніторинг працездатності: балансувальники навантаження можуть контролювати працездатність екземплярів мікросервісу. Несправні екземпляри можна видалити з пулу, щоб вони не вплинули на загальну продуктивність.	потрібні конфігурації високої доступності. ● Дорого: балансування навантаження створює певні накладні витрати, оскільки вимагає прийняття рішень щодо маршрутизації та керування пулами підключень. ● Обмежена відмовостійкість: лише балансування навантаження може не впоратися з усіма типами збоїв. Він не вирішує таких проблем, як помилки програмного забезпечення в самому екземплярі мікросервісу. ●
9, 11	Redundancy and Replication	● Дуже висока доступність: резервування мінімізує час простою за рахунок створення резервних копій або копій даних. Якщо основний компонент виходить з ладу, вторинний бере на себе роботу, забезпечуючи безперервність обслуговування. ● Узгодженість даних: реплікація може допомогти підтримувати узгодженість даних у примірниках залежно від вибраної стратегії реплікації. Це стає критично важливим для програм, які покладаються на синхронізацію даних у реальному час	● Дуже дорого: резервування вимагає додаткових ресурсів (серверів, сховищ) для підтримки реплік, що призводить до вищих операційних витрат. ● Складність: керування декількома копіями збільшує складність. Синхронізація даних у репліках вимагає належного впровадження та може призвести до додаткової затримки. ● Обмежена відмовостійкість: резервування може не вирішити всі збої. Такі проблеми, як помилки програмного забезпечення в самому екземплярі мікросервісу, можуть впливати на всі репліки.
	Моніторинг	● Профілактика: Дозволяє виявити проблеми до їх виникнення.	● Складність: Потребує додаткової інфраструктури та інструментів. ● Витрати: Може потребувати значних ресурсів.

		• Швидке реагування: Можливість вжити заходів до того, як проблема призведе до збою	
12, 13	Функцірнальне тестування	• Профілактика: Дозволяє виявити та виправити помилки до їх випуску в продакшн. • Підвищення якості: Покращує загальну якість програмного забезпечення	• Час: Потребує значних часових витрат. • Вартість: Може потребувати додаткових ресурсів.
14	Тестування хаосу	• Покращена відмовостійкість: виявляючи приховані слабкі місця, хаос-тестування допомагає розробникам зміцнити свої системи, щоб більш витончено справлятися з неочікуваними збоями. • Раннє виявлення проблем. Тестування хаосу може виявити проблеми до того, як вони виникнуть у виробництві, запобігаючи збоям і впливу на користувачів. • Підвищена впевненість: успішне проходження випробувань хаосу створює впевненість у здатності системи справлятися з реальними збоями. • Покращена спостережуваність: Тестування хаосу допомагає виявити прогалини в системах моніторингу та оповіщення, що призводить до швидшого виявлення та вирішення реальних проблем	• Збої та час простою: тестування хаосу може порушити поточні операції та потенційно призвести до простою для деяких користувачів. Дуже важливо ретельно планувати та проводити тести, щоб мінімізувати вплив. • Складність: впровадження тестування хаосу та керування ним вимагає досвіду та планування. Вибір правильних сценаріїв і інструментів може бути складним завданням. • Вартість: інвестиції в інструменти для тестування хаосу та кваліфікований персонал збільшують загальну вартість розробки. • Пізнє використання: тестування хаосу виконується на пізньому етапі розробки програмного забезпечення • Обмежений обсяг: тестування хаосу не може змоделювати всі можливі сценарії збою. Важливо поєднувати його з іншими методами тестування
20	(Контрольні точки)	• Простіше відновлення: у разі збою мікросервіс можна відновити до раніше відомого справного стану, що спрощує процес відновлення.	• Підвищена складність: впровадження логіки контрольних точок ускладнює сам мікросервіс. • Накладні витрати на продуктивність: створення контрольних точок може

		• Швидший перезапуск: у порівнянні з відновленням усього стану мікросервісу, перезапуск із контрольної точки може бути набагато швидшим. • Зменшена втрата даних: контрольні точки дозволяють відновити певний момент часу з мінімальною втратою даних, залежно від того, як часто створюються контрольні точки.	призвести до навантаження на продуктивність мікросервісу. • Проблеми узгодженості даних: підтримка узгодженості даних між мікросервісами може бути складною під час використання контрольних точок, особливо якщо контрольні точки не синхронізовані між службами. • Можливість втрати даних: втрата даних може статися, якщо контрольні точки не створюються досить часто.

Обговорення результатів дослідження

Аналіз літератури та визначення переваг та недоліків виявив кілька ключових проблем, які ще не є вирішені наявними методами:

1. *Потреба в комплексному підході*: Більшість досліджень зосереджені на окремих методах забезпечення відмовостійкості, таких як повторні спроби, вимикачі ланцюгів або реплікація. Проте, для досягнення високого рівня відмовостійкості мікросервісної системи необхідно застосовувати комплексний підхід, який враховує різні типи відмов та особливості конкретної системи [5].
2. *Складність використання*: Науковці стверджують, що немає чіткого опису, які методи слід використовувати в тих чи інших місцях, сервісах та етапах життєвого циклу пз, що не дозволяє побудувати мікросервісну систему з оптимальними показниками відмовостійкості та накладними витратами [10]
3. *Недостатність автоматизації:* Багато ефективних існуючих методів вимагають ручного втручання для виявлення та виправлення помилок. Це може бути неефективним та призводити до затримок у відновленні системи. Необхідно розробляти інструменти та підходи, які дозволяють автоматизувати ці процеси.

4. *Специфіка мікросервісної архітектури*: Багато методів забезпечення відмовостійкості були розроблені для монолітних застосунків і не враховують специфіку мікросервісної архітектури, такої як розподіленість, незалежність сервісів та складність взаємодій. Необхідні подальші дослідження щодо адаптації та розробки методів, які враховують особливості мікросервісів.
5. Ціна: Методи які забезпечують найкращу відмовостійкість, наприклад, реплікація, резервування, балансування навантаження є складними та надзвичайно дорогими в реалізації.

В майбутніх дослідженнях я планую спробувати знайти вирішення цих проблем. Потенційними шляхами можуть бути:

1. Розробка комплексного підходу, який поєднує різні методи забезпечення відмовостійкості, щоб покрити різні типи відмов. Також комплексний підхід повинен дозволяти розробникам обирати оптимальні комбінації методів та параметрів для досягнення найкращого балансу між відмовостійкістю та накладними витратами. Це може включати створення рекомендацій або інструменту для вибору методів на основі аналізу характеристик мікросервісної системи.
2. Розробка протоколу, який чітко опише, які методи слід використовувати в тих чи інших місцях, сервісах та етапах життєвого циклу ПЗ. Протокол може містити рекомендації щодо вибору методів залежно від типу сервісу, його критичності, вимог до продуктивності тощо. Це має дозволити знизити складність використання методів.
3. Автоматизація методів забезпечення відмовостійкості та тестування, включаючи їх в конвеєр CI/CD. Це може включати автоматичне масштабування сервісів, автоматичне відновлення після збоїв, автоматичне тестування відмовостійкості. Такий підхід

має дозволити пришвидшити процес забезпечення відмовостійкості та здешевити його.

4. Покращення методу балансування навантаження, адаптувавши його до специфіки мікросервісного програмного забезпечення. Це може включати розробку нового алгоритму балансування навантаження для мікросервісів, який враховує не тільки навантаження на сервіси, але й їхню доступність, затримку відповіді тощо.
5. Розробка методу для підвищення відмовостійкості мікросервісного програмного забезпечення шляхом безперервного функціонального тестування та тестування відмовостійкості. Найбільшу увагу слід приділити інтеграційному, наскрізному тестуванню та тестуванню відмовостійкості. Це має дозволити виявляти помилки, які будуть призводити до відмов на ранніх етапах та дасть можливість позбутись їх, що в свою сергу знизить накладні витрати.

Отже за результатами виконаної роботи можна сформулювати такі наукову новезну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження полягає у систематизації та критичному аналізі існуючих підходів до забезпечення відмовостійкості мікросервісів, визначено потребу розробки комплексних підходів, які, на відміну від існуючих, адаптують наявні методи до розподіленої природи мікросервісів, автоматизують їх та зменшують накладні витрати (за риахунок)

виявленні їхніх переваг та недоліків, а також у формулюванні конкретних напрямків для подальших досліджень. Новизна полягає також у виявленні прогалин у існуючих дослідженнях, таких як відсутність комплексних підходів, недостатня автоматизація та неврахування специфіки мікросервісної архітектури.

Практична значущість отриманих результатів дослідження полягає у можливості використання систематизованих знань та запропонованих напрямків досліджень для розробки нових та вдосконалення існуючих методів забезпечення відмовостійкості мікросервісів. Це дозволить створювати більш надійні та стійкі до збоїв системи у різних галузях, таких як фінанси, електронна комерція, медицина тощо. Запропоновані підходи можуть бути використані для зниження ризиків виникнення збоїв, мінімізації їх впливу на роботу системи та підвищення загальної ефективності мікросервісних застосунків.

Висновки/Conclusions

Проведений аналіз літератури виявив основні методи забезпечення відмовостійкості мікросервісів, їх класифікацію та особливості застосування. Було встановлено, що існуючі підходи мають свої переваги та недоліки, а їх ефективність залежить від конкретних умов та вимог до системи.

Основними проблемами у забезпеченні відмовостійкості мікросервісів є відсутність комплексних підходів, складність використання існуючих методів, недостатня автоматизація, неврахування специфіки мікросервісної архітектури та дороговизна.

Результати дослідження підкреслюють необхідність подальших досліджень, спрямованих на розробку комплексних, автоматизованих та адаптивних методів забезпечення відмовостійкості, які враховують специфіку мікросервісної архітектури та дозволяють знизити накладні витрати.

У подальших наукових роботах плануємо розробити метод забезпчення відмовостійкості, який повинен виправиди виявлені проблеми.

Розробка протоколів застосування нявних методів

References

1. Baboi M., Iftene A., Gîfu D. Dynamic Microservices to Create Scalable and Fault Tolerance Architecture. Procedia Computer Science. 2019. Vol. 159. P. 1035–1044. URL: https://doi.org/10.1016/j.procs.2019.09.271
2. Livora, T. (2016). Fault Tolerance in Microservices, Master`s thesis, Masaryk University https://is.muni.cz/th/ubkja/masters-thesis.pdf
3. Kumari P., Kaur P. A survey of fault tolerance in cloud computing. Journal of King Saud University - Computer and Information Sciences. 2018. URL: https://doi.org/10.1016/j.jksuci.2018.09.021
4. Ataallah S. M. A., Nassar S. M., Hemayed E. E. Fault tolerance in cloud computing - survey. 2019 11th International Computer Engineering Conference (ICENCO), Cairo, Egypt, 29–30 December 2019. 2019. URL: https://www.jetir.org/papers/JETIR1905444.pdf
5. Bouizem Y. Fault tolerance in FaaS environments : Doctoral disertation. Renn, 2022. 135 p. URL: https://theses.hal.science/tel-03882666/file/BOUIZEM_Yasmina.pdf
6. Molchanov H., Zhmaiev A. CIRCUIT BREAKER IN SYSTEMS BASED ON MICROSERVICES ARCHITECTURE. Advanced Information Systems. 2018. Vol. 2, no. 4. P. 74–77. URL: https://doi.org/10.20998/2522-9052.2018.4.13
7. Falahah, Surendro K., Sunindyo W. D. Circuit Breaker in Microservices: State of the Art and Future Prospects. IOP Conference Series: Materials Science and Engineering. 2021. Vol. 1077, no. 1. P. 012065. URL: https://doi.org/10.1088/1757-899x/1077/1/012065
8. Fault-Tolerant Load Balancing in Cloud Computing: A Systematic Literature Review / V. Mohammadian et al. IEEE Access. 2022. Vol. 10. P. 12714–12731. URL: https://doi.org/10.1109/access.2021.3139730
9. Sharma, P., & Prasad, R. (2023). Techniques for Implementing Fault Tolerance in Modern Software Systems to Enhance Availability, Durability,

and Reliability. Eigenpub Review of Science and Technology, 7(1), 239–251. Retrieved from https://studies.eigenpub.com/index.php/erst/article/view/33

10. MIRAJ M., NURUL FAJAR A. MODEL-BASED RESILIENCE PATTERN ANALYSIS FOR FAULT TOLERANCE IN REACTIVE MICROSERVICE. Journal of Theoretical and Applied Information Technology. 2022. Vol. 100, no. 9. P. 3075–3093. URL: https://www.jatit.org/volumes/Vol100No9/30Vol100No9.pdf
11. Troubitsyna E. Model-Driven Engineering of Fault Tolerant Microservices. The Fourteenth International Conference on Internet and Web Applications and Services ICIW 2019, Turku URL: https://personales.upv.es/thinkmind/dl/conferences/iciw/iciw_2019/iciw_2019_1_10_20069.pdf
12. Automated functional and robustness testing of microservice architectures / L. Giamattei et al. Journal of Systems and Software. 2023. P. 111857. URL: https://doi.org/10.1016/j.jss.2023.111857
13. Kuzmych O. M., Lakhai V. Y., Seniv M. M. Удосконалений підхід до автоматизованого інтеграційного тестування ПЗ за умов застосування безсерверної архітектури. Scientific Bulletin of UNFU. 2023. Т. 33, № 3. С. 97–101. URL: https://doi.org/10.36930/40330314
14. Akuthota, Arunkumar, "Chaos Engineering for Microservices" (2023). Culminating Projects in Computer Science and Information Technology. 42. https://repository.stcloudstate.edu/csit_etds/42
15. Козлов М.С., Аналіз деяких методів підвищення надійності мікросервісної архітектури веб-додатку / М.С. Козлов, Т.І. Петрушина // Інформатика, інформаційні системи та технології: тези доповідей XIX Всеукр. конференції студентів і молодих науковців. Одеса, 29 квітня 2022 р. - Одеса, 2022. – С. 103-104
16. Козлов М. Підвищення надійності мікросервісної архітектури освітньої платформи : Master`s thesis. Одеса, 2022. 62 с

17. Недоступ Д., Солом'яний М. АНАЛІЗ ПІДХОДІВ ЗАБЕЗПЕЧЕННЯ ВІДМОВОСТІЙКОСТІ АРХІТЕКТУР EXTENDED CLOUD. Восьма Міжнародна науково-технічна конференція «Проблеми електромагнітної сумісності перспективних безпроводових мереж зв'язку (EMC-2022)» : матеріали конф., м. Харків, 24 листоп. 2022 р
18. Khurana D., Prakash Shukla A. A Survey on Fault Injection and Bebugging. International Journal of Innovations in Engineering and Technology (IJIET). 2016. Vol. 7, no. 3. P. 283–288. URL: https://ijiet.com/wp-content/uploads/2017/01/40.pdf.
19. FaultSee: Reproducible Fault Injection in Distributed Systems / M. Amaral et al. 2020 16th European Dependable Computing Conference (EDCC), Munich, Germany, 7–10 September 2020. 2020. URL: https://doi.org/10.1109/edcc51268.2020.00014
20. Garg R., Kumar P. A Review of Checkpointing Fault Tolerance Techniques in Distributed Mobile Systems. (IJCSE) International Journal on Computer Science and Engineering. 2010. Vol. 2, no. 4. P. 1052–1063. URL: https://www.researchgate.net/publication/49619228_A_Review_of_Checkpointing_Based_Fault_Tolerance_Techniques_in_Mobile_Distributed_Systems.
21. ISO/IEC 25010. Software and data quality. Effective from 2011-05-06. Official edition. URL: https://www.iso.org/obp/ui/ru/#iso:std:iso-iec:25010:ed-1:v1:en.
22. Dubois S. Tolerating Transient, Permanent, and Intermittent Failures : Master`s thesis. Paris, 2011. 295 p.
23. B. Ugrynovsky, “Methods and means of increasing the reliability of software, taking into account the process of its software aging,” Ph.D. dissertation, Lviv Polytechnic National Univ., Lviv, Ukraine, 2022