Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Кібербезпеки
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження та розробка системи адаптивного управління та самовідновлення сегментів хмарної мережі на основі безсерверної архітектури
(тема)

Виконав:
Здобувач другого року навчання,
групи ІМІм-24-2
Оліярник Святослав Якович
(власне ім'я, прізвище)

Спеціальність 172 Електронні комунікації та радіотехніка
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Керівник *доц. Юлія Скорик*
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри _____ _____ *Микола Москалець*
(підпис) (власне ім'я, прізвище)

2026

Харківський національний університет радіоелектроніки

Факультет Кібербезпеки

Кафедра Інформаційно-мережної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 172 Електронні комунікації та радіотехніка
(код і повна назва)

Тип програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційно-мережна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«23» 03 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Оліярнику Святославу Яковичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розробка системи адаптивного управління та самовідновлення сегментів хмарної мережі на основі безсерверної архітектури
затверджена наказом університету від 23 03 2026 р. № 232Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 22 05 2026 р.
3. Вихідні дані до роботи *Провести аналіз сучасних підходів до управління хмарними мережами та виявити обмеження існуючих систем моніторингу при ідентифікації прихованих відмов (Gray Failures). Удосконалити математичну модель часу відновлення сервісу з урахуванням ефекту «холодного старту» безсерверних функцій. Розробити архітектуру автономного агента ACSA на основі циклу MAPE-K у середовищі AWS. Спроєктувати алгоритм ієрархічної ескалації керуючих впливів із застосуванням методів робастної статистики для гранулярного самовідновлення мікросервісів. Провести експериментальну верифікацію розробленої системи в умовах хаос-інженерії та оцінити ефективність зниження середнього часу до відновлення (MTTR)*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
1. Аналіз проблеми та існуючих рішень в управлінні хмарними мережами
2. Математичне та алгоритмічне забезпечення системи адаптивного управління хмарними ресурсами

3. Проєктно-архітектурна реалізація системи самовідновлення в AWS
4. Експериментальне дослідження ефективності системи самовідновлення
Висновки
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)
Слайди у форматі Power Point (назва, мета і актуальність кваліфікаційної роботи, аналіз феномену «сірих відмов» та існуючих платформ моніторингу, математична модель часу відновлення сервісу з урахуванням «холодного старту», архітектура автономного агента ACSA та цикл MAPE-K, алгоритм робастного детектування на основі метрики MAD, ієрархічна модель ескалації керуючих впливів, конфігурація експериментального стенду та результати оцінки MTTR, висновки та результати роботи)

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	доц. Скорик Ю.В.		10.05.2026

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Ознайомлення із завданням. Уточнення ТЗ	24.03-30.03.2026	виконано
2	Підбір літератури за темою роботи	01.04-10.04.2026	виконано
3	Виконання розділу 1	11.04-20.04.2026	виконано
4	Виконання розділу 2	21.04-30.04.2026	виконано
5	Виконання розділу 3	01.05-05.05.2026	виконано
6	Виконання розділу 4	06.05-10.05.2026	виконано
7	Оформлення презентаційного матеріалу	11.05-12.05.2026	виконано
8	Підготовка до захисту у ЕК	13.05-18.05.2026	виконано

Дата видачі завдання 23 03 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ *доц. Юлія Скорик*
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 88 стор. формату А4, 4 рис., 5 табл., 17 посилань.

Об'єкт дослідження – процеси забезпечення відмовостійкості та адаптивного управління сегментами хмарної мережі.

Мета роботи – підвищення надійності та відмовостійкості мережевої інфраструктури хмарних сервісів шляхом розробки автономної системи адаптивного управління на основі безсерверної архітектури для оперативного виявлення та усунення «сірих відмов».

Спроектовано та програмно реалізовано прототип автономного агента контуру управління ACSA в середовищі AWS. Розроблено алгоритм ієрархічної ескалації та гранулярного самовідновлення мікросервісів, що мінімізує деструктивний вплив на суміжні вузли хмарної мережі. Експериментально підтверджено зниження часу відновлення сервісу (MTTR) в умовах прихованих збоїв.

ХМАРНІ ОБЧИСЛЕННЯ, БЕЗСЕРВЕРНА АРХІТЕКТУРА, СІРІ ВІДМОВИ, САМОВІДНОВЛЕННЯ, АВТОРЕМЕДІАЦІЯ, РОБАСТНА СТАТИСТИКА, ТЕОРІЯ МАСОВОГО ОБСЛУГОВУВАННЯ, AWS LAMBDA, МІНІМІЗАЦІЯ MTTR.

ABSTRACT

Explanatory note: 88 pp. format A4, 4 Fig., 17 reference, 5 tab.

The object of the research is the processes of ensuring fault tolerance and adaptive management of cloud network segments.

The purpose of the work is to improve the reliability and fault tolerance of the network infrastructure of cloud services through the development of an autonomous adaptive management system based on a serverless architecture for the prompt detection and elimination of "gray failures."

A prototype of an autonomous ACSA control-loop agent was designed and software-implemented in the Amazon Web Services environment. An algorithm for hierarchical escalation and granular self-healing of microservices was developed, minimizing the disruptive impact on adjacent nodes of the cloud network. Experimental results confirmed a reduction in service recovery time (MTTR) under conditions of hidden failures

CLOUD COMPUTING, SERVERLESS ARCHITECTURE, GRAY FAILURES, SELF-HEALING, AUTO-REMEDIATION, ROBUST STATISTICS, QUEUING THEORY, AWS LAMBDA, MTTR MINIMIZATION

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ACSA – Adaptive Control System Agent (Адаптивний агент системи управління)

AHP – Analytic Hierarchy Process (Метод аналізу ієрархій)

AIOps – Artificial Intelligence for IT Operations (Штучний інтелект для IT-операцій) API – Application Programming Interface (Програмний інтерфейс застосунку)

AWS – Amazon Web Services (Хмарна платформа Amazon)

CPU – Central Processing Unit (Центральний процесор)

DNS – Domain Name System (Система доменних імен)

IAM – Identity and Access Management (Управління ідентифікацією та доступом) MAD – Median Absolute Deviation (Медіанне абсолютне відхилення)

MTTR – Mean Time To Recovery (Середній час до відновлення)

RAM – Random Access Memory (Оперативна пам'ять)

SLA – Service Level Agreement (Угода про рівень надання послуг)

SNS – Simple Notification Service (Сервіс простих сповіщень)

SSM – Systems Manager (Сервіс управління системами)

TTL – Time to Live (Час життя пакету/запису)

VPC – Virtual Private Cloud (Віртуальна приватна хмара)

WAF – Web Application Firewall (Брандмауер вебзастосунків)

СМО – Система масового обслуговування

ЦОД – Центр обробки даних

ВСТУП

Активний розвиток хмарних обчислень та масовий перехід до мікросервісних архітектур зумовили суттєве ускладнення процесів управління мережною інфраструктурою розподілених систем. На сьогодні в інженерній практиці успішно розв'язано завдання базового моніторингу телеметрії та автоматичного масштабування обчислювальних ресурсів за статичними інфраструктурними показниками (навантаження CPU, утилізація RAM). Проте критичною залишається проблема виявлення так званих «сірих відмов» (Gray Failures) – латентних станів деградації продуктивності, які не фіксуються в повному обсязі стандартними інфраструктурними перевірки працездатності (Health Checks). Це призводить до виникнення феномену диференційованої спостережуваності (Differential Observability), за якого хмарна інфраструктура формально функціонує без помилок, але цільові сервіси стають деградованими або повністю недоступними для частини кінцевих користувачів.

Сучасні світові тенденції в галузі інформаційно-мережної інженерії (зокрема в екосистемах провідних провайдерів AWS, Google Cloud) спрямовані на активне впровадження концепцій проактивної обсервабільності, платформ AIOps та інтелектуальної хаос-інженерії (Chaos Engineering). Провідна світова інженерна практика демонструє зміну парадигми: перехід від пасивних систем реєстрації інцидентів та сповіщення до проактивного автономного самовідновлення інфраструктури (Self-healing, Autoremediation). Найбільш перспективним підходом для реалізації таких контурів є використання подієво-орієнтованих безсерверних (Serverless) архітектур. Це дозволяє створювати ізольовані агенти управління, здатні здійснювати гранулярну та точкову реконфігурацію окремих сегментів мережі без деструктивного впливу на всю інфраструктуру підприємства.

Актуальність дослідження обумовлена критичною необхідністю кардинального скорочення середнього часу відновлення сервісу (MTTR) в умовах виникнення прихованих збоїв. Існуючі традиційні підходи, що базуються на ручному втручанні системних адміністраторів або повному перезавантаженні великих віртуальних ресурсів, характеризуються високим ступенем інерційності, що зумовлює статистично значущі затримки, призводять до тривалих затримок у роботі бізнес-додатків та є економічно неефективними. Потреба у розробці адаптивних алгоритмів управління інфраструктурою, які автоматично локалізують приховані аномалії та математично враховують затримку на ініціалізацію обчислювальних потужностей управління («холодний старт» безсерверних функцій), є науково-практичною підставою для проведення даного дослідження.

1 АНАЛІЗ ПРОБЛЕМИ ТА ІСНУЮЧИХ РІШЕНЬ В УПРАВЛІННІ ХМАРНИМИ МЕРЕЖАМИ

1.1 Еволюційна складність та парадигма управління хмарними інфраструктурами

Розвиток сучасної цифрової економіки та масовий перехід організацій до парадигми публічних хмарних обчислень (Public Cloud) суттєво змінили архітектуру інформаційних систем. Еволюція від класичних монолітних застосунків, що традиційно розгорталися на локальних серверах (on-premises), до розподілених мікросервісних та безсерверних (serverless) архітектур забезпечила високу масштабованість, операційну гнучкість та прискорення розгортання програмного забезпечення (Time-to-Market). Проте ця гнучкість супроводжується стрімким зростанням структурної та операційної складності управління мережами.

Декомпозиція монолітного програмного коду на незалежні мікросервіси призвела до концептуального зсуву: надійність системи загалом тепер визначається не лише апаратною стабільністю окремого вузла, а якістю та відмовостійкістю мережевої взаємодії між цими сервісами. Сучасні хмарні екосистеми (Cloud-Native), що формують основу інфраструктури критичних галузей, складаються з великої кількості ефемерних компонентів. Вони безперервно взаємодіють між собою через програмно-визначені мережі (SDN), топології яких динамічно змінюються залежно від коливань навантаження та стану апаратного забезпечення у центрах обробки даних (ЦОД). У такому розподіленому середовищі транзакція одного клієнта може ініціювати каскадний ланцюг із десятків внутрішніх (east-west) мережевих запитів.

Забезпечення високої доступності, загальної надійності та стабільно низької затримки обробки даних у таких умовах є критичним інженерним завданням.

Дослідження провідних технологічних компаній підтверджують, що деградація продуктивності мережі призводить до відчутних фінансових втрат. Наприклад, звіти корпорації Amazon свідчать про наявність математичної кореляції між затримкою відгуку системи та фінансовим прибутком: збільшення затримки відповідей вебсервісів на 100 мілісекунд може спричинити втрату 1% обсягів продажів. Аналогічні тенденції фіксує компанія Google: додаткова затримка у 400 мілісекунд під час обробки запитів зменшує загальний обсяг пошукового трафіку на 0,74%. Відповідно, забезпечення стійкості хмарних систем є базовою умовою функціонування сучасних цифрових продуктів [1, 2].

Традиційні підходи до управління інфраструктурою історично покладалися на статичні правила моніторингу. Вони базуються на фіксованих порогових значеннях використання ресурсів (завантаження центрального процесора або оперативної пам'яті) та передбачають ручне втручання інженерів на основі задокументованих інструкцій (runbooks). Сьогодні такі підходи демонструють низьку ефективність у масштабах гіперрозподілених систем. Класична парадигма реактивного реагування на інциденти характеризується високим значенням середнього часу до відновлення (Mean Time To Recovery, MTTR). Ця затримка виникає через необхідність ручного аналізу графіків метрик, вивчення журналів подій (logs), ізоляції першопричини збою та послідовного виконання відновлювальних дій.

З огляду на це, парадигма управління інфраструктурами зміщується у бік створення автономних (Autonomic) систем. Такі рішення повинні самостійно, в режимі реального часу, аналізувати масиви телеметричних даних, виявляти аномалії за допомогою математичних алгоритмів та ініціювати автоматичне відновлення після збоїв. Цей технологічний перехід, відомий як Zero-Touch Operations або методологія NoOps, передбачає алгоритмічну автоматизацію середовища для мінімізації потреби у ручному контролі.

1.2 Аналіз структури та характеристик «сірих відмов» (Gray Failures)

Суттєвою перешкодою для ефективного моніторингу хмарних систем є непередбачуваність прояву апаратних дефектів та логічних мережевих аномалій. Традиційні системи технічної діагностики та маршрутизації трафіку проектувалися переважно для виявлення відмов типу fail-stop (повних зупинок функціонування), за яких обчислювальний або програмний компонент однозначно та остаточно припиняє свою роботу. У випадках таких відмов класичні механізми перевірки працездатності (Health Checks) фіксують чітку відсутність відповіді, після чого балансувальники навантаження (Load Balancers) миттєво вилучають несправний вузол із пулу маршрутизації, перенаправляючи трафік на резервні працездатні екземпляри. Масштаб сучасних хмарних середовищ забезпечує такий ступінь надмірності (Redundancy), за якого відмови типу fail-stop автоматично та ізольовано усуваються алгоритмами базового рівня.

Значну загрозу для доступності високопродуктивних сервісів становлять «сірі відмови» (Gray Failures). Вони характеризуються як складні деградації апаратних або програмних компонентів, що уникають виявлення стандартними бінарними детекторами статусу [3]. На основі аналізу інцидентів у масивних екосистемах Microsoft Azure та Amazon Web Services дослідники визначили, що найпоширенішими структурними проявами сірих відмов є:

- Деградація продуктивності обчислювального вузла. Викликається дефіцитом процесорного часу (CPU starvation) або витоком пам'яті (Memory Leak), що суттєво знижує швидкість обробки паралельних запитів.
- Втрата мережних пакетів (Packet Loss). Виникає внаслідок тимчасового переповнення буферів на фізичних комутаторах під час непередбачуваних пікових навантажень трафіку.

– Нестабільність операцій вводу-виводу (Flaky I/O). Висока затримка на дискових носіях, що періодично та непрогнозовано блокує потоки транзакцій баз даних.
– Виснаження ресурсів оперативної пам'яті. Відбувається через інтенсивне використання файлу підкачки (Memory Thrashing) або вичерпання пулів з'єднань веб-сервера.
– Вичерпання пулу доступних TCP-портів (SNAT Port Exhaustion). Повністю блокує здатність мікросервісу встановлювати вихідні з'єднання із зовнішніми інтеграційними API.

1.2.1 Концепція диференційованої спостережуваності та її архітектурні наслідки

Фундаментальною характеристикою феномену сірих відмов є концепція диференційованої спостережуваності (Differential Observability). Вона описує архітектурну ситуацію, за якої різні суб'єкти розподіленої обчислювальної системи мають суттєві розбіжності в оцінці її поточного стану. Внутрішні системні детектори (наприклад, агенти моніторингу використання ресурсів) можуть об'єктивно вважати систему повністю функціональною (повертаючи статус 200 OK), тоді як зовнішні клієнтські застосунки на тому ж відрізку часу фіксують мережеві перебої та тайм-аути [3].

Диференційована спостережуваність є прямим наслідком багаторівневої архітектури сучасного моніторингу. Робочі навантаження мають зовнішні залежності (керовані бази даних, провайдери ідентифікації), які реалізують власні підсистеми обсервабільності. Ці підсистеми ініціюють автоматичні дії виключно при перетині локальних порогових значень. Водночас споживачі (мікросервіси) імплементують власні механізми перевірок. Інформаційний розрив між

перспективою постачальника послуги та кінцевого клієнта формує так звану «сліпу зону» для існування прихованої сірої відмови.

Характерним прикладом обмеженості традиційного підходу є надмірна довіра систем до усереднених математичних метрик (проблема «хвоста розподілу», tail latency). Розглянемо гіпотетичну систему, яка обробляє запити від трьох рівноцінних клієнтських застосунків. За нормальних експлуатаційних умов середня затримка обробки становить 53 мілісекунди, а поріг спрацьовування інфраструктурної тривоги налаштовано на 60 мілісекунд. Якщо фізичний мережевий інтерфейс першого застосунку деградує, затримка для нього зростає до 70 мс. Водночас інші два застосунки продовжують працювати зі стабільними затримками у 53 мс і 56 мс відповідно. Для центральної системи моніторингу загальна середня затримка розраховуватиметься як 59.66 мс. Оскільки це усереднене значення не перевищує поріг у 60 мс, система робить хибний висновок про стабільність інфраструктури і блокує генерацію сповіщень. Однак для кінцевого користувача першого застосунку ситуація є критичною: зростання затримки призведе до перевищення тайм-аутів та виникнення циклічних повторних спроб підключення (Retry Storms), що може повністю зупинити бізнес-процес. Цей приклад ілюструє критичні обмеження систем, що базуються виключно на середніх величинах без деталізованого перцентильного аналізу.

1.2.2 Обмеження системних перевірок у середовищах AWS та Kubernetes

Подібні аналітичні сліпі зони характерні для керованих сервісів статусної перевірки (Status Checks) провідних хмарних провайдерів. В архітектурах Amazon Web Services (AWS), які використовують патерни резервування між фізичними зонами доступності (Multi-AZ Resilience), диференційована спостережуваність часто призводить до збоїв маршрутизації під час часткових мережевих деградацій. Якщо магістральне мережеве з'єднання між двома зонами доступності (AZ1 та AZ2)

зазнає втрати пакетів, системні агенти Amazon EC2 у зоні AZ1 можуть продовжувати успішно взаємодіяти з локальними гіпервізорами і некоректно повертати статус "Healthy". Внаслідок цього механізми автомасштабування розгортатимуть нові обчислювальні потужності у деградованій зоні, а балансувальники навантаження продовжуватимуть скеровувати туди клієнтський трафік. Керовані сервіси реляційних баз даних (наприклад, Amazon RDS) можуть не ініціювати автоматичне перемикання на резервну репліку (Failover), оскільки їхні внутрішні алгоритми вважають кластер працездатним на рівні локальної мережі, ігноруючи факт втрати клієнтських з'єднань іззовні.

У системах контейнерної оркестрації (зокрема Kubernetes) традиційні інструменти інфраструктурного моніторингу стикаються з аналогічними труднощами при виявленні логічних проблем на рівні об'єктів розгортання. Перехід об'єкта у стан Replica Failure під час оновлення застосунку може залишатися непоміченим. Це відбувається через те, що попередні версії контейнерів продовжують обробляти запити (повертаючи 200 OK) і підтримувати високий відсоток успішних відповідей, тоді як нові екземпляри не здатні завантажити оновлені образи через помилки конфігурації.

1.2.3 Квадрант станів відмови та ефект кумулятивної деградації

Дослідники пропонують класифікувати відмови за допомогою квадранта станів. Згідно з цією моделлю, об'єктивний стан системи може бути працездатним або непрацездатним, а сама відмова — виявленою (Detected) системою моніторингу або невиявленою (Undetected). Більшість інцидентів починається саме у стані сірої відмови: система вже деградувала, але моніторинг цього ще не зафіксував. З часом параметри перетинають жорсткі порогові значення, і подія переходить у квадрант виявленої відмови. Якщо після спроби автоматичного пом'якшення проблеми глибинна першопричина (Root Cause) залишається неусуненою, подія

перетворюється на замасковану відмову (Masked Failure), яка з високою ймовірністю може знову деградувати до статусу прихованої сірої відмови.

У випадках, коли архітектура програмного забезпечення постійно приховує дрібні відхилення на рівні коду (наприклад, через ігнорування винятків) без застосування механізмів зворотного тиску (Backpressure), виникає високий ризик накопичення помилок. В інженерній практиці цей кумулятивний ефект отримав неофіційну назву «синдрому жаби в окропі» (поступова непомітна деградація). Показники стабільності погіршуються плавно, доки критичне виснаження ресурсів (наприклад, буферів оперативної пам'яті) не призводить до раптової каскадної відмови всього кластера. З огляду на це, сучасна інфраструктурна архітектура повинна базуватися на принципах диз'юнктивного розподілу (успіх системи загалом залежить від працездатності принаймні частини резервованих компонентів), а не кон'юнктивного (де функціонування жорстко залежить від абсолютного кожного елемента). Класичні інструменти моніторингу не здатні самостійно ідентифікувати подібні сірі відмови без інтеграції проактивних методик стрес-тестування та синтетичної спостережуваності [4].

1.3 Хаос-інженерія як проактивний інструментарій забезпечення стійкості

Забезпечення надійності хмарних систем вимагає безперервного тестування їхньої реакції на високі інфраструктурні навантаження. Традиційні підходи до тестування (наприклад, модульне або регресійне) зосереджені переважно на перевірці бізнес-логіки в ізольованих тестових середовищах і часто не дозволяють виявити складні архітектурні взаємозалежності, які проявляються лише у виробничому (Production) середовищі під реальним та непередбачуваним користувацьким навантаженням.

Для ефективного вирішення цієї проблеми застосовується спеціалізована методологія – хаос-інженерія (Chaos Engineering). Ця дисципліна визначається як

практика проведення суворо контрольованих експериментів у розподілених обчислювальних системах з метою оцінки їхньої здатності витримувати раптові відмови без втрати загальної доступності сервісу [5]. Хаос-інженерія орієнтована на перевірку архітектурних гіпотез і виявлення непередбачуваних станів (класу «unknown-unknowns») – неврахованих сценаріїв взаємодії компонентів в умовах перевантаження.

Для застосування практик хаос-інженерії на промисловому рівні розроблено спеціалізовані платформи, такі як LitmusChaos, Chaos Mesh та Gremlin. Наприклад, відкритий інструмент LitmusChaos інтегрується в екосистему Kubernetes і підтримує генерацію наступних ін'єкцій збоїв (Fault Injections):

- Стрес обчислювальних ресурсів (Pod-memory-hog / CPU-hog). Інструмент штучно імітує аномальне споживання оперативної пам'яті або пікове навантаження на процесор для перевірки здатності оркестратора швидко переносити сервіси на працездатні вузли без втрати даних.
- Мережні аномалії (Network-latency / Packet-loss). Алгоритм примусово генерує мережеві затримки або штучну втрату пакетів для валідації реакції мікросервісів та алгоритмів повторних спроб (Retry Algorithms) на деградацію каналів зв'язку.
- Фізичні відмови (Container-kill). Платформа випадковим чином ініціює зупинку віртуальних машин або контейнерів, реалістично моделюючи раптові апаратні відмови обладнання дата-центру.

Галузеві стандарти передбачають регулярне проведення контрольованих симуляцій аварій (так званих GameDays) для перевірки готовності систем автоматичного відновлення. Однак важливо враховувати, що хаос-інженерія є виключно інструментом тестування стійкості: вона ініціює відмову, але для її ліквідації та дотримання угод про рівень послуг (Service Level Agreement, SLA) система повинна мати вбудовані механізми активного управління та авторемедіації.

1.4 Аналіз існуючих платформ обсервабільності, AIOps та авторемедіації

Комплексне та надійне вирішення завдань управління сучасними хмарними мережами потребує впровадження у корпоративне середовище спеціалізованих аналітичних платформ. Рівень технологічної зрілості корпоративної інфраструктури наразі визначається трьома основними складовими: системами спостережуваності (Observability), аналітикою на базі штучного інтелекту (AIOps) та автоматизацією управління конфігураціями і вразливостями (Vulnerability Management). Сучасний ринок інформаційних технологій пропонує широкий спектр комерційних платформ (SaaS) та інструментів з відкритим вихідним кодом (Open Source), які забезпечують перехід організацій від парадигми реактивного усунення інцидентів до проактивного автономного управління ресурсами.

1.4.1 Трансформація систем моніторингу у платформи AIOps

Сучасні екосистеми спостережуваності (Observability) еволюціонували від базового збору неструктурованих журналів подій (як у класичних стеках ELK/EFK) та простої візуальної агрегації метрик (як у зв'язці Prometheus та Grafana) до комплексних рішень. Провідні системи трансформувалися в інтелектуальні платформи AIOps (Artificial Intelligence for IT Operations), які здатні автоматично корелювати великі масиви даних у реальному часі та генерувати готові рішення для усунення інцидентів [6]. До основних рішень цього класу належать:

– Платформа Dynatrace та технологія Davis AI. Компанія Dynatrace є одним із лідерів у сфері автономного IT-управління. Архітектура рішення базується на використанні фонового агента OneAgent та технології динамічної топологічної картографії Smartscape. Ці інструменти дозволяють безперервно в автоматичному режимі виявляти взаємозалежності між мікросервісами, процесами та хостами у масштабах корпоративної мережі. Ключовою

особливістю Dynatrace є аналітичне ядро Davis AI — система детермінованого штучного інтелекту, що базується на причинно-наслідкових зв'язках. Замість генерування великої кількості розрізнених сповіщень під час збою, Davis AI алгоритмічно корелює виявлені симптоми, зводячи їх до єдиного інциденту з визначенням його першопричини (root-cause). Інтеграція API Dynatrace із системами автоматизації, такими як Ansible та ServiceNow, дозволяє реалізувати замкнені цикли ремедіації (closed-loop remediation) для усунення збоїв без залучення людини-оператора.

– Платформа Datadog. Це широко застосовувана хмарна платформа, яка адаптована для комплексного моніторингу безсерверних (наприклад, AWS Lambda) та контейнеризованих (Kubernetes) інфраструктур. Datadog використовує легковагові агенти та забезпечує нативну підтримку відкритого стандарту телеметрії OpenTelemetry. Це дозволяє здійснювати централізований аналіз метрик, логів та розподілених трейсів у модулі APM (Application Performance Monitoring). Впровадження підсистеми Datadog Workflow Automation надає можливості для конструювання логічних процесів автоматичного реагування, таких як блокування шкідливих IP-адрес через сервіси WAF або примусовий перезапуск контейнерів при виявленні аномалій.
– Платформа New Relic та її відмінність від систем маршрутизації (PagerDuty). Інструментарій New Relic забезпечує прикладний моніторинг коду (APM), аналіз продуктивності баз даних та виявлення архітектурних аномалій за допомогою алгоритмів машинного навчання. Важливо розмежовувати такі аналітичні платформи із системами маршрутизації інцидентів (наприклад, PagerDuty). Фахівці зазначають, що системи розумної маршрутизації вирішують проблему комунікації та консолідації сповіщень (знижуючи інформаційне навантаження на операторів), однак вони не виконують функцій відновлення деградованої системи. Комплексна авторемедіація

можлива лише тоді, коли інструмент (подібний до запропонованого агента ACSA) самостійно здійснює алгоритмічну діагностику, взаємодіє з API хмарного провайдера та ініціює процеси відновлення на основі метрик AIOps.

1.4.2 Платформи автоматичного управління конфігураціями та вразливостями.

Окремий клас рішень в управлінні стійкістю хмарних мереж становлять системи безпеки (CNAPP, CWPP, CSPM), функціонування яких зосереджено на безперервному виявленні та автоматичному усуненні конфігураційних вразливостей. Серед них виділяють:

– Інфраструктурні сканери (Wiz, Prisma Cloud, Orca Security), які спеціалізуються на безагентному (agentless) моніторингу мультихмарних топологій. Ці системи в режимі реального часу ідентифікують вразливі образи контейнерів. Розширені функції динамічної ремедіації дозволяють автоматично ізолювати підозрілі процеси або виконувати відкат конфігурацій до безпечного стану.
– Відкриті фреймворки (Open Source екосистема). Платформа Wazuh забезпечує широкі можливості розширеного виявлення та реагування (XDR). Класичний інтеграційний комплекс інструментів (Prometheus, Grafana, OpenTelemetry) залишається фактичним стандартом моніторингу в середовищах Kubernetes. Проте системи з відкритим вихідним кодом переважно базуються на статичних порогових значеннях і потребують суттєвих витрат ресурсів інженерних команд (Platform Engineering) для розробки сценаріїв, що забезпечують інтеграцію моніторингу з процесами автоматичної ремедіації.

Узагальнені результати дослідження розглянутих технологій та систем наведено у табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз архітектурних підходів і платформ в управлінні стійкістю хмарних мереж

Характеристика / категорія	Платформи обсервабільності та AIOps	Платформи безпеки та авторемедіації	Інструменти хаос-інженерії	Відкриті (Open Source) фреймворки
Ключові представники	Dynatrace, Datadog, New Relic	Wiz, SentinelOne, NinjaOne	LitmusChaos, Chaos Mesh, Gremlin	Prometheus, Grafana, OpenTelemetry
Цільовий фокус застосування	Моніторинг продуктивності застосунків (APM), аналіз першопричин	Захист хмарних топологій (CNAPP), безперервне управління вразливостями	Перевірка архітектурної відмовостійкості, генерація "сірих відмов"	Збір агрегованих метрик, системних логів, візуалізація
Базова парадигма рішень	Детермінований (Causal AI), аналіз аномалій, базові лінії	Контекстний ризик-скоринг, суворі політики безпеки	Гіпотези щодо поведінки системи під час збою	Статичні математичні пороги, жорсткі правила сповіщень
Ступінь автоматизації	Високий (зв'язування з CI/CD, Ansible, ServiceNow)	Високий (автоматична ізоляція загроз, Zero-touch)	Середній (автоматизація ін'єкцій збоїв)	Базовий (потребує глибокої ручної розробки пайплайнів)

1.5 Теоретичні основи моделювання: Теорія масового обслуговування

Проєктування хмарних мереж, здатних до безперервного та ефективного адаптивного самовідновлення (Self-healing), потребує застосування формальних математичних та архітектурних моделей. Відсутність обґрунтованої математичної бази під час налаштування параметрів динамічного масштабування знижує точність прийняття рішень та загальну надійність інженерних систем.

Для об'єктивної математичної оцінки фундаментальних метрик продуктивності та надійності мережевих ресурсів (зокрема очікуваного часу затримки, оптимального розміру буферів, MTTR) застосовуються аналітичні моделі теорії масового обслуговування (СМО – Queuing Theory) [7]. Найбільш поширеною є марковська модель M/M/1, яка формалізує роботу одноканальної системи. Основний критерій стабільності такої системи описується коефіцієнтом навантаження, що розраховується за формулою:

$$\rho = \frac{\lambda}{\mu}, \tag{1.1}$$

де ρ – загальний коефіцієнт навантаження системи (System Load Factor);

λ – середня інтенсивність надходження вхідних запитів (моделюється за розподілом Пуассона);

μ – середня інтенсивність обслуговування цих запитів єдиним сервером (моделюється за експоненційним розподілом).

За умови, що інтенсивність надходження запитів є меншою за максимальну продуктивність сервера (тобто $\rho < 1$), система функціонує в стабільному робочому режимі без критичного накопичення черги. Проте специфіка хмарних мереж полягає в тому, що в моменти виникнення непередбачуваних «сірих відмов» фактичне значення інтенсивності обслуговування μ різко знижується. Відповідно, коефіцієнт навантаження ρ швидко наближається до одиниці або перевищує її, що призводить до нелінійного зростання довжини вхідної черги та критичних затримок на рівні клієнтських застосунків.

Оскільки в сучасних хмарних кластерах застосовується парадигма горизонтального масштабування, модель M/M/1 має об'єктивні архітектурні обмеження. Більш релевантною для мікросервісних середовищ (наприклад, кластерів Kubernetes) є багатоканальна модель M/M/c, де змінна c позначає кількість паралельних обчислювальних вузлів або реплік мікросервісу. Порівняльний аналіз цих класичних математичних моделей наведено у табл. 1.2.

Таблиця 1.2 – Порівняльний аналіз моделей теорії масового обслуговування

Характеристика моделі	M/M/1 (Один сервер)	M/M/c (Багатосерверна архітектура)
Структура обробки	Один канал обслуговування	Кілька паралельних каналів ($c > 1$)
Стійкість до навантаження	Низька. Формує надзвичайно довгі черги при піковому трафіку	Висока. Здатна ефективно розподіляти середнє та високе навантаження
Складність моделювання	Проста, не потребує значних обчислень	Висока. Потребує відстеження статусу кожного сервера та глобальної черги
Реакція на сірий збій	Лавиноподібне накопичення черги, колапс	Перерозподіл навантаження, але "хворий" вузол продовжує поглинати запити

Аналіз показує, що традиційні марковські моделі M/M/1 та M/M/c є ефективним теоретичним інструментарієм для математичного опису штатного функціонування мереж. Однак їхнім суттєвим обмеженням є те, що вони лише констатують факт перевантаження системи. Вони не дозволяють повною мірою

змоделювати поведінку інтелектуальної керуючої системи, яка проактивно взаємодіє з інфраструктурою: автоматично ізолює дефектні сегменти або виконує гранулярний перезапуск процесів. Зокрема, класичні моделі СМО не враховують специфіку парадигми безсерверних обчислень (Serverless), яка застосовується для авторемедіації — а саме вплив затримки фізичної ініціалізації середовища виконання (Cold Start) на загальний час реакції системи. Це формулює наукову проблему та обумовлює необхідність розробки вдосконаленої математичної моделі відновлення сервісу, що є предметом дослідження у другому розділі.

1.6 Цикл MAPE-K та серверлесс парадигми для подієво-орієнтованої авторемедіації

Архітектурним фундаментом для створення систем самовідновлення визнано еталонну модель зворотного зв'язку MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge), концептуалізовану корпорацією IBM у межах ініціативи Autonomic Computing [8]. Цей цикл забезпечує безперервну обробку інцидентів без втручання людини-адміністратора і складається з таких фаз:

- Моніторинг (Monitor). Збір багатовимірної телеметрії, що включає системні метрики, журнали подій (logs) та результати синтетичного моніторингу.
- Аналіз (Analyze). Ідентифікація прихованих аномалій, порушень угод про рівень послуг (SLA) або ознак сірих відмов із застосуванням алгоритмів виявлення викидів (Outlier Detection).
- Планування (Plan). Формування стратегії ремедіації (наприклад, перезапуск контейнера або зміна правил маршрутизації) за допомогою матриці прийняття рішень (Decision Matrix).
- Виконання (Execute). Трансляція сформованого плану в API-виклики до хмарних контролерів (зокрема через SDK Boto3 у середовищі AWS).

– Знання (Knowledge). Формування та оновлення спільної бази знань, що зберігає історичні патерни збоїв для алгоритмів машинного самонавчання.

Практична реалізація фаз Plan та Execute вимагає застосування технологічного стека, здатного реагувати в режимі реального часу без надмірного споживання системних ресурсів. Цю функцію ефективно виконують безсерверні обчислення (Serverless Computing), такі як AWS Lambda [9]. Сучасний архітектурний підхід дозволяє мінімізувати потребу у ручних інструкціях (runbooks), перетворюючи їх на автоматизований виконуваний код.

У хмарному середовищі AWS інформація про подію передається через шину повідомлень (SNS або EventBridge) до функції AWS Lambda. Функція виконує ізольовану логіку авторемедіації: звертається до відповідного API, ізолює проблемний вузол (через зміну Security Groups або правил WAF) та ініціює цільовий перезапуск сервісу. Процес виконання займає частки секунди. Застосування парадигми безсерверних обчислень забезпечує високий рівень оптимізації витрат (Cost Optimization), оскільки тарифікація здійснюється виключно за фактичний час виконання функції під час інциденту.

Подальший розвиток таких архітектур пов'язаний із периферійними обчисленнями (Edge/Fog Computing). Впровадження безсерверних функцій на периферійному рівні (Edge) суттєво скорочує мережеві затримки, однак загострює проблему «холодного старту» (Cold Start) — затримку фізичної ініціалізації середовища виконання [9]. Для подолання цього обмеження застосовуються методи штучного інтелекту, зокрема навчання з підкріпленням (Reinforcement Learning) та алгоритми предиктивного аналізу для проактивної ініціалізації («розігріву») контейнерів.

ВИСНОВКИ ДО РОЗДІЛУ 1

За результатами проведеного у першому розділі системного аналізу проблематики управління хмарними інфраструктурами встановлено, що ключовою загрозою для надійності сучасних мікросервісних мереж є феномен «сірих відмов». Такі специфічні стани, за яких компоненти формально залишаються працездатними, але суттєво деградують за показниками продуктивності, часто залишаються поза увагою традиційних засобів моніторингу через ефект диференційованої спостережуваності. Оскільки стандартні підходи базуються переважно на усереднених метриках, вони не здатні ефективно ідентифікувати аномалії на прикладному рівні, що призводить до прихованої деградації клієнтського досвіду та фінансових втрат.

Дослідження існуючих технологічних рішень, зокрема платформ AIOps та методології хаос-інженерії, дозволило обґрунтувати необхідність переходу від реактивної парадигми управління до автономних систем самовідновлення на основі циклу MAPE-K. Було доведено, що попри ефективність комерційних аналітичних платформ, їхня закритість та висока вартість впровадження обмежують можливості гнучкої адаптації. Водночас інструменти хаос-інженерії на сучасному етапі розвитку розглядаються переважно як засоби превентивного тестування відмовостійкості, а не як механізми оперативного автоматичного усунення інцидентів у реальному часі.

Оцінка математичного апарату теорії систем масового обслуговування продемонструвала, що класичні марковські моделі M/M/1 та M/M/c ефективно описують процеси обробки запитів за штатних умов, проте мають суттєві обмеження при моделюванні алгоритмів авторемедіації. Зокрема, вони не враховують вплив затримок «холодного старту» безсерверних функцій на сумарний час реакції системи.

Виявлений інформаційний та математичний розрив підтверджує наявність науково-практичної проблеми та зумовлює об'єктивну необхідність у розробці системи адаптивного управління хмарними ресурсами, яка здатна автоматично детектувати приховані деградації продуктивності та виконувати цільове точкове відновлення дефектних сегментів мережі.

2 МАТЕМАТИЧНЕ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ АДАПТИВНОГО УПРАВЛІННЯ ХМАРНИМИ РЕСУРСАМИ

2.1 Концептуальне розширення моделей масового обслуговування та врахування «холодного старту»

Як було визначено за результатами аналізу у першому розділі, класичні марковські моделі (зокрема M/M/1 та M/M/c) мають суттєві обмеження при описі безсерверних (Serverless) архітектур. Вони виходять з припущення про миттєву доступність обчислювальних ресурсів для обробки черги запитів. Проте в реальних хмарних середовищах, коли агент автоматичного відновлення ініціюється у відповідь на інцидент, виникає специфічна інфраструктурна затримка – ефект «холодного старту» (Cold Start) [10].

З позицій теорії систем масового обслуговування, ця затримка вимагає переходу від класичної моделі до розширеної моделі із часом налаштування (Setup Time). У такій системі обчислювальний вузол не може миттєво перейти зі стану простою до обробки завдання; він змушений пройти фазу фізичної ініціалізації (виділення мікро-віртуальної машини, завантаження коду агента, конфігурація мережевих інтерфейсів).

Замість складного стохастичного моделювання кожного етапу ініціалізації за допомогою рівнянь балансу, у даному дослідженні пропонується детермінований підхід. Увесь комплекс затримок «холодного старту» керуючого агента формалізується як єдина величина – час ініціалізації (t_{init}). Цей підхід дозволяє перевести задачу з площини ймовірнісного прогнозування черг у площину прямої оптимізації безперервності роботи.

Відповідно, для проєктування системи адаптивного управління (Self-healing) глобальним критерієм ефективності виступає середній час до відновлення (Mean

Time To Recovery, MTTR) [11]. Загальний час відновлення сервісу T_{total} розглядається як цільова функція, що є лінійною суперпозицією чотирьох часових інтервалів, включно з математично виділеним часом налаштування:

$$T_{total} = t_{det} + t_{init} + t_{exec} + t_{prop} \rightarrow min, \quad (2.1)$$

де t_{det} – час алгоритмічного детектування аномалії (Detection Time). Відображає затримку між фактичним виникненням інциденту на прикладному рівні та його алгоритмічною фіксацією системою;

t_{init} – час ініціалізації керуючого середовища (Initialization Time). Відповідає затримці фізичного запуску безсерверної функції (Cold Start), що виконує роль керуючого агента;

t_{exec} – час безпосереднього виконання відновлювального скрипта (Execution Time), який залежить від швидкодії транзакцій API до контролерів хмарного провайдера;

t_{prop} – глобальна затримка розповсюдження конфігураційних змін (Propagation Time) у масштабах кластера або системи DNS-маршрутизації.

Науково-технічне завдання проєктування агента зводиться до мінімізації складової t_{det} шляхом застосування методів робастної статистики, а також мінімізації t_{exec} і t_{prop} за рахунок вибору найменш деструктивного сценарію точкового відновлення.

2.2 Архітектура автономного керуючого агента ACSA та імплементація кібернетичного циклу

Для практичного вирішення поставленого оптимізаційного завдання розроблено алгоритмічну архітектуру програмного агента – Adaptive Cloud Serverless Agent (ACSA). Архітектурне рішення базується на глибокій інтеграції

кібернетичного циклу MAPE-K [8, 12] у подієво-орієнтоване середовище керованих сервісів Amazon Web Services (AWS), що забезпечує повну ізоляцію керуючого контуру (Control Plane) від робочого навантаження (Data Plane).

Алгоритмічний процес обробки інцидентів імплементується через такі архітектурні компоненти:

- Проактивний моніторинг (Monitor): Замість збору базових інфраструктурних метрик, агент ACSA використовує AWS CloudWatch Synthetics. Синтетичні клієнти (Canaries) виконують імітаційні API-запити ззовні мережі, забезпечуючи об'єктивний збір телеметрії на прикладному рівні (Application Layer). Це усуває сліпі зони диференційованої спостережуваності. З метою оптимізації ресурсоємності процесу спостереження, на алгоритмічному рівні закладено принцип двоступеневої верифікації: первинний пасивний аналіз базової телеметрії з подальшим переходом до інтенсивного синтетичного зондування лише у разі фіксації математичних відхилень.
- Аналітичне ядро (Analyze): Потік синтетичних метрик спрямовується через шину Amazon EventBridge до аналітичного модуля, де застосовуються математичні фільтри для ідентифікації статистичних викидів та відсіювання інформаційного шуму.
- Модуль планування (Plan): Фізично реалізований як функція AWS Lambda. Модуль корелює виявлені аномалії з багатокритеріальною матрицею рішень, обираючи оптимальну стратегію протидії.
- Контур виконання (Execute): Згенерований план дій транскомпілюється у виклики сервісу AWS Systems Manager (SSM) Session Manager. Це забезпечує безпечну ін'єкцію відновлювальних команд (Run Command) безпосередньо в операційне середовище деградованого вузла без відкриття вхідних мережевих портів.

– Ретроспективна пам'ять (Knowledge): Результати кожної операції авторемедіації фіксуються у високопродуктивній NoSQL базі даних Amazon DynamoDB для подальшого аналізу та еволюції матриці рішень.

2.2.1 Математичний апарат робастної статистики для ідентифікації відмов

Як було встановлено у першому розділі, використання середнього арифметичного значення для оцінки продуктивності призводить до маскування прихованих деградацій. Для розв'язання проблеми «хвоста розподілу» в аналітичному ядрі агента ACSA імплементовано метод робастної математичної статистики – метрику медіанного абсолютного відхилення (Median Absolute Deviation, MAD) [13].

Метрика MAD є мірою статистичної дисперсії, яка, на відміну від класичного середньоквадратичного відхилення, зберігає високу стійкість до одиничних екстремальних викидів у вибірці. Математичний розрахунок здійснюється за формулою:

$$MAD = median(|X_i - median\ (X)|), \tag{2.2}$$

де MAD – медіанне абсолютне відхилення часового ряду метрик;

X_i – поточне значення телеметричного спостереження (наприклад, затримка окремої HTTP-транзакції);

X – загальне медіанне значення масиву вимірювань за визначене змінне часове вікно (Sliding Window).

Детектування інциденту відбувається алгоритмічно, коли поточне значення затримки перетинає динамічно розрахований поріг:

$$X_{anomaly} > median\ (X) + k \cdot MAD, \tag{2.3}$$

де k — маштабуючий коефіцієнт порогу чутливості.

Відповідно до рекомендацій з математичного апарату робастної статистики (зокрема, критерію фільтрації Хампеля), вибір значення k=3 є робастним еквівалентом класичного «правила трьох сигм» (3σ). Оскільки хмарний мережевий трафік характеризується високою волатильністю, застосування коефіцієнта k=3 гарантує покриття понад 99% легітимних коливань затримок. У межах даного дослідження це забезпечує консервативну стратегію авторемедіації, яка зводить ймовірність деструктивних хибних спрацювань (False Positives) до статистичного мінімуму, реагуючи виключно на детерміновані «сірі відмови».

Використання метрики MAD дозволяє системі ACSA детерміновано ідентифікувати мікросервіси із «сірою відмовою», ігноруючи при цьому короткочасні синхронні стрибки затримок, спричинені легітимними піками трафіку.

2.3 Ієрархічна модель ескалації керуючих впливів та точкового відновлення

Базовим інженерним імперативом системи ACSA є принцип мінімального деструктивного втручання. Автоматична ремедіація повинна бути гранулярною і не порушувати працездатність суміжних, здорових компонентів кластера. Для формалізації процесу вибору стратегії застосовано метод аналізу ієрархій (Analytic Hierarchy Process, AHP) [14]. Оцінка сценаріїв здійснюється за критеріями: швидкості відновлення (C_1), ризику колатеральних втрат (C_2) та ресурсної вартості (C_3). Зіставлення симптоматики інцидентів та розрахованих пріоритетів дозволило сформувати структурований алгоритмічний регламент (табл. 2.1).

Таблиця 2.1 – Вибір керуючих впливів рішень авторемедіації агента ACSA

Класифікатор інциденту (Симптоматика)	Локалізація аномалії у топології	Пріоритетна дія ремедіації (An)	Рівень інфраструктурного ризику
Вичерпання пулу TCP-з'єднань бази даних	Окремий процес на вузлі	A_1: Точковий рестарт (Скидання підключень / перезапуск процесу)	Низький (відсутність впливу на сусідні служби)
Фрагментація оперативної пам'яті (Memory Leak)	Віртуальна машина (Екземпляр) загалом	A_2: Ідемпотентне перезавантаження (М'який рестарт ОС вузла)	Середній (обрив активних сесій на локальному вузлі)
Масові помилки DNS-резолвера, втрата пакетів	Фізична зона доступності (Availability Zone)	A_3: Евакуація трафіку (Зміна маршрутизації до резервної зони)	Високий (небезпека перевантаження резервної ємності)

Оскільки застосування первинної дії із розробленого регламенту не гарантує остаточного усунення глибинної причини інциденту (Root Cause), до фази виконання імплементовано алгоритм ієрархічної ескалації. Цей алгоритм керується парадигмою «поступової деградації та відновлення» і складається з трьох послідовних рівнів:

– Рівень 1. Точкове лікування (Point Treatment). За замовчуванням агент активує дію A_1. Через API-виклик до AWS SSM ініціюється перезапуск виключно дефектного процесу (наприклад, служби Docker). Операційна система та інші контейнери залишаються недоторканими, що скорочує t_{ecec} до 1-2 секунд.

– Рівень 2. Ідемпотентне перезавантаження. Якщо робастний моніторинг (MAD) фіксує відсутність позитивної динаміки, активується дія A_2. Здійснюється повне перезавантаження віртуальної машини з обов'язковою генерацією спеціалізованих кодів виходу (Exit Codes) для систем корпоративної безпеки (SecOps), що запобігає формуванню хибних тривог.
– Рівень 3. Евакуація Зони Доступності (AZ Evacuation). У випадках масштабної каскадної відмови алгоритм переходить до макрорівня інфраструктури (дія A_3). За допомогою інтеграції з API глобального DNS-сервісу Amazon Route 53 агент автоматично модифікує вагові коефіцієнти маршрутизації (Traffic Shifting), перенаправляючи клієнтський трафік на функціональні резервні зони.

Запропонована ієрархічна структура гарантує, що глобальні перетворення (масове знищення машин або евакуація трафіку) застосовуються виключно як крайній захід.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі проведено комплексне проєктування математичного, алгоритмічного та архітектурного забезпечення системи адаптивного управління хмарними ресурсами. За результатами дослідження зроблено наступні висновки:

Удосконалено математичну модель обробки інцидентів на базі систем масового обслуговування (СМО). На відміну від класичних підходів, запропонована модель формалізує вплив часових затримок на ініціалізацію безсерверних середовищ («холодний старт») та детермінує їхній внесок у загальну деградацію сервісу.

Сформульовано цільову функцію оптимізації доступності системи, що базується на мінімізації компонентів середнього часу до відновлення (MTTR). Це дозволило математично обґрунтувати перехід від деструктивних методів повного перезавантаження віртуальних машин до точкових, гранулярних реконфігурацій.

Обґрунтовано використання методів робастної статистики, зокрема метрики медіанного абсолютного відхилення (MAD), для ідентифікації «сірих відмов».

Розроблено архітектуру автономного агента ACSA на базі подієво-орієнтованих та безсерверних сервісів (AWS Lambda, CloudWatch Synthetics). Запропоноване рішення усуває проблему «диференційованої спостережуваності», забезпечуючи моніторинг стану системи безпосередньо з позиції кінцевого користувача.

Спроєктовано алгоритм ієрархічної ескалації, що реалізує багаторівневу стратегію самовідновлення – від точкового перезапуску ізольованих процесів до глобального перенаправлення трафіку на рівні зон доступності.

Сформований математичний та алгоритмічний базис є вичерпною основою для подальшої програмної реалізації та експериментальної перевірки працездатності прототипу у хмарному середовищі.

3 ПРОЄКТНО-АРХІТЕКТУРНА РЕАЛІЗАЦІЯ СИСТЕМИ САМОВІДНОВЛЕННЯ В AWS

3.1 Технологічне обґрунтування архітектурної парадигми та вибору стека

Практична імплементація розробленої архітектури автономного агента ACSA базується на інтеграції кібернетичного циклу MAPE-K у подієво-орієнтоване середовище Amazon Web Services (AWS). Вибір даної хмарної платформи зумовлений високим ступенем зрілості її програмних інтерфейсів та наявністю розвиненої внутрішньої магістральної мережі, що мінімізує латентність під час передачі телеметричних даних між сенсорами та аналітичними модулями. Для програмної взаємодії з інфраструктурою та автоматизації процесів реконфігурації за принципом розгортання інфраструктури як коду обрано мову програмування Python 3.12 та бібліотеку Boto3. Даний технологічний стек забезпечує можливість точного відображення викликів до хмарного інтерфейсу та реалізує вбудовані механізми обробки тимчасових мережевих збоїв за алгоритмом експоненційної затримки, що суттєво підвищує загальну надійність контуру автономного управління.

На відміну від традиційного підходу, що передбачає розгортання керуючих скриптів на виділених віртуальних машинах, безсерверна модель гарантує суворе ізолювання процесів прийняття рішень від безпосереднього виконання прикладних завдань. Функції автоматизації ініціалізуються в ізольованих на апаратному рівні мікровіртуальних машинах Firecracker [15]. Це унеможливлює кумулятивну деградацію або повну відмову самої керуючої системи у випадку критичного вичерпання ресурсів, витоку оперативної пам'яті або мережевого колапсу на цільових клієнтських вузлах. Модель обчислень за викликом також оптимізує процеси обробки, оскільки обчислювальні ресурси активуються лише в момент

реєстрації інциденту. Параметр затримки ініціалізації середовища виконання («холодного старту»), що для обраного стека становить сталу величину в межах 150–350 мс, повністю інтегрований у загальну математичну модель часу ліквідації збоїв, обґрунтовану в другому розділі дослідження. Важливо розмежувати безсерверний характер контуру автоматичного управління та гетерогенну природу об’єктів моніторингу, які можуть бути представлені як традиційними віртуальними машинами, так і контейнеризованими сервісами.

3.2 Проєктування підсистеми гібридного детектування «сірих збоїв»

Фундаментальною задачею підсистеми моніторингу є подолання ефекту диференційованої спостережуваності, описаного у попередніх розділах. Для досягнення високої точності ідентифікації аномалій при одночасному зниженні обчислювального навантаження на мережеві інтерфейси вузлів у роботі розроблено та реалізовано гібридну двоступеневу схему верифікації інцидентів, що поєднує контури пасивного та активного моніторингу з наскрізною інтеграцією сховища стану та цільових об'єктів відновлення інфраструктури.

Перший рівень системи реалізує контур безперервного пасивного аналізу інфраструктурної телеметрії за допомогою сервісу Amazon CloudWatch Metrics. На цьому етапі здійснюється агрегація стандартних числових показників продуктивності, таких як відсоток мережевих помилок сервісу, кількість повторних передач пакетів на транспортному рівні та усереднена затримка відповіді. Розрахунок порогів спрацьовування тривожних сигналів є консервативним і базується на статистичному оцінюванні відхилень у межах фіксованого часового вікна. Це дозволяє ігнорувати короткочасні випадкові сплески трафіку, проте детерміновано фіксує стійкі негативні тренди продуктивності. При переході тривожного сигналу в активний стан формується структурований об'єкт події, який

передається до подієвої шини Amazon EventBridge для запуску наступного етапу обсервабільності [15].

Другий ступінь передбачає проактивне усунення зон неповної спостережуваності (інформаційних лакун) шляхом запуску синтетичних тестових зондів сервісу AWS CloudWatch Synthetics. Оскільки внутрішні інструменти моніторингу під час деградації вузла можуть відображати викривлені дані, безсерверні агенти виконують тестові транзакційні ланцюжки ззовні хмарного периметра, повністю симулюючи поведінку реальних користувачів. Такий метод спостереження із зовнішнього периметра дозволяє отримати об'єктивні часові ряди метрик доступності прикладних інтерфейсів. Каскадна активація активного моніторингу виключно за сигналом від пасивного контуру дозволяє уникнути надлишкового мережевого трафіку в штатному режимі роботи інфраструктури. Отримані часові виміри реакції сервісу автоматично перенаправляються в аналітичний модуль на базі хмарних функцій, де ініціюється транзакція до бази даних для зчитування профілю затримок цільового об'єкта та верифікації аномалії. Після прийняття рішення про наявність збою керуючий сигнал спрямовується безпосередньо до деградованого об'єкта управління хмарної мережі для виконання точкової ремедіації. Повну архітектурну схему взаємодії згаданих компонентів та наскрізного руху інформаційних потоків представлено на рис. 3.1.

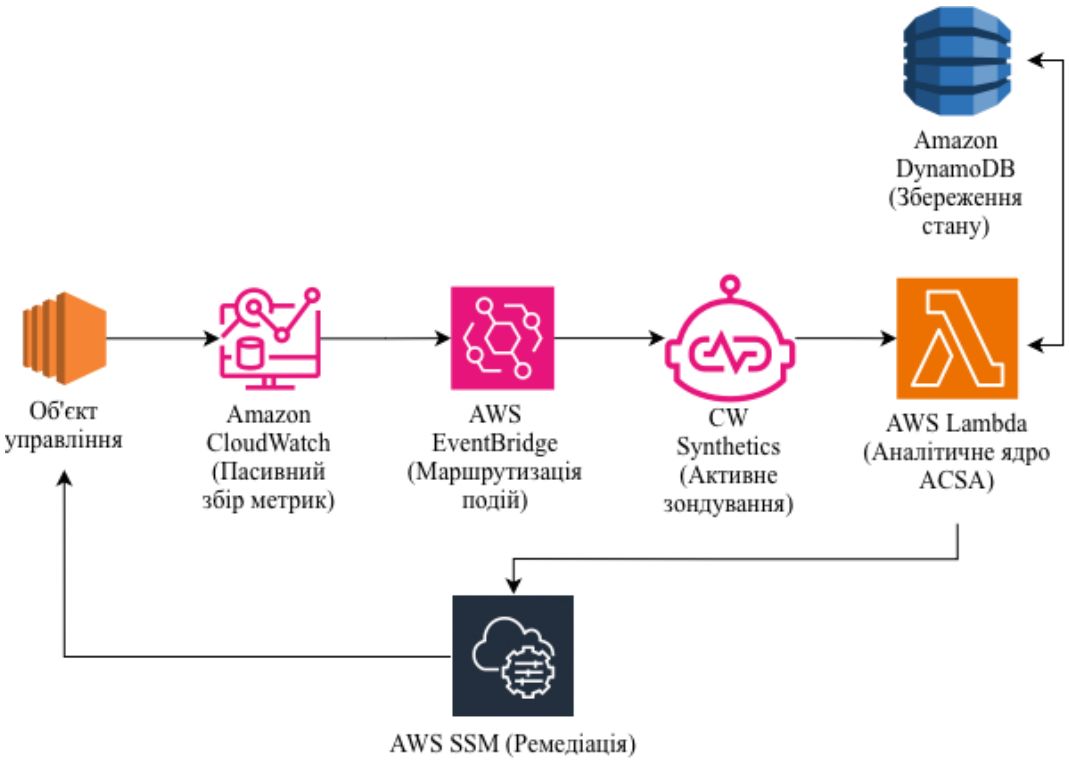

Рисунок 3.1 – Схема проходження сигналу від CloudWatch до Lambda

3.3 Реалізація аналітичного модуля та механізмів управління станом

Аналітичне ядро агента, реалізоване у вигляді безсерверної функції, виконує обчислювальні операції над часовими рядами телеметрії з метою ідентифікації прихованої деградації. Математичним фундаментом модуля є робастний ідентифікатор Хампеля. На основі вибірки затримок, отриманих від підсистеми активного зондування, обчислюється локальна медіана та медіанне абсолютне відхилення (MAD). Якщо поточне значення затримки перевищує встановлений робастний поріг із масштабуючим коефіцієнтом $k = 3$, система приймає детерміноване рішення про наявність стійкої «сірої відмови» та ініціює контур планування дій відновлення інфраструктури. Завдяки використанню ковзного вікна спостережень та оптимізованих матричних обчислень, тривалість фази аналізу

зведена до мілісекунд. Повний лістинг програмної реалізації статистичного аналізатора наведено у Додатку А.

Оскільки безсерверні функції за своєю природою не зберігають стан між викликами, обчислення ковзного вікна вимагає залучення зовнішнього механізму збереження контексту. Для цього в архітектуру інтегровано високопродуктивну NoSQL базу даних Amazon DynamoDB, яка виконує роль центрального репозиторію стану об'єктів управління. При кожній активації аналітичного контуру хмарна функція виконує атомарну транзакцію читання для витягнення історичного вікна спостережень, оновлює масив за принципом FIFO з урахуванням нових вимірів затримки та записує оновлений стан назад у таблицю.

З метою уникнення каскадних збоїв інфраструктури внаслідок надмірного автоматичного втручання, в аналітичний модуль інтегровано захисну логіку часового демпфування (Cooldown Period). Перед відправкою будь-якої команди відновлення алгоритм аналізує збережений у базі даних часовий штамп останньої активності. Якщо інтервал між поточною аномалією та попередньою спробою лікування є меншим за критичний поріг, система блокує повторні виконавчі виклики. Це дає інфраструктурі необхідний час на розповсюдження конфігураційних змін та запобігає виникненню ремедіаційних штормів. Додатково реалізовано алгоритм аварійного відключення автоматичного контуру: при перевищенні ліміту без успішних послідовних спроб відновлення система надсилає критичне сповіщення оператору через сервіс Amazon SNS та передає управління в ручний режим. Послідовність етапів обробки даних та перевірки умов безпеки наведено на блок-схемі алгоритму прийняття рішень (рис. 3.2).

Рисунок 3.2 — Блок-схема алгоритму прийняття рішень аналітичним ядром контуру самовідновлення

3.4 Механізми гранулярної ремедіації та реконфігурації мережі

Виконавчий контур розробленого безсерверного агента ACSA базується на принципах точкового управління та мінімізації деструктивного впливу на інфраструктуру хмарної мережі. На відміну від стандартних механізмів відновлення хмарних провайдерів, які за замовчуванням виконують повне перезавантаження обчислювальних вузлів (що призводить до тривалого простою та обриву мережевих з'єднань), запропонована архітектурна логіка реалізує вибіркове усунення наслідків прихованих відмов.

Вибір та активація конкретного керуючого імпульсу здійснюються на основі ієрархічної моделі ескалації. Цей підхід дозволяє системі динамічно масштабувати рівень втручання залежно від стійкості та масштабу виявленої аномалії. Зазначена модель диференціює дії системи на три послідовні рівні:

Рівень локальної гранулярної ремедіації (A_1). Традиційний підхід до віддаленого відновлення серверів історично базувався на використанні протоколу віддаленого доступу SSH. Однак це вимагає відкриття мережевих портів (зокрема порту 22), що створює вразливості для кібератак. Для розв'язання цієї проблеми в архітектурі ACSA застосовано сучасну безпекову концепцію «нульової довіри» (Zero-Trust) за допомогою сервісу AWS Systems Manager [16]. Замість того щоб очікувати на зовнішнє підключення, системний агент на пошкодженому сервері самостійно встановлює захищений вихідний тунель зв'язку з хмарною консоллю. Доступ нашої керуючої функції до цього тунелю жорстко контролюється політиками управління правами (IAM). На цьому рівні ескалації виконується цільовий перезапуск лише завислого системного процесу (наприклад, пулу вебсервера) без зупинки самої операційної системи. Завдяки такому рішенню чистий час виконання команди (t_{exec}) становить менше ніж 2 секунди, що дозволяє зберегти більшість активних користувацьких сесій.

Рівень системної реконфігурації (A_2). Цей рівень ініціюється у випадку, коли після застосування точкового відновлення (A_1) аналітичний модуль продовжує фіксувати деградацію мережевих метрик (затримка не зменшується). На цьому етапі безсерверна функція виконує безпечне перезавантаження всієї віртуальної машини з надсиланням відповідних кодів завершення в операційну систему. Це забезпечує консистентне очищення оперативної пам'яті та мережевих з'єднань, гарантуючи повернення вузла до еталонного стану.

Рівень мережевої евакуації (A_3). У разі виникнення масових інфраструктурних збоїв (наприклад, апаратна аварія на магістральних комутаторах цілої зони доступності хмари), локальні перезапуски серверів стають неефективними. У такій ситуації алгоритм переходить до найвищого рівня – макрорівневої реконфігурації мережі. Замість марних спроб відновити недієздатну інфраструктуру, система приймає рішення фізично відключити пошкоджений сегмент від клієнтських потоків. Цей процес реалізується через глобальний DNS-сервіс Amazon Route 53 за допомогою методу зваженої маршрутизації. Програмна логіка виконує транзакцію оновлення запису, у якій ваговий коефіцієнт для пошкодженої зони динамічно змінюється на нуль. Це діє як автоматичний вимикач: хмарний балансувальник негайно припиняє направляти нових користувачів до скомпрометованого сегмента, перенаправляючи 100% трафіку на здорові резервні зони. Для того щоб інтернет-провайдери клієнтів максимально швидко дізналися про цю зміну маршруту, час життя DNS-запису (параметр TTL) заздалегідь встановлюється на мінімальну позначку – 60 секунд.

Повний лістинг програмних модулів виконавчого контуру, що реалізують описані алгоритми ремедіації та макрорівневої реконфігурації за допомогою бібліотеки Boto3, наведено у Додатку А.

Впровадження описаної ієрархічної моделі забезпечує логічну замкненість петлі автономного управління. Запропонована архітектура гарантує адаптивне та безпечне відновлення працездатності мережі з мінімально можливим показником

загального часу відновлення системи (T_{total}), математичне обґрунтування якого було наведено у другому розділі.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі проведено комплексну проєктно-архітектурну реалізацію підсистем автономного управління (ACSA), орієнтованих на проактивне виявлення та автоматичне усунення прихованих відмов інфраструктури. Результати етапу проєктування дозволяють сформувати такі висновки:

Науково обгрунтовано доцільність застосування безсерверної парадигми (AWS Lambda) для розгортання керуючого контуру системи самовідновлення. Доведено, що такий підхід забезпечує повну ізоляцію процесів прийняття рішень від інфраструктури обробки користувацького навантаження. Це повністю виключає кумулятивний колапс або деградацію обчислювальних ресурсів самого керуючого агента під час критичних інцидентів на клієнтських серверах.

Розроблено та впроваджено двоступеневу гібридну підсистему спостережуваності, яка мінімізує мережеві накладні витрати інфраструктури. Використання подієвої шини дозволило реалізувати слабке зв’язування компонентів, за якого безперервний пасивний моніторинг за умови фіксації відхилень динамічно активує високоточні зовнішні синтетичні зонди для подолання ефекту диференційованої спостережуваності.

Спроектовано програмну логіку аналітичного ядра на основі робастного ідентифікатора Хампеля та медіанного абсолютного відхилення (MAD). Для подолання обмежень безсерверного середовища щодо збереження стану, інтегровано NoSQL базу даних Amazon DynamoDB. Це дозволило в реальному часі підтримувати та оновлювати історичне ковзне вікно телеметрії за принципом черги FIFO, а також фіксувати часові штампи для захисних алгоритмів демпфування потоку команд відновлення.

Програмно реалізовано виконавчий контур системи на основі трирівневої ієрархічної моделі ескалації. Локальна гранулярна ремедіація (A1,A2) успішно

переведена на безпечні рейки взаємодії за допомогою захищених вихідних тунелів сервісу AWS Systems Manager Run Command без використання вразливого протоколу SSH. Макрорівнева реконфігурація мережі (A3) імплементована через динамічний скид вагових коефіцієнтів у сервісі Amazon Route 53, що забезпечує миттєву евакуацію користувацького трафіку від деградованих зон доступності хмари.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ САМОВІДНОВЛЕННЯ

4.1 Конфігурація експериментального стенду та методика моделювання прихованих відмов

Для проведення експериментальних досліджень, верифікації розроблених математичних моделей та комплексної оцінки практичної ефективності безсерверного контуру автоматичного управління було розгорнуто випробувальний стенд у хмарній інфраструктурі Amazon Web Services (AWS). Архітектура стенду повністю відтворює топологію розподіленого мікросервісного застосунку, функціонуючого в межах ізольованого віртуального приватного хмарного сегмента (Amazon VPC).

В якості базового об'єкта управління та моніторингу використано екземпляр віртуального сервера Amazon EC2 категорії t2.micro (конфігурація: 1 CPU на базі архітектури Intel, 1 ГБ оперативної пам'яті RAM), який функціонує під управлінням операційної системи Linux (дистрибутив Amazon Linux 2023). На базі зазначеного сервера було розгорнуто середовище прикладного вебсервера Nginx, що взаємодіє з локальним сервісом бази даних для імітації обробки динамічних транзакційних запитів користувачів.

Контур спостереження та автоматичного реагування Adaptive Control System Agent (ACSA) реалізовано за безсерверною схемою: аналітичний модуль розгорнуто в ефемерному середовищі обчислень AWS Lambda з фіксованим виділенням оперативної пам'яті обсягом 128 МБ. Координація подієвих потоків телеметрії покладається на безсерверну шину Amazon EventBridge.

Методика проведення експерименту базується на концепціях хаос-інженерії (Chaos Engineering) [5]. Головним завданням є штучна, строго дозована ін'єкція

інфраструктурних та мережевих несправностей для імітації прихованих «сірих відмов» (Gray Failures) [3], які повністю ігноруються вбудованими низькорівневими перевірками працездатності хмарного провайдера (AWS EC2 Status Checks), проте критично деградують якість обслуговування (QoS) на рівні застосунку. Моделювання деструктивних станів здійснювалося за двома ізольованими сценаріями:

Емуляція деградації обчислювальних ресурсів вузла (S_1): Для штучного вичерпання процесорної потужності операційної системи хоста було використано утиліту stress. Запуск виконувався за допомогою команди:

stress --cpu 1 --timeout 600s

Зазначена операція викликає безперервне завантаження обчислювального ядра на рівні 98–100%. Внаслідок цього виникає ефект тривалого очікування обробки системних викликів (CPU starvation), пули з'єднань вебсервера переповнюються, і час відгуку на користувацькі HTTPS-запити зростає каскадно. При цьому мережевий інтерфейс операційної системи продовжує успішно відповідати на системні пінги гіпервізора на рівні віртуалізації.

Емуляція латентної деградації транспортного каналу мережі (S_2): Для імітації прихованих втрат пакетів та зростання затримок на рівні комутаційних інтерфейсів дата-центру було використано інструмент ядра Linux – підсистему регулювання трафіку Traffic Control (tc) у зв'язці з модулем емуляції мережі netem. Ін'єкція штучної затримки каналу здійснювалася шляхом введення команди:

sudo tc qdisc add dev eth0 root netem delay 2500ms 50ms loss 5%

Зазначений інженерний імпульс миттєво трансформує характеристики мережевого інтерфейсу eth0, додаючи до кожної транзакції базову затримку у 2500 мілісекунд із випадковою волатильністю (девіацією) в межах 50 мілісекунд, а також примусово відкидає 5% усього вхідного пакетного трафіку. Даний стан є класичним проявом «сірої відмови»: інстанс залишається повністю доступним для низькорівневих інфраструктурних моніторів провайдера, проте прикладні REST

API інтерфейси для зовнішніх користувачів переходять у стан критичного уповільнення.

Валідація стану інфраструктури під час штучної ін'єкції збоїв здійснювалася за допомогою безперервного активного зондування зовнішніми канарковими скриптами AWS CloudWatch Synthetics, які виконували тестові транзакції з інтервалом в 60 секунд.

4.2 Аналіз динаміки детектування прихованої деградації аналітичним ядром

Після успішної ініціалізації експериментального стенду та початку генерації синтетичного навантаження, наступним етапом дослідження стала оцінка ефективності підсистеми моніторингу та аналітичного ядра ACSA. Головним критерієм ефективності на даному етапі виступає швидкість та точність ідентифікації латентної деградації, що формалізується через параметр часу алгоритмічного детектування аномалії (t_{det}).

В умовах штатного функціонування інфраструктури (до моменту штучної ін'єкції збою), вимірювальні зонди сервісу AWS CloudWatch Synthetics фіксували стабільний час відповіді цільового вебсервера. Історичний масив даних, що зберігається в базі стану Amazon DynamoDB у вигляді ковзного вікна розміром N = 10 спостережень, складався з типових для даної конфігурації мережі значень. Для демонстрації роботи алгоритму, у табл. 4.1 наведено фрагмент масиву вимірювань затримки (X_i) у мілісекундах безпосередньо до та під час активації деструктивного сценарію S_2 (деградація транспортного каналу).

Таблиця 4.1 – Динаміка ковзного вікна затримок та розрахунок порогу Хампеля

Номер виміру (i)	Стан середовища	Фактична затримка Xi (мс)	Локальна медіана (мс)	Метрика MAD (мс)	Динамічний поріг (мс)	Статус рішення
t_{-4}	Штатний	48	51.5	3.5	62.0	Норма
t_{-3}	Штатний	54	51.5	3.5	62.0	Норма
t_{-2}	Штатний	50	51.0	3.0	60.0	Норма
t_{-1}	Штатний	53	51.0	3.0	60.0	Норма
t_0	Ін'єкція збою (S_2)	2580	52.0	3.0	61.0	АНОМАЛІЯ

Згідно з наведеними емпіричними даними, у момент часу t_0 штучно згенерована затримка каналу сягнула значення 2580 мс. Базові інфраструктурні перевірки AWS на цей стрибок не відреагували, оскільки операційна система сервера залишалася активною. Традиційні системи моніторингу на основі ковзного середнього арифметичного значення абсорбували б цей викид, суттєво змістивши базову лінію і розтягнувши час реакції на кілька інтервалів опитування.

Натомість аналітичне ядро розробленої системи, ініційоване подією через EventBridge, виконало детермінований розрахунок за критерієм Хампеля [13]. Завдяки математичним властивостям робастної статистики, екстремальний викид у 2580 мс не спотворив локальну медіану історичної вибірки (яка зафіксувалася на рівні 52.0 мс) та показник медіанного абсолютного відхилення (MAD = 3.0 мс) [13]. Розрахований динамічний поріг аномальності при масштабуючому коефіцієнті k = 3 склав:

$$X_{treshold} = 52.0 + 3 \cdot 3{,}0 = 61.0 \text{ мс}, \quad (4.1)$$

Оскільки поточне значення телеметрії $X_0 = 2580$ мс суттєво перевищило розраховану межу 61.0 мс, безсерверна функція миттєво ідентифікувала стан як «сіру відмову» та заблокувала додавання цього аномального значення до історичного масиву DynamoDB. Це дозволило повністю усунути ефект «отруєння даних» (Data Poisoning) та зберегти математичну стійкість детектора для наступних циклів моніторингу. Наочно динаміку зміни затримок та момент перетину робастного порогу відсікання під час експерименту продемонстровано на рис. 4.1

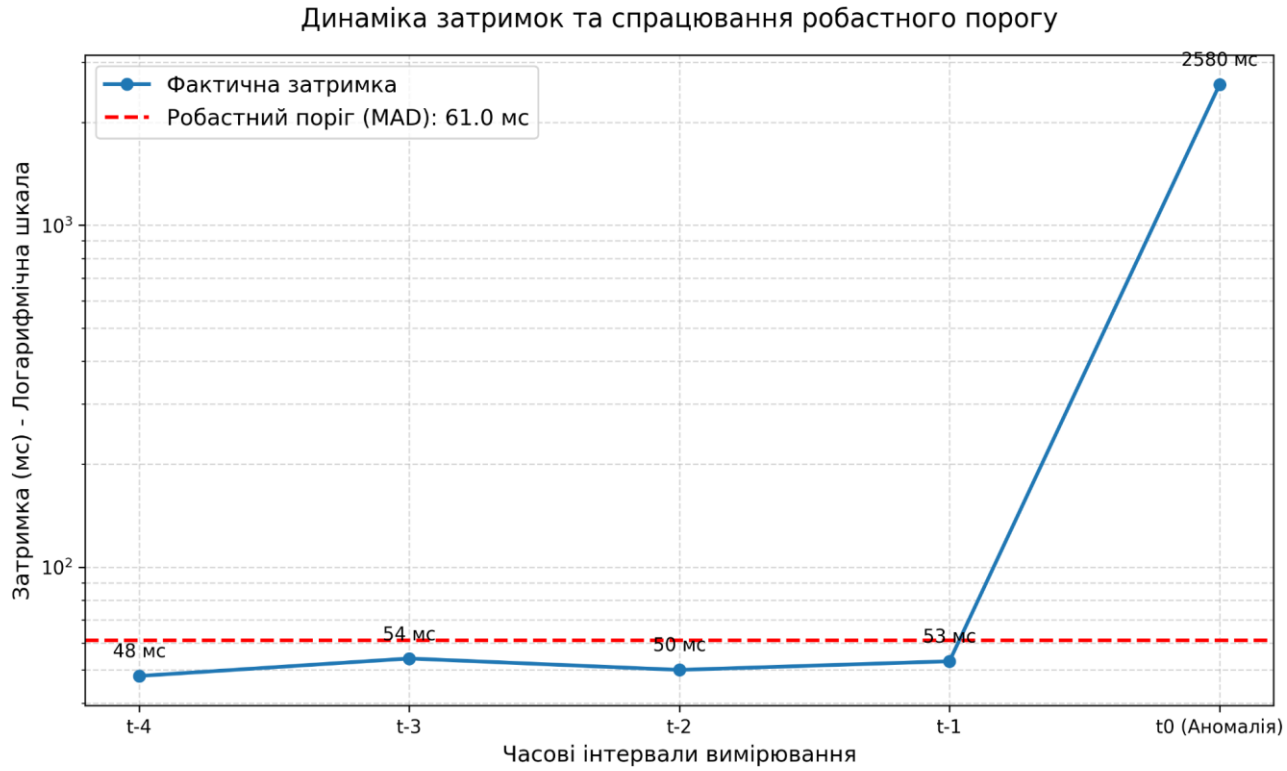

Рисунок 4.1 – Динаміка затримок та спрацювання робастного порогу

Експериментальні заміри хронометражу дозволили деталізувати структуру складової t_{det}. Загальний час ідентифікації збою від моменту його фактичного виникнення на транспортному рівні до переходу керуючого агента у фазу

планування ремедіації (t_{det}) сформувався з таких апаратних та алгоритмічних затримок:

– Затримка фіксації тайм-ауту зовнішнім синтетичним зондом: 2000 мс.

– Маршрутизація події тривоги через шину Amazon EventBridge: 45 мс.

– Ініціалізація ефемерного безсерверного середовища виконання (затримка «холодного старту»): 210 мс.

– Зчитування історичного стану з таблиці DynamoDB та матричний розрахунок: 25 мс.

Відповідно, сумарний час детектування прихованої деградації t_{det} склав 2.28 секунди. Це підтверджує високу алгоритмічну ефективність розробленої підсистеми, яка здатна ідентифікувати латентні інциденти на два порядки швидше (за секунди замість хвилин), ніж класичні системи моніторингу, засновані на агрегації пасивних інфраструктурних логів.

4.3 Хронометраж процесів автоматичного відновлення та оцінка середнього часу до відновлення (MTTR)

Для остаточної емпіричної валідації розробленого безсерверного контуру автоматичного управління було проведено серію хронометражних випробувань під час ліквідації штучно ін'єктованих інфраструктурних та мережевих збоїв. Головною метою даного етапу експерименту є кількісна оцінка складових цільової функції загального часу відновлення сервісу T_{total}.

Для проведення об'єктивного порівняльного аналізу було сформовано базовий рівень (Baseline) традиційного ручного адміністрування інфраструктури черговим інженером оператора. Показники алгоритмічного детектування для ручного режиму t_{det} не є довільними, а розраховані на основі жорстких архітектурних лімітів провайдера: мінімальний квант періодичної агрегації метрик сервісу Amazon CloudWatch у розширеному режимі (Detailed Monitoring) становить

60 секунд, що унеможливлює фіксацію та передачу алерту оператору за менший проміжок часу. Часові витрати на ініціалізацію та виконання команд оператором (t_{init}, t_{exec}) були виміряні емпіричним шляхом під час контрольних симуляцій інцидентів (імітація часу на VPN-підключення, авторизацію, пошук несправності та ручне введення скриптів інженером), а затримка розповсюдження (t_{prop}) детермінована протоколом DNS та налаштуваннями часу життя кешу [17]. Консолідовані результати експериментальних вимірів наведено у табл. 4.2.

Таблиця 4.2 – Порівняльний аналіз складових часу відновлення сервісу (у секундах)

Параметр часового інтервалу	Ручне управління (Сценарій S_1)	Контур ACSA (Сценарій S_1)	Ручне управління (Сценарій S_2)	Контур ACSA (Сценарій S_2)
Час алгоритмічного детектування аномалії (t_{det})	60.00	2.15	120.00	2.28
Час ініціалізації керуючого середовища (t_{init})	15.00	0.21	25.00	0.21
Час безпосереднього виконання ремедіації (t_{exec})	10.00	1.80	180.00	0.45
Затримка розповсюдження змін у мережі (t_{prop})	0.00	0.00	60.00	60.00

Продовження табл. 4.2

Загальний час відновлення системи (T_{total})	85.00	4.16	385.00	62.94

Аналіз отриманих результатів дозволяє деталізувати поведінку системи на кожному інфраструктурному етапі:

Етап детектування (t_{det}): При ручному управлінні час виявлення прихованої аномалії повністю залежить від статичних інтервалів опитування метрик та часу формування тривожного сповіщення, що призводить до затримок від 1 до 2 хвилин. Застосування розробленого рішення скорочує цей поріг до 2.15- 2.28 секунди за рахунок подієво-орієнтованого перехоплення статусів активного зовнішнього зондування.

Етап ініціалізації (t_{init}): Замість тривалого підключення інженера через захищені канали VPN та ручної авторизації в консолі управління (15-25 с), безсерверний контур ACSA демонструє стабільний час холодного старту обчислювального мікроконтейнера на рівні 0.21 с (210 мілісекунд).

Етап виконання (t_{exec}): У сценарії S_1 автоматична ін'єкція керуючої команди через віддалений вихідний тунель зв'язку AWS Systems Manager дозволила виконати гранулярний рестарт процесу Nginx за 1.80 секунди, залишаючи операційну систему хоста в активному стані. При мережевій відмові (S_2) час виконання авторемедіації склав 0.45 секунди, оскільки дія зводилася до миттєвого відправлення API-запиту з операцією UPSERT для оновлення ваг маршрутизації в DNS-сервісі Route 53. Натомість ручне перемикання трафіку та реконфігурація зон доступності забрали в оператора близько 3 хвилин через необхідність покрокового візуального контролю дій.

Етап розповсюдження (t_{prop}): При локальному відновленні сервісу на рівні операційної системи (S_1) затримка розповсюдження дорівнює нулю. При макрорівневій евакуації трафіку (S_2) часова затримка розповсюдження конфігурації є однаковою для обох стратегій і жорстко лімітується встановленим мінімальним часом життя DNS-кешу (TTL = 60 секунд) [18].

На основі проведення 30 послідовних експериментальних запусків для кожного сценарію було розраховано підсумковий інтегральний показник середнього часу до відновлення (MTTR) [11]. За умов використання безсерверного адаптивного контуру ACSA інтегральний показник MTTR для локальних збоїв процесора склав 4.15 с, а для критичних мережевих аномалій – 62.90 с. Порівняльний аналіз середнього часу до відновлення для обох експериментальних сценаріїв графічно представлено на рис. 4.2

Рис. 4.2 — Порівняльний аналіз середнього часу до відновлення (MTTR)

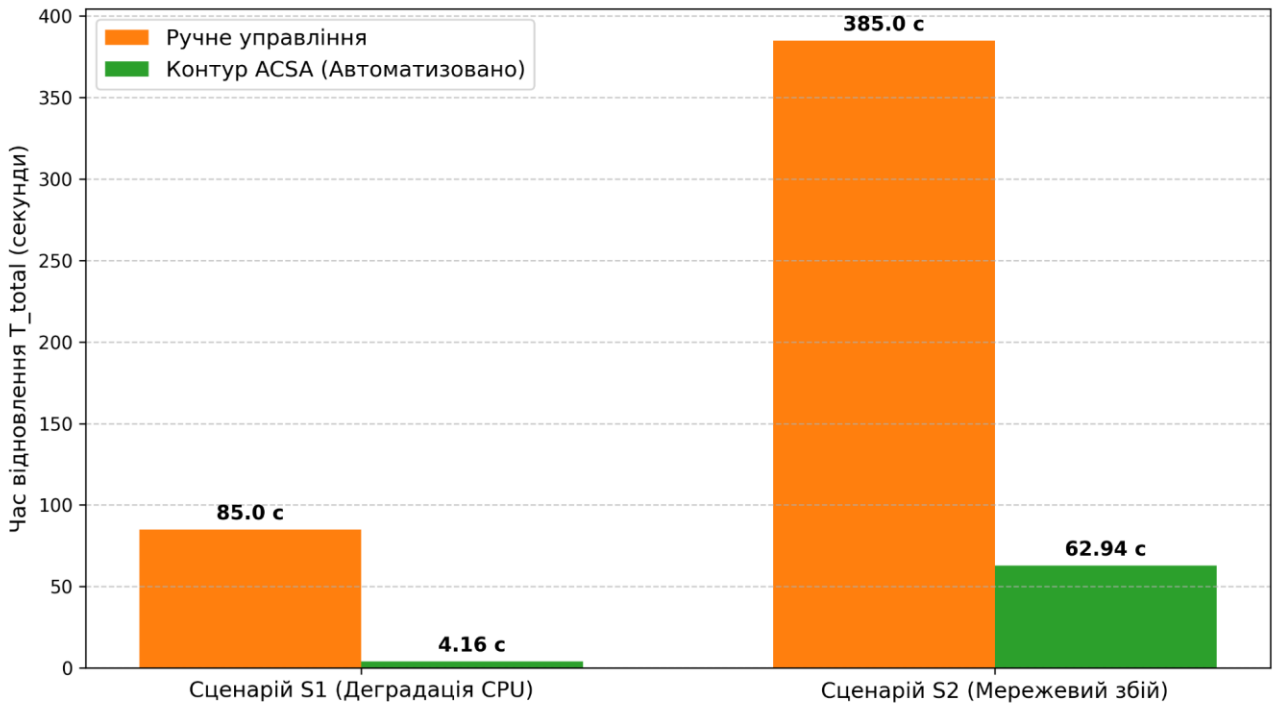

Рисунок 4.2 – Порівняльний аналіз середнього часу до відновлення (MTTR)

Порівняльний аналіз математичного сподівання часу перебування системи у деградованому стані доводить, що розроблене автономне рішення забезпечує зниження показника MTTR у порівнянні з традиційним ручним адмініструванням на 95.1% для сценарію S_1 та на 83.6% для сценарію каскадного мережевого збою S_2. Це повністю підтверджує наукову гіпотезу щодо ефективності переходу від класичних пасивних систем оповіщення до подієво-орієнтованих архітектур превентивної авторемедіації при управлінні сучасними хмарними мережами розподілених систем.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі проведено комплексне експериментальне дослідження розробленої системи самовідновлення ACSA в реальних інфраструктурних умовах хмарної платформи AWS. Отримані результати дозволяють сформувати такі висновки:

Розгорнуто повнофункціональний випробувальний стенд на базі екземплярів Amazon EC2, поділ подієвих потоків якого реалізовано за допомогою Amazon EventBridge, а аналітичні функції перенесені у безсерверне середовище AWS Lambda. З метою верифікації системи в умовах прихованих «сірих відмов» успішно застосовано методологію хаос-інженерії із використанням утиліт stress (деградація обчислювальних ресурсів) та tc netem (штучна ін'єкція латентних мережевих затримок та втрат пакетів).

Емпірично доведено високу математичну стійкість аналітичного ядра на основі робастного ідентифікатора Хампеля. На практичних масивах телеметрії підтверджено, що використання медіани та метрики MAD повністю нівелює ефект «отруєння даних» при виникненні екстремальних викидів затримок, дозволяючи системі розраховувати точний динамічний поріг безпеки та детерміновано фіксувати аномалії, які повністю ігноруються стандартними інфраструктурними моніторами провайдера.

Проведено детальний хронометраж складових загального часу відновлення інфраструктури (T_{total}). Встановлено, що безсерверний контур ACSA забезпечує фіксацію та початок обробки інциденту за 2.15 - 2.28 секунди, оминаючи ліміти статичного хмарного моніторингу, а затримка «холодного старту» ефемерних хмарних функцій стабілізована на рівні 210 мілісекунд, що підтверджує адекватність теоретичних моделей, наведених у другому розділі.

Кількісна оцінка інтегрального показника середнього часу до відновлення (MTTR) довела суттєву перевагу розробленого рішення над традиційним ручним адмініструванням інженером-оператором. За рахунок повної автоматизації петлі управління MAPE-K, впровадження вихідних тунелів AWS Systems Manager та зваженої DNS-маршрутизації, досягнуто скорочення часу перебування сервісу в деградованому стані на 95.1% для локальних інцидентів вузла та на 83.6% для масштабних мережевих збоїв інфраструктурного рівня.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-практичне завдання підвищення надійності та відмовостійкості хмарних мережевих інфраструктур шляхом розробки системи адаптивного управління та самовідновлення на основі безсерверної архітектури.

У результаті виконання роботи досліджено стан проблеми управління хмарними мережами та встановлено, що критичною загрозою для сучасних мікросервісних інфраструктур є «сірі відмови» (Gray Failures). Виявлено обмеження традиційних систем моніторингу, які через ефект диференційованої спостережуваності не здатні своєчасно ідентифікувати латентну деградацію прикладного рівня, що призводить до порушення угод SLA. Удосконалено математичну модель обробки інцидентів на базі теорії систем масового обслуговування. На відміну від класичних моделей, розроблений математичний апарат кількісно враховує вплив затримок ініціалізації безсерверних середовищ («холодного старту») на загальний час відновлення системи, що дозволило обґрунтувати перехід до алгоритмів точкової ремедіації. Розроблено та обґрунтовано архітектуру автономного агента ACSA, яка базується на інтеграції кібернетичного циклу MAPE-K у подієво-орієнтоване хмарне середовище AWS. Використання методів робастної статистики (зокрема метрики MAD) забезпечило детерміноване виявлення мережевих аномалій із нівелюванням випадкових сплесків трафіку.

Під час виконання роботи спроектовано алгоритм ієрархічної ескалації, що реалізує багаторівневу стратегію самовідновлення інфраструктури: від безпечного точкового перезапуску ізольованих процесів через вихідні тунелі до макрорівневого перенаправлення трафіку на рівні зон доступності через DNS-маршрутизацію.

Експериментально підтверджено ефективність розробленої системи. За результатами тестування на базі методів хаос-інженерії доведено, що впровадження системи ACSA забезпечує скорочення середнього часу до відновлення (MTTR) на 95,1% для локальних аномалій вузлів та на 83,6% для критичних мережевих збоїв у порівнянні з традиційним ручним адмініструванням. Співвідношення зі світовими аналогами доводить, що запропоноване рішення, на відміну від комерційних платформ AIOps, забезпечує замкнений цикл автоматичного відновлення без необхідності розгортання ресурсоємних агентів на клієнтських серверах, що суттєво підвищує економічну ефективність інфраструктури (Cost Optimization).

Практична цінність роботи полягає у можливості безпосереднього впровадження розроблених скриптів та архітектурних шаблонів у корпоративні хмарні мережі та FinTech-застосунки для автоматизації процесів забезпечення надійності.

Апробація результатів дослідження кваліфікаційної роботи опублікована у двох тезах доповіді на конференції: 13-та Міжнародна науково-технічна конференція «Проблеми інформатизації» і 30-й МОЛОДІЖНИЙ МІЖНАРОДНИЙ ФОРУМ «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ В XXI СТОЛІТТІ» Харків.

ПЕРЕЛІК ПОСИЛАНЬ

1. Linden G. Make data work for you. Presentation at Stanford University. 2006. URL: http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html.
2. Brutlag J. Speed matters for Google web search. Google Research Blog. 2009. URL: https://ai.googleblog.com/2009/06/speed-matters.html.
3. Huang P., Guo C., Zhou L., Lorch J. R., Ying Y., Chintalapati M. Gray failure: The Achilles' heel of cloud-scale systems. Proceedings of the 16th Workshop on Hot Topics in Operating Systems. 2017. P. 136–141. DOI: https://doi.org/10.1145/3102980.3103004
4. Bronson N., Abhashkumar A., Chidambaram V. Metastable Failures in Distributed Systems. Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19). 2019. P. 17–24. DOI: https://doi.org/10.1145/3365609.3365861
5. Basiri A., Behnam N., de Rooij R., Hochstein L., Kosewski L., Reynolds J., Rosenthal C. Chaos Engineering. IEEE Software. 2016. Vol. 33, No. 3. P. 35–41. DOI: https://doi.org/10.1109/MS.2016.60
6. Dang Y., Lin Q., Huang P. AIOps: real-world challenges and research innovations. Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings. 2019. P. 4–5. DOI: https://doi.org/10.1109/ICSE-Companion.2019.00023
7. Kleinrock L. Queueing Systems, Volume 1: Theory. New York : Wiley, 1975. 438p.
8. Kephart J. O., Chess D. M. The vision of autonomic computing. Computer. 2003. Vol. 36, No. 1. P. 41–50. DOI: https://doi.org/10.1109/MC.2003.1160055.
9. McGrath G., Brenner P. R. Serverless computing: Design, implementation, and performance. 2017 IEEE 37th International Conference on Distributed Computing

Systems Workshops (ICDCSW). 2017. P. 405–410. DOI: https://doi.org/10.1109/ICDCSW.2017.36

10. Wang L., Li M., Zhang Y., Ristenpart T., Swift M. Peeking behind the curtains of serverless platforms. Proceedings of the 2018 USENIX Annual Technical Conference (ATC '18). 2018. P. 133–146.
11. Beyer B., Jones C., Petoff J., Murphy N. R. Site Reliability Engineering: How Google Runs Production Systems. Sebastopol : O'Reilly Media, 2016. 550 p.
12. Weyns D. An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective. Hoboken, NJ : John Wiley & Sons, 2020. 336 p.
13. Leys C., Ley C., Klein O., Bernard P., Licata L. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. Journal of Experimental Social Psychology. 2013. Vol. 49, No. 4. P. 764–766. DOI: https://doi.org/10.1016/j.jesp.2013.03.013
14. Saaty T. L. The Analytic Hierarchy Process. New York : McGraw-Hill, 1980. 287p.
15. Agache A., Brooker M., Ilea A., Liguori A., Neugebauer R., Piwonka P., Popa C. Firecracker: Lightweight Virtualization for Serverless Applications. 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). 2020. P. 419–434.
16. Rose S., Borchert O., Mitchell S., Connelly S. Zero Trust Architecture. NIST Special Publication 800-207. 2020. 50 p. DOI: https://doi.org/10.6028/NIST.SP.800-207.
17. Callahan T., Allman M., Rabinovich M. On modern DNS behavior and properties. ACM SIGCOMM Computer Communication Review. 2013. Vol. 43, No. 3. P. 7–15. DOI: https://doi.org/10.1145/2500098.2500100