Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи забезпечення самовідновлення програмного хмарних інформаційних систем

(тема)

здобувач 2 року навчання,
групи СПзм-23-1
Олексій ГАЛУШКА
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

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

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: ст. викл. Андрій БУГРІЙ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

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

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Галушці Олексію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи забезпечення самовідновлення програмного забезпечення хмарних інформаційних систем

затверджена наказом по університету від "07" квітня 2025 р. № 53стз

2. Термін подання студентом роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи

1. Технології забезпечення функціональної стійкості інформаційних систем
2. Моделі та методи забезпечення самовідновлення програмних систем
3. Моделі та методи управління розподіленими інформаційними системами
4. Методи та засоби моделювання хмарних систем

4. Перелік питань, що потрібно опрацювати в роботі

1 Аналіз предметної області
2 Методи управління розподіленим обчислювальним процесом в гетерогенних хмарних системах
3 Опис розробленого методу та експериментальні дослідження
4 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 12 слайдів

6. Консультанти розділів роботи (заповнюється 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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	08.04.25-18.04.25	
2	Розробка моделей	19.04.25-01.05.25	
3	Реалізація алгоритмів	02.05.25-10.05.25	
4	Розробка структури програмних засобів	11.05.25-21.05.25	
5	Розробка програмних модулів	22.05.25-02.06.25	
6	Оформлення матеріалів кваліфікаційної роботи	03.06.25-05.06.25	
7	Подання кваліфікаційної роботи керівникові та її попередній захист	06.06.25-10.06.25	
8	Подання кваліфікаційної роботи на рецензування	11.06.25-12.06.25	

Дата видачі завдання 07 квітня 2025 р.

Студент _____
(підпис)

Керівник роботи _____ ст. викл. Андрій БУГРІЙ
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 56 с., 22 рис., 6 табл, 1 дод., 15 джерел.

ХМАРНІ ОБЧИСЛЕННЯ, САМОВІДНОВЛЕННЯ, РЕПЛІКАЦІЯ, КОНТРОЛЬНА ТОЧКА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Робота присвячена дослідженню методів забезпечення відмовостійкості та самовідновлення у хмарних інформаційних системах. У дослідженні проведено систематизацію підходів до fault-tolerance у хмарному середовищі, таких як реактивні, проактивні та стійкі методи, зокрема з використанням машинного навчання та штучного інтелекту.

Особливу увагу приділено адаптивному підходу, що поєднує історичні дані про відмови фізичних машин, класифікацію ризиків за допомогою алгоритму KNN і прийняття рішень з використанням нечіткої логіки. Запропонована система FTMSFIS дозволяє динамічно вибирати між реплікацією та контрольними точками на основі таких параметрів, як ризик відмови, доступність ресурсів та пріоритет задачі.

Ефективність запропонованого підходу підтверджено експериментальним моделюванням, що продемонструвало зростання кількості успішних відновлень у порівнянні з традиційними підходами. Робота містить опис реалізації в MATLAB, оцінку метрик (відновлюваність, споживання ресурсів), а також враховує питання безпеки та автоматизації процесів самовідновлення.

ABSTRACT

Master’s thesis: 56 pages, 22 figures, 6 tables, 1 appendice, 15 sources.

CLOUD COMPUTING, SELF-HEALING, REPLICATION, CHECKPOINT, SOFTWARE.

The work is devoted to the study of methods for ensuring fault tolerance and self-healing in cloud information systems. The study systematizes approaches to fault tolerance in the cloud environment, such as reactive, proactive and resilient methods, in particular using machine learning and artificial intelligence.

Particular attention is paid to the adaptive approach that combines historical data on failures of physical machines, risk classification using the KNN algorithm and decision-making using fuzzy logic. The proposed FTMSFIS system allows you to dynamically choose between replication and checkpoints based on parameters such as failure risk, resource availability and task priority.

The effectiveness of the proposed approach is confirmed by experimental modeling, which demonstrated an increase in the number of successful recoveries compared to traditional approaches. The work contains a description of the implementation in MATLAB, an assessment of metrics (recoverability, resource consumption), and also takes into account security issues and automation of self-healing processes.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

ВМ – віртуальна машина

ПМ – програмна модель

ЦП – Центральний процесор

ШНМ – штучні нейронні мережі

IaaS – Infrastructure as a Service

EDCKP – Edge Switch Failure-Aware

FT – Fault Tolerance

FTMSFIS – Fault Tolerance Method Selection Fuzzy Inference System

ID – Identity

FCFS – First-Come First-Served

FPGA – Field-Programmable Gate Array

GKE – Google Kubernetes Engine

HAL – Hardware Abstraction Layer

HLA – High Level Architecture

HPC – High Performance Computing

HPF – Highest Priority First

PaaS – Platform as a Service

PM –Physical Machine

SaaS – Simulation as a Service

TBAFT – Threshold-Based Adaptive Fault Tolerance

VM – Virtual Machine

VPN – Virtual Private Network

ВСТУП

Відмовостійкість є однією з дуже важливих проблем хмари для забезпечення безпеки. Через різноманітну архітектуру послуг, деталізовані архітектури та численні взаємозв'язки, які виникають у хмарі, реалізація є складною. Існуючі дослідження намагаються об'єднати різні типи відмовостійкості, запропоновані для хмарного середовища, але деякі їх можливості виявляються обмеженими. Це дослідження надає детальний огляд найсучасніших методів забезпечення відмовостійкості для хмарних обчислень і класифікує методики відмовостійкості на три категорії: реактивні методи, проактивні методи та стійкі методи.

Реактивні методи дозволяють системі досягти несправного стану, але натомість намагаються відновити пристрій. Проактивні методи допомагають запобігти переходу пристрою в несправний стан шляхом впровадження дій для мінімізації дефектів до того, як вони вплинуть на пристрій. З обох сторін нещодавно розроблені стійкі методи прагнуть скоротити час, потрібний пристрою для пошуку несправності.

У контексті стійких методів машинне навчання та штучний інтелект зіграли важливу роль у відображенні періоду відновлення для завдання, яке потрібно налаштувати. В роботі пропонується вичерпний і детальний опис різних видів несправностей, факторів і різних методів забезпечення відмовостійкості, які використовуються в хмарі. У світлі основних методів та деяких інших специфічних характеристик, також пропонується дослідження різних механізмів толерантності до відмов і забезпечується порівняльне дослідження структур.

Існуючі підходи до відмовостійкості в хмарі в основному базуються на реплікації та контрольних точках. Кожен із цих підходів має свої переваги та наслідування. У цьому документі представлено адаптивний метод відмовостійкості для визначення того, який із двох підходів підійде для

успішного виконання завдання в даних хмарних умовах. Запропонований метод класифікує ризик відмови хост-машин, доступних для виконання завдання, на основі історії їх відмови. Згодом нечітка логіка використовується для визначення відповідного підходу до відмовостійкості шляхом врахування ризику відмови хоста, визначеного користувачем пріоритету завдання та рівня резервування ресурсів. Встановлення пріоритету завдання надає користувачеві можливість контролювати потрібний рівень відмовостійкості, а доступність ресурсів відображає здатність хмарного постачальника запропонувати відмовостійкість. Експерименти з моделюванням підтвердили, що проактивний вибір методу відмовостійкості збільшує кількість успішно виконаних завдань.

Хмарна система самовідновлення визначає використання попередньої інформації, керованої даними, і автоматизації для виявлення потенційних проблем і їх пом’якшення в умовах гібридної хмари. Ці системи розроблені для розпізнавання та усунення збоїв без участі людини. Хмарні системи, що самовідновлюються, можна розглядати як поєднання машинного навчання, штучного інтелекту та автоматизації, щоб запропонувати прогнозну аналітику, яка допомагає встановлювати працездатність системи хмарної інфраструктури [1]. Вони пропонують постійний моніторинг продуктивності системи, щоб розпізнавати потенційні проблеми до їх серйозної появи. Це також допомагає підвищити стійкість системи, мінімізувати час простою та підвищити загальну ефективність роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Забезпечення надійності у хмарних інформаційних системах

За останні роки використання хмарних обчислень значно зросло. Через хмари стає доступним все більше об’єктів. Хмарні обчислювальні компанії, такі як IBM, Amazon, Yahoo та Google, пропонують клієнтам по всьому світу хмарні послуги. Кінцеві користувачі не повинні встановлювати програмне забезпечення на своїх локальних комп’ютерах у цій новій парадигмі, натомість програми та послуги надаються кінцевим користувачам за запитом [2]. Хмарні архітектури передбачають в основному передачу обчислювальних послуг між кількома користувачами. Програми, апаратне та програмне забезпечення є спільними ресурсами. Як правило, хмарні архітектури складаються з трьох основних рівнів: інфраструктура як послуга (IaaS), програмне забезпечення як послуга (SaaS) і платформа як послуга (PaaS). Несправності можуть виникати на всіх цих рівнях, але алгоритми рівня програмного забезпечення для відновлення програм після помилок визначаються та застосовуються. Таблиця 1.1 описує класичну хмарну архітектуру [3].

На надійність ресурсів можуть безпосередньо впливати недоліки, які виникають у хмарних інфраструктурах [1]. Відмовостійкість є складним напрямком досліджень одночасно з розподіленими грід-обчисленнями. Це пов’язано з тим, що грід-системи з інтенсивним використанням комп’ютерів споживають величезну кількість часу для вирішення окремої проблеми [3]. Відмовостійкість хмарних обчислень є гнучкою, як правило, призводить до непередбачених дій, які призводять до збоїв і відмов. Щоб підвищити надійність хмарних обчислень, дефекти слід аналізувати та належним чином виправляти. У комп’ютерних системах є дві підобласті відмовостійкості: апаратне та програмне забезпечення.

Таблиця 1.1 – Класична хмарна архітектура

<table>
<tr><th>Моделі доступу</th><th>Сервіси Cloud</th><th>Ресурси Cloud</th></tr>
<tr><td rowspan="4">Приватні, загальнодоступні, спільноти та гібридні хмари</td><td>Програмне забезпечення як послуга (SaaS)</td><td>Прикладне програмне забезпечення: API для Інтернет-сервісів, корпоративні програми.</td></tr>
<tr><td>Платформа як послуга (PaaS)</td><td>Платформи : Frameworks for application dev, Capacity packages.</td></tr>
<tr><td rowspan="2">Інфраструктура як послуга (IaaS)</td><td>Цифрова інфраструктура: віртуальні машини, цифрові блоки</td></tr>
<tr><td>Фактична інфраструктура: процесорне обладнання, ємність пам'яті, трафік даних.</td></tr>
</table>

Відмовостійкість також є дуже важливою, пропонуючи хмарні ресурси для виявлення, локалізації та відновлення після несправності та підвищення ефективності при виконанні завдань користувача [4]. Відмовостійкість стала основною проблемою для хмарних систем. Основні переваги коригування стійкості до відмов включають зниження витрат, відновлення відмов і покращені показники ефективності [5].

Реактивні методи базуються на звичайних стратегіях керування несправностями в розподілених системах, таких як реплікація, контрольні точки/перезапуск, виявлення та відновлення. Друга категорія проактивних методів - це клас стратегій, які в основному застосовують метод уникнення невдач. Ці стратегії характеризують використання таких підходів, як моніторинг, прогнозування та випередження. По-третє, останнім часом автори привернули більше уваги до стійких методів. У хмарних системах стійкі методи є напрямком майбутньої відмовостійкості. У стійких методах навчання та адаптація всієї системи базується на штучному інтелекті або машинному навчанні [1].

1.2 Відмовостійкість у хмарних обчисленнях

Динамічний характер хмарної системи підвищує ймовірність відмов. Тому методи відмовостійкості є обов’язковими для уникнення наслідків збоїв у хмарній системі. Методи дозволяють системам або пристроям надавати важливі послуги навіть за наявності несправних компонентів. Вони допомагають у виявленні та управлінні системними збоями, викликаними апаратними чи програмними проблемами.

Реактивні схеми використовується для зменшення ефекту збою в хмарі після виникнення збоїв. Найбільш часто використовуваними реактивними схемами є реплікація, а також використання повернень до контрольних точок [6]. Для порівняння, проактивні методи використовуються, щоб заздалегідь передбачити збій і замінити підозрілий компонент деякими запущеними елементами. Основна мета проактивного підходу полягає в тому, щоб уникнути існування збоїв і уникнути процесу відновлення реактивних підходів FT. Хоча використання цих методів має на меті зменшити кількість майбутніх збоїв, вони менш практичні в хмарних системах [7].

Метод відмовостійкості реплікації підтримує кілька копій віртуальних машин або завдань і розміщує їх у різних місцях у центрі обробки даних. У разі збою завдання або віртуальної машини виконання може продовжуватися, поки доступна копія або репліка. Основні переваги використання реплікації полягають у тому, що вона забезпечує вищу доступність і надійність, зменшує переміщення даних у мережі, а отже, має менші накладні витрати на зв’язок. Однак відновлення репліки віртуальної машини або завдання споживає більше ресурсів з точки зору часу та мережі. Іншими словами, впровадження методу реплікації є дорогим, оскільки вона споживає більше місця для зберігання та спричиняє накладні витрати на оновлення реплік [7].

З іншого боку, метод контрольних точок використовується для збереження безпомилкового стану виконання системи регулярно до

стабільного зберігання. У разі збою хмарна система може продовжити запит із останнього збереженого стану контрольної точки, а не перезапускати спочатку [8]. Процес встановлення контрольних точок продемонстровано на рисунку 1.1. Основна перевага впровадження методу контрольних точок полягає в тому, що він дешевший, ніж реплікація. Однак періодичне збереження контрольних точок і відновлення служби займає багато часу, а також створює додаткові накладні витрати під час нормальної роботи.

Рисунок 1.1 – Процес обчислень з використанням контрольних точок

Таким чином, кожен із двох методів має свої плюси та мінуси, тому важливий розумний вибір відповідного методу.

Основним мотивом застосування методів у хмарній системі є покращення доступності та надійності. Щоб досягти цієї мети, у цій галузі було проведено значні дослідження, результатом яких є реплікація, а також протоколи на основі контрольних точок.

Один з методів на основі реплікації розроблено для підвищення узгодженості серверних хмарних служб [9]. Автори використовували евристичну процедуру для вибору відповідного хост-сервера для збереження репліки віртуальних машин, розробили адаптивну модель для прогнозування та вирішення проблем, які виникають у хмарних обчисленнях у реальному часі. Запропонована адаптивна модель використовує реплікацію, а також метод відмовостійкості повторної подачі. Інший метод реплікації та

повторної подачі, заснований на вимозі надійності: якщо сервер із максимальною надійністю виходить з ладу, то завдання або віртуальна машина, що виконується на цьому сервері, планується на сервері, який має друге місце за надійністю. До цього методу може бути додана двоетапна техніка відновлення. У першому стані працюють алгоритм ранжування послуг за їх значущістю. Стратегії реплікації для служб змінюються відповідно до їх рангів. На другому етапі для пошуку реалізована евристична процедура пошуку альтернативного шляху на основі відповідної системи заміни для вирішення проблем складних служб .

Існує одноранговий метод контрольних точок для хмарних центрів обробки даних [10]. За допомогою цього методу зображення контрольних точок віртуальних машин зберігаються на одноранговому або сусідньому сервері. Метод відмовостійкості EDCKP (граничний комутатор із усвідомленням відмови) для обробки відмов сервера, а також крайового комутатора топології повного дерева в хмарному центрі обробки даних. У цій схемі зображення контрольних точок віртуальних машин розміщуються на сервері, який знаходиться в тому самому модулі. Ця схема підвищує надійність хмарних служб, а також споживає менше мережевих ресурсів. Метод контрольних точок, який використовує гнучкий інтервал контрольних точок, де обсяг інтервалу визначається на основі частоти відмов сервера, на якому виконуються віртуальні машини, щоб завершити виконання програми. Метод планування використовується для моніторингу продуктивності віртуального вузла, а також для ефективної схеми контрольних точок для переміщення завдань і завдань з невеликими накладними витратами.

Крім того, було помічено, що дослідження в цій галузі використовували методи програмного обчислення для розробки ефективних методів відновлення. Ці методи стійкі до невизначеності, неточності, наближення та забезпечують простежуваність, надійність та недорогі оптимальні рішення складних обчислювальних. Найбільш часто використовуваними підходами до м’яких обчислень є еволюційні

обчислення, штучні нейронні мережі (ШНМ) і нечітка. З методів м'якого обчислення нечітка логіка має справу з неточною та розпливчастою інформацією, щоб прийняти найкраще можливе рішення для заданих вхідних даних. Таким чином, нечітка логіка знаходить застосування в усіх проблемах, пов'язаних із невизначеними або неточними даними. Оскільки хмарні системи по суті є складними та динамічними системами, методи м'яких обчислень знайшли застосування в цих системах [11].

Також існує багато підходів на основі нечіткої логіки для вибору та міграції віртуальних машин у хмарі [12]. Протестована та оцінена продуктивність запроектованого алгоритму на інструментарії CloudSim. Результати моделювання показали, що ця схема є більш енергоефективною порівняно з іншими. Був розроблений прототип проактивного автомасштабування для хмарної системи. Ця модель складається з двох компонентів для прогнозування та прийняття рішень щодо масштабування.

Прогноз виконується на основі даних часових рядів, і для цього використовується нечіткий метод, нейронна мережа та генетичний алгоритм. Для рішення про масштабування використовується прогнозоване значення, а також наближення порушення SLA для остаточного розподілу ресурсів. Тестування запропонованого рішення на даних реального робочого навантаження підтвердило здійсненність запропонованого підходу, а також його вищу ефективність порівняно з існуючими методами. Застосовується адаптивний метод для виявлення збоїв у хмарній системі на початковому етапі.

Для моніторингу системи автори використали методику прогнозування. Крім того, для виявлення несправностей вони використали алгоритм, заснований на нечіткій логіці. Щоб полегшити планування завдань без помилок, запропоновано ефективний та адаптивний підхід до планування, стійкий до відмов. Щоб досягти оптимальної надійності та доступності інфраструктури хмарних обчислень запропонувана методологію, яка допускає збої ЦП віртуальної машини. Поєднуючи проактивні та реактивні

методи в [6] представлено нову стратегію адаптивної відмовостійкості на основі порогу (TBAFT) у хмарі.

На рисунку 1.2 показана класифікація методів забезпечення відмовостійкості.

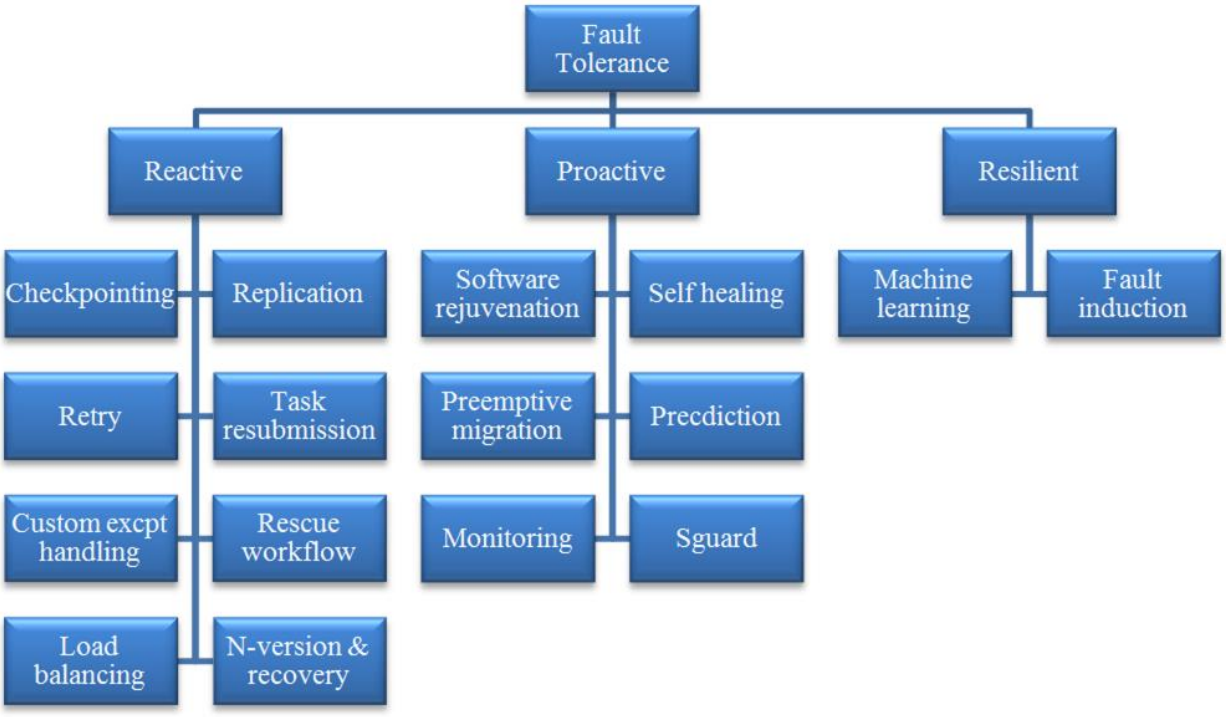

Рисунок 1.2 – Класифікація методів забезпечення відмовостійкості

При розробці методів відмовостійкості треба приділяти увагу певній категорії недоліків і наскільки використовується певний метод відмовостійкості у встановленому випадку. Проаналізуємо оцінки, які взяті з більш ніж 40 систем, які включені до впливових опитувань, про які вже повідомлялося в літературі, разом із структурами, згаданими висче.

На рисунку 1.3 показана статистика застосовності методів відмовостійкості, яка показує поширеність реактивних підходів (з 77% застосовністю) над проактивними підходами. Переконливо, що постійний контроль системи є важливим для проактивних методів. Використовуючи теорію ймовірностей і штучний інтелект, вони дуже покладаються на навчання та передбачення. Завдання, що виконуються в умовах

конструктивної відмовостійкості, залишаються безперервними, доки система не поведе себе відповідно до можливого майбутнього стану системи. З іншого боку, реактивні методи в основному втілені методом реплікації (36% застосовності), тоді як контрольні точки перезапуску (27% застосовності) і міграції (14% застосовності) зазвичай служать допоміжними.

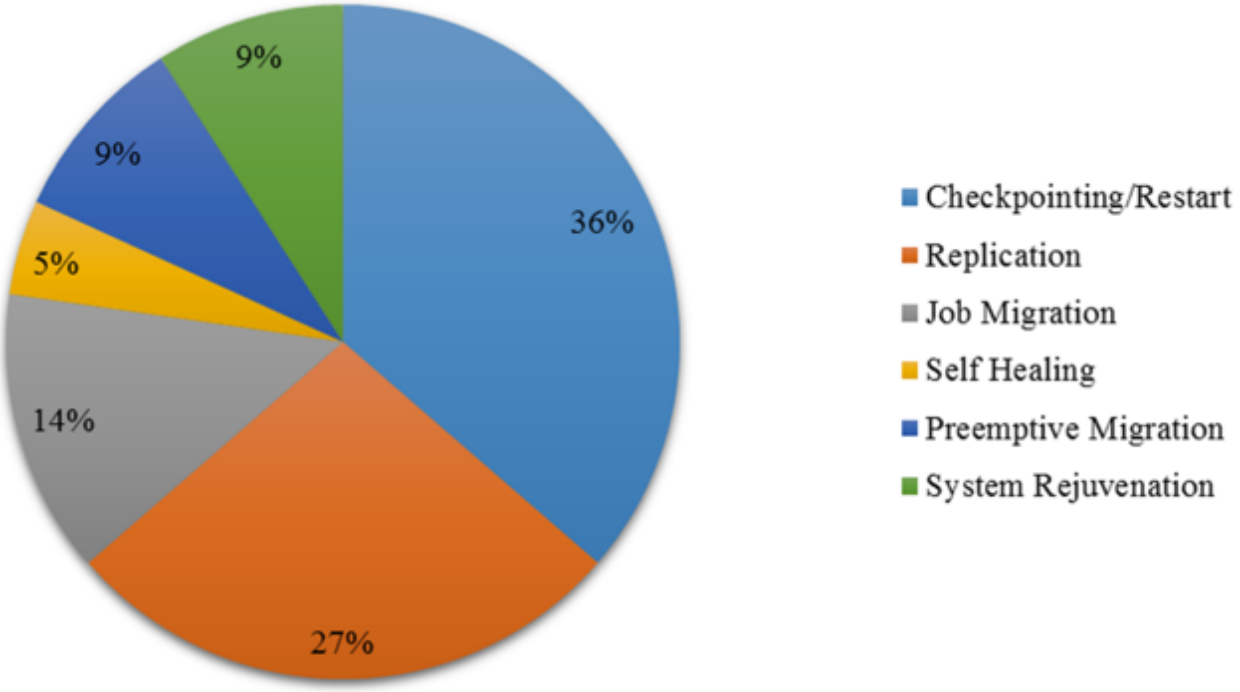

Рисунок 1.3 – Методи забезпечення відмовостійкості

На рисунку 1.4 показано, як часто в літературі вказується на конкретний тип несправності. Можна помітити, що 43% фреймворків націлені на збої, 26% націлені на візантійські помилки, а 31% націлені на обидві. Це показує, що дослідники дещо більше мотивовані до стійкості до збоїв. Це може бути пов’язано з тим, що виявлення та стійкість до аварійних помилок легше моделювати порівняно з візантійськими помилками. Однак можуть бути й інші причини.

Залежно від дослідження зрозуміло, що наразі існує велика кількість відмовостійких стратегії, зосереджених головним чином на реактивному та проактивному підходах. Ефективність, універсальність і стійкість цих підходів необхідно довести в сенсі хмарних систем.

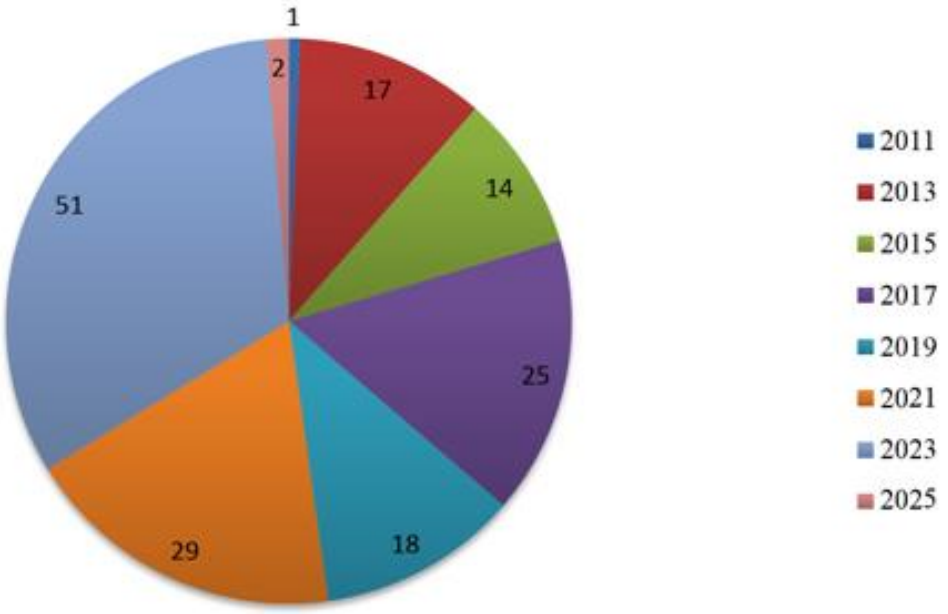

Рисунок 1.4 – Кількість значущих робот в області дослідження

Для деяких підходів незрозуміло, чи працюватимуть вони на системах, реалізованих на розподілених і гетерогенних хмарних платформах. Існуючі методи не є універсальними і вимагають певної кількості ручного втручання для їх безперебійного функціонування та специфікації. Тому пропонується, щоб автоматизація більш поширювалась у майбутньому на відмовостійкість хмарі.

1.3 Самовідновлення програмного забезпечення хмарних інформаційних систем

Самовідновлювальні хмарні системи розроблені для виявлення та усунення збоїв у продуктивності без участі людей і підтримки здорового хмарного середовища. Вони ефективно використані розробниками програмного забезпечення та хмарними інженерами для підвищення ефективності хмарних сервісів і забезпечення безперервної доступності сервісів [13]. Ці системи можуть зменшити час простою та максимізувати ефективне використання хмарних служб. Його також було оптимізовано для зменшення ручного втручання, коли інформаційні технології підтверджують

виконання стратегічних завдань.

Концепція самовідновлюваних хмарних систем значною мірою покладається на автоматизований аналіз причин, який дозволяє спостереженнями визначити причини шляхом відстеження потоків додатків або інших подій. Крім того, хмарні системи з самовідновленням допомагають упроваджувати оновлення в хмарних умовах і застосовувати стратегії для вирішення нових майбутніх загроз. Концепція автоматизованого аналізу причин базується на визначенні причин збоїв, якими можуть бути перевантаження мережі, помилки програмного забезпечення та апаратні проблеми. Хмарні системи самовідновлення виконуються для таких дій відновлення, як перезапуск служб, перерозподіл ресурсів і перенаправлення мережевого трафіку після виявлення потенційних причин.

Заявлена мета дослідження – вивчити дизайн самовідновлюваних хмарних систем, які покращують автономність і стійкість у безперервній доступності послуг з мінімальною участю людини, щоб уникнути збоїв у хмарних службах – безпосередньо пов’язана з перевагами та принципами системи. Це пов’язано з тактикою пом’якшення викликів через використання машинного навчання, автоматизації та штучного інтелекту.

Загальне дослідження висвітлило різні аспекти самовідновлюваних хмарних систем, щоб зрозуміти їх ефективність, що відображає автономність і стійкість. Воно передбачає успішну інтеграцію елементів самовідновлюваних хмарних систем, таких як виявлення проблем, моніторинг і автоматичне відновлення з наступним пом’якшенням проблем під час його впровадження. З цієї причини це дослідження сприяло адаптивній розробці стратегічних хмарних систем, які можуть ефективно працювати з мінімальним часом простою.

Існує декілька структур систем самовідновлення, заснованих на програмному забезпеченні, що враховують стратегію ASSURE, використовуючи віртуалізацію відмов і стратегію WRP, щоб допомогти серверним програмам уникнути пошкоджених маршрутів. Вони можуть

підтримувати вияв несправностей у програмному забезпеченні та гарантувати відмінне обслуговування в хмарних системах. Самовідновлювання використовується двома способами. На рисунку 1.5 показана архітектура системи забезпечення відмовостійкості з використанням методів самовідновлення.

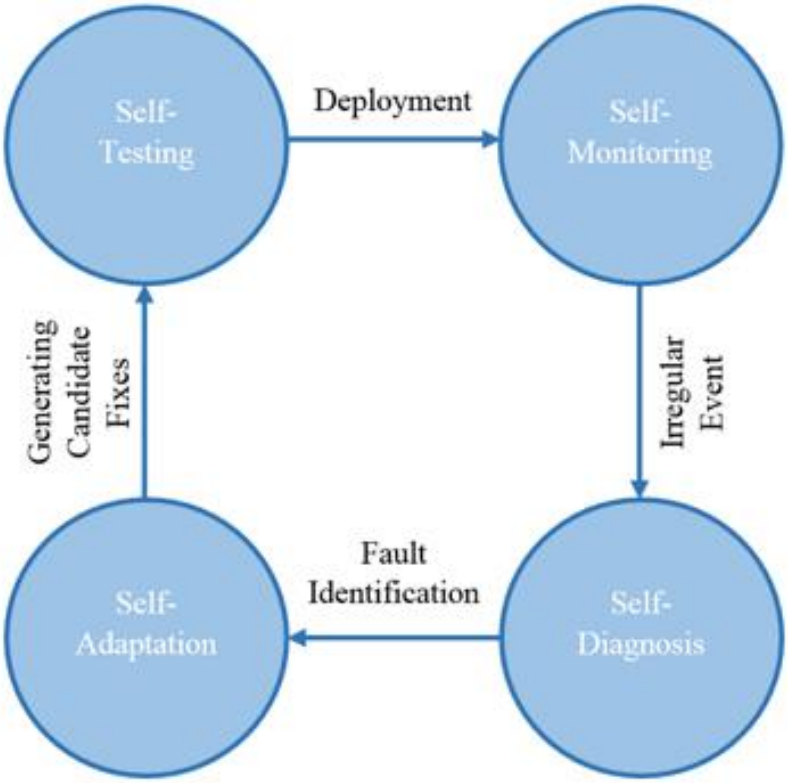

Рисунок 1.5 – Архітектура системи забезпечення відмовостійкості з використанням методів самовідновлення

WRP використовується для відновлення після дефектів, які важко контролювати ASSURE. Використовується впровадження дворівневого репозиторію RP, який дозволяє нам зв’язувати знання про помилки з додатками, корисними для відновлення більшої кількості збоїв. Виконується контроль помилок для ситуацій, коли одна і та ж програма виконується на різних віртуальних машинах, встановлених на одній комп’ютерній системі. Як контрольна точка та платформа відкату, системи вимагають контрольної точки/перезапуску (BLCR).

1.4 Основні використані концепції організації самовідновлення

Перша концепція – це теорія автономних обчислень. Концепція автономних обчислень була започаткована міжнародною корпорацією бізнес-машин, яка передбачала модель обчислень із самокеруванням, здатну автоматично відновлювати, конфігурувати, оптимізувати та захищати. Стандартом вимагається наявність класу, який забезпечує автономне обчислення (рисунок 1.6).

```python
class AutonomicComputing:
    def __init__(self):
        self.system_state = "optimal"

    def monitor_system(self):
        # Simulate system monitoring
        return "System stable"

    def detect_issue(self):
        # Detect potential issues
        if self.system_state != "optimal":
            return True
        return False

    def heal(self):
        # Self-healing process
        if self.detect_issue():
            self.system_state = "optimal"
            return "Issue resolved"
        return "No issues detected"

# Example usage
ac = AutonomicComputing()
ac.monitor_system()
ac.heal()
```

Рисунок 1.6 – Об'явлення класу автоматичних обчислень

Ця теорія узгоджується з розробкою системи, яка допомагає в управлінні операціями для досягнення цілей, поставлених адміністраторами

[14]. Застосування теорії автономних обчислень тісно пов’язане з хмарними системами, що самовідновлюються, де розвиток механізмів самовідновлення дозволяє покращити хмарну інфраструктуру для пом’якшення таких проблем, як перевантаження мережі та помилки програмного забезпечення.

Друга концепція – теорія стійкості. Розробка стійкості – це багатогранний підхід, який допомагає розробляти програмні системи, які можуть ефективно відновлювати раптові збої та підтримувати постійну роботу служб. Ця теорія допомагає зрозуміти різні аспекти, такі як організаційне управління та розробка для розвитку хмарних систем, які можуть підтримувати раптові збої та функціональні проблеми в негативній атмосфері [6]. Відповідно до цієї теорії хмарні системи, що самовідновлюються, повинні залишатися в робочому стані, незважаючи на будь-які збої. Ця теорія передбачає, що хмарні системи повинні прагнути до самовідновлення та забезпечення постійної доступності послуг (рисунок 1.7).

```python
class ResilienceEngineering:
    def __init__(self):
        self.service_availability = 99.9  # percentage

    def detect_disruption(self):
        # Simulate detection of a disruption
        return "Disruption detected"

    def recover(self):
        # Self-healing process
        self.service_availability = 100
        return "Service availability restored"

# Example usage
re = ResilienceEngineering()
re.detect_disruption()
re.recover()
```

Рисунок 1.7 – Об'явлення класу забезпечення самовідновлення

Під час вивчення літератури було виявлено кілька недоліків, які створили перешкоди для розуміння ефективності хмарних систем, що самовідновлюються, у різних хмарних умовах. Інформація про ці системи обмежена або неповна.

1.5 Постановка мети та завдань роботи

Метою роботи є розроблення адаптивного методу забезпечення відмовостійкості у хмарному середовищі з використанням самовідновлення, який б динамічно обирал найбільш ефективний метод fault-tolerance (реплікація або контрольні точки) на основі аналізу ризику відмов, пріоритету завдання та доступних ресурсів.

Завдання дослідження:

- проаналізувати сучасні підходи до fault-tolerance у хмарних системах;
- сформулювати критерії вибору між методами реплікації та контрольних точок;
- запровадити метод класифікації рівня ризику машин з використанням KNN;
- розробити систему нечіткого висновку FTMSFIS для прийняття рішень;
- провести моделювання та оцінку ефективності системи в умовах хмарного середовища.

Об'єкт дослідження: процеси забезпечення відмовостійкості та самовідновлення у хмарних обчислювальних системах.

Предмет дослідження: методи й моделі адаптивного вибору механізмів fault-tolerance (реплікація або контрольні точки) на основі класифікації рівня ризику фізичних машин, нечіткого логічного висновку та пріоритетів користувацьких завдань у хмарному середовищі.

2 МЕТОДИ УПРАВЛІННЯ РОЗПОДІЛЕНИМ ОБЧИСЛЮВАЛЬНИМ ПРОЦЕСОМ В ГЕТЕРОГЕННИХ ХМАРНИХ СИСТЕМАХ

2.1 Методологія організації самовідновлення

Методологія проектування відіграє життєво важливу роль у висуненні ключових припущень і переконань, пов’язаних із певною темою. Це важлива методологічна парадигма, яка визначає, які дані потрібно зібрати та оцінити для досягнення цілей дослідження. Визначення відповідної концепції дослідження залежить від типу знань, які будуть отримані в результаті дослідження, оскільки воно спрямовує шляхи досягнення встановлених переваг. В роботі була використана парадигма інтерпретативізму, яка зосереджена на інтерпретації ідентифікованих елементів дослідження шляхом оптимізації відповідних моделей і теорій. Основна мета вибору цієї конкретної філософії полягає в тому, що дослідження було зосереджено на зборі якісних даних, які підтримуються відповідними теоріями.

Дослідницькі підходи - це важливі плани та процедури, які використовуються для визначення етапів збору, інтерпретації та оцінки даних. Залежно від визначеного підходу можна розпізнати процедури збору, аналізу та інтерпретації даних. Основною метою дослідницького підходу є спостереження за систематичними планами, які можуть передбачати напрямки виконання методології. У цьому дослідженні було використано індуктивний дослідницький підхід, який відображає оптимізацію значущих моделей для збору необхідної інформації та пропозиції детальної оцінки обраної теми. Включення індуктивного підходу допомогло зв’язати теорії та моделі, за допомогою яких дані були організовані для досягнення поставлених цілей.

Концепція дизайну взаємопов’язана з виконанням дизайну та планів дослідження шляхом точної організації зібраних даних. Це комплексна

техніка методології, яка допомагає генерувати точні відповіді на поставлені питання по темі. Дизайн дослідження дозволяє створювати важливі процедури, які можуть відповідати цілям і завданням, підтримуючи методи аналізу даних. У цьому дослідженні були використані переваги пояснювального дослідження, яке має на меті з'ясувати причини, фактори та результати наявності будь-якого інциденту шляхом детального пояснення, яке може вплинути на наявну інформацію. Пояснювальний дизайн допоміг у зборі даних та організації кількох сегментів шляхом узгодження визначених цілей.

Методи збору даних можна визначити як комплексні підходи, які допомагають у зборі відповідної інформації з різних джерел. Це відома систематична техніка, яка забезпечує кращу якість, точність і адаптивність зібраної інформації таким чином, за допомогою якої можна відповідати поставленим дослідницьким питанням або гіпотезам. У цьому дослідженні був використаний метод збору вторинних даних, який допоміг у вивченні інформації, наявної в існуючих джерелах. Цей метод збору даних допоміг у зборі інформації з онлайн-журналів, статей, публікацій новин тощо.

Під час включення досліджуваних методологічних прийомів дотримувались відповідних етичних міркувань. Для збору точної інформації використовувалися вторинні джерела, які мають певний доступ і відповідають заявленим цілям. Онлайн-журнали, статті, електронні книги та інші джерела, опубліковані з 2018 по 2024 рік, використовувалися для збору оновленої інформації та підтримки результатів досліджень.

Хмарні системи, що самовідновлюються, значною мірою залежать від інновацій і технологічних досягнень, таких як автоматизація, машинне навчання та штучний інтелект, щоб підтримувати стійкість і безперервність роботи. Ці системи використовують машинне навчання для розуміння хмарної системи, де її конфігурація дозволяє розпізнавати потенційні збої. Вона також оцінює автоматизовані процедури відновлення, щоб відобразити їх ефективність у покращенні результатів, продуктивності та надійності

системи. У цьому терміні можна спостерігати деякі ключові показники, такі як скорочення часу простою, покращення безперервності обслуговування та підвищення ефективності роботи. Однак під час оптимізації самовідновлювальних хмарних систем можуть виникнути проблеми через різноманітність хмарних платформ, погані застарілі системи, проблеми з масштабованістю та виправлення безпеки.

2.2 Пропонований підхід до реалізації самовідновлення

У роботі представлено адаптивний підхід до відмовостійкості в хмарах IaaS, які надають обчислювальні ресурси як послугу кінцевим користувачам за допомогою технології віртуалізації. Фізична машина (PM), наявна в хмарі, здатна розміщувати одну або кілька віртуальних машин (VM) залежно від власної конфігурації та конфігурації розміщеної віртуальної машини. Передбачається, що кожна віртуальна машина виконує завдання користувача. Збій у роботі віртуальної машини призведе до запобігання або затримки виконання завдання, що призведе до зниження задоволеності користувачів, а також втрати для хмарного постачальника. Віртуальна машина може вийти з ладу через численні причини, пов’язані з програмним або апаратним забезпеченням. З них збої програмного забезпечення є результатом виконання завдання або програми на віртуальній машині та виходять за межі сфери діяльності постачальника послуг. Однак збій PM – це ризик, який можна передбачити на основі історичних даних PM. Таким чином, ця робота використовує дані про попередні збої PM, щоб позначити PM відповідно до ризику відмови. Згодом мітки ризику PM беруться разом із рівнем пріоритету завдання та рівнем сховища, доступним у хмарі, щоб визначити підхід до відмовостійкості, який підходить для завдання.

Як перший крок до самовідновлення ми розглядаємо фізичні машини, на яких розміщені віртуальні машини. Дані, пов’язані з минулими випадками збоїв ПМ, аналізуються з метою визначення ризику виходу з ладу ПМ у

майбутньому. Набір даних від хмарного постачальника містить різні функції щодо збою PM. У роботі використовувалися такі ознаки для маркування ризиків: причина збою, час простою під час збою, тип фізичної машини та її призначення. Причина несправності вказує на ймовірність повторення несправності чи ні. Наприклад, апаратні збої можуть повторюватися, оскільки вони пов'язані з машиною, але програмний збій може бути спричинений розміщеною віртуальною машиною або завданням, виконаним на ній, і навряд чи повториться. Час простою використовується, оскільки очікується, що він відображає серйозність несправності.

Алгоритм KNN використовувався для неконтрольованої кластеризації даних про несправності. Це непараметричний алгоритм, який не передбачає жодного конкретного розподілу даних і, таким чином, відповідає реальним даним про помилки. Крім того, дані не позначаються, і KNN використовується для того, щоб спочатку згрупувати дані в три кластери на основі схожості функцій (Java, n.d.). Потім сформовані кластери були проаналізовані та отримали позначку ризику, тобто високий, середній та низький. Таким чином, PM, що знаходяться в кластері з мітками (високий/середній/низький), класифікуються як PM з (високим/середнім/низьким) ризиком відмови. Кластери, сформовані за допомогою алгоритму KNN на наборі даних NERSC, і прогнозовані результати та їх аналіз з точки зору точності, запам'ятовування, а також вимірювання F1 наведено в таблиці 2.1.

Таблиці 2.1 – Статистичний аналіз ризиків відмов

	Високий	Середній	Низький
Точність	100	75	100
Перезапуск	100	100	90
Вияв збоїв	100	85	95

Загальна отримана точність становить 99,98%. Згодом виконується налаштування мітки для кластеризованого значення, щоб позначити машину як високий, помірний і низький ризик.

Нечітка логіка – це метод міркування та прийняття рішень, який часто використовується в системах, що містять невизначену, неточну або розпливчасту інформацію. Він заснований на концепції відносного градуйованого членства, мотивованого людським сприйняттям і процесом пізнання. Нечітка логіка застосовна в задачах, пов’язаних із людським досвідом, невеликою кількістю даних або там, де вхідні параметри є лінгвістичними за своєю природою або не мають чітких меж. Оскільки він забезпечує ефективний спосіб вирішення багатокритеріальних проблем прийняття рішень, він використовувався в розробці інтелектуальних систем для оптимізації, розпізнавання шаблонів, контролю, оцінки ризиків проекту, систем оцінки довіри для провайдерів хмарних послуг та багато іншого.

У роботі використовується нечітка логіка для отримання ефективного методу FT у хмарі. Вибір нечіткої логіки залежить від нечіткої природи вхідних параметрів доступності сховища, ризику відмови машини та рівня пріоритету користувача. Постулюється, що система на основі нечіткої логіки, яка нагадує людське пізнання, дасть ефективні результати для поточної проблеми.

Запропонована система на основі нечітких правил, яка називається системою нечіткого висновку щодо вибору методу відмовостійкості (FTMSFIS), вибирає відповідний метод FT для обробки збоїв у хмарі. FTMSFIS – це система нечіткого висновку для адаптації вхідних параметрів до двох вихідних класів реплікації та контрольних точок.

Рисунок 2.1 демонструє модель FTMSFIS. Ця нечітка система складається з трьох атрибутів як вхідних змінних, а саме: ризик машини, доступність сховища і пріоритет завдання. Єдиною вихідною змінною є метод відмовостійкості. Детальний опис значень для вхідних і вихідних змінних наведено в таблиці 2.2.

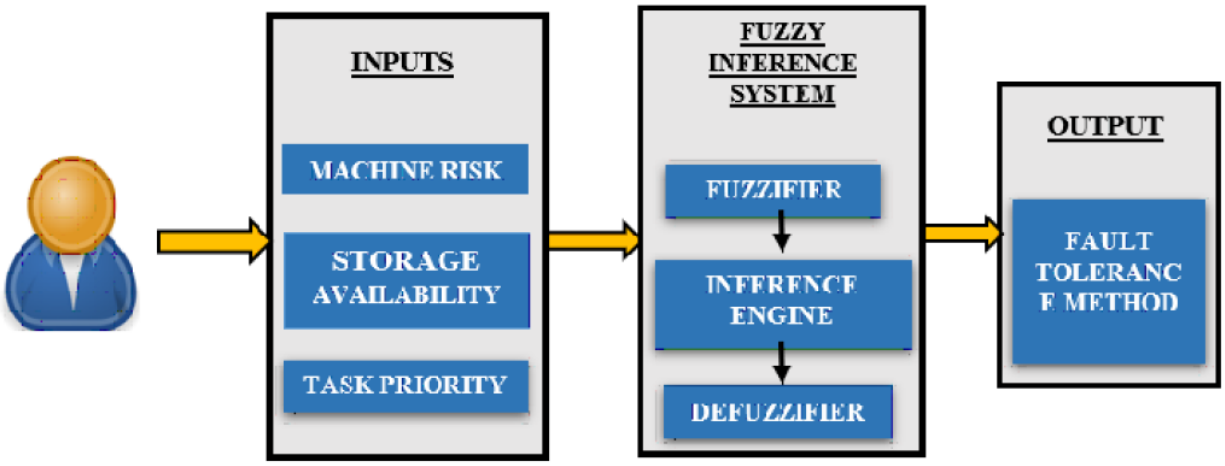

Рисунок 2.1 – Запропонована модель забезпечення відмовостійкості

Таблиця 2.2 – Системні характеристики для фази-логіки

Змінні	Тип даних	Рівень ризику/назва методу	Кінцеві точки інтервалу	
			Діапазон	Точки
Ризик	Вхідні	Low	(0-2)	(0,0,1)
		Medium		(0,1,2)
		High		(0,2,2)
Доступність сховища даних	Вхідні	Low	(0-100)	(0,0,15,30)
		Medium		(15,50,85)
		High		(30,60,100)
Пріоритет завдання	Вхідні	Low	(0-2)	(0,0,5)
		Medium		(3,6,9)
		High		(6,8,10)
Метод самовідновлення	Вихідні	Контрольні точки	(0-90)	(0,35,60)
		Реплікація		(35, 50, 90)

Система нечіткої логіки складається з фазифікатора, механізму логічного висновку та дефазифікатора. Термін фазифікатор використовується для перекладу вхідних параметрів у лінгвістичні терміни. Дефазифікатор перетворює результат механізму логічного висновку в чітке значення.

Механізм логічного висновку відповідає за отримання вихідних даних за допомогою чітко визначених правил і зберігається в базі даних.

Машинний ризик – це змінна, яка вказує на типи ризику, пов’язані з фізичною машиною. Ця змінна представлена за шкалою 0-2. Змінна позначається за допомогою 3 нечітких наборів, таких як низький, помірний і високий.

Доступність сховища вказує на обсяг сховища, доступний на конкретній фізичній машині. Вона описується у відсотках, тобто за шкалою від 0 до 100. Ця змінна представлена за допомогою трьох нечітких наборів, таких як менший, середній і високий.

Пріоритет завдання означає пріоритет, пов’язаний із завданнями, тобто виконанням на віртуальній машині. Зазвичай він задається користувачем, але може визначатися і системою. Ця змінна представлена за шкалою від 0 до 10 за допомогою 3 нечітких наборів: низького, середнього та високого.

Нечітка система має лише одну вихідну змінну, тобто метод самовідновлення. Вона вказує, який тип техніки відмовостійкості вибрано для завчасного вирішення проблем. Ця змінна представлена за допомогою двох нечітких наборів контрольних точок і реплікації та позначена за шкалою в діапазоні 0-80.

Функція приналежності для вхідних і вихідних змінних: для зменшення ускладнень, функція приналежності трапеції та трикутника використовується як для вхідних, так і для вихідних змінних. Приналежність нечіткого набору змінної машинного ризику, доступності сховища, пріоритету завдання та методу відмовостійкості описується наступним чином.

Машинний ризик – ця змінна визначається за допомогою трьох нечітких наборів: низький, помірний і високий, як показано на рисунку 2.2. Горизонтальна та вертикальна осі позначають вхідне значення та ступінь приналежності атрибута MR відповідно.

Доступність пам’яті – ця змінна визначається за допомогою трьох нечітких наборів (низько, середньо та високо), як показано на малюнку 2.3.

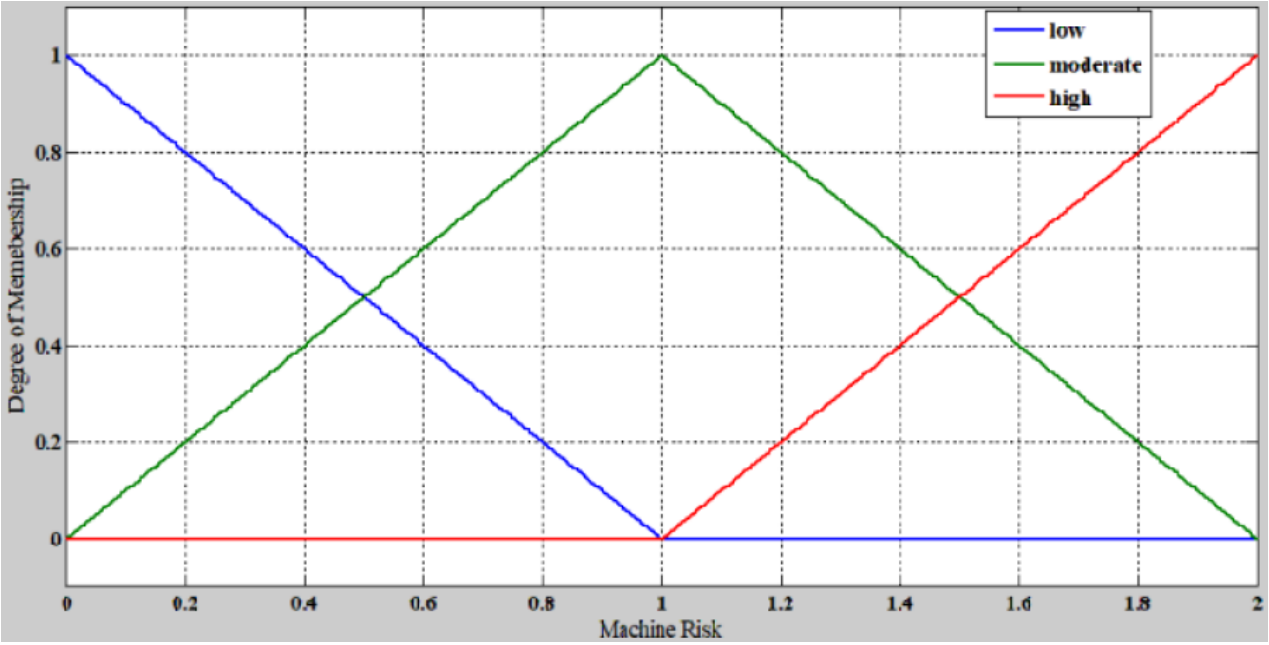

Рисунок 2.2 – Нечітке правило машинного ризику (MR)

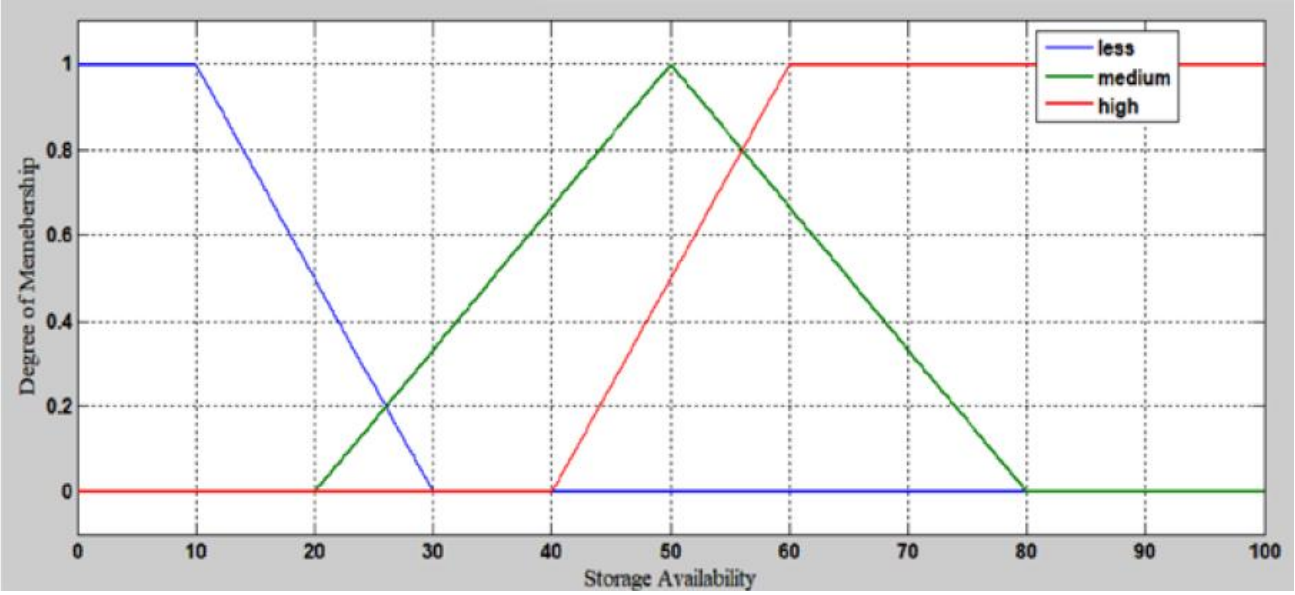

Рисунок 2.3 – Нечітке правило для доступності сховища даних (SA)

Пріоритет завдання – ця змінна визначається за допомогою 3 нечітких наборів: низький, середній і високий, як показано на рисунку 2.4. Вона може визначатися користувачем, який надіслав завдання.

Метод відмовостійкості визначається за допомогою двох нечітких наборів контрольних точок і реплікації, як показано на рисунку 2.5. Вибір методу забезпечення відмовостійкості здійснюється системою.

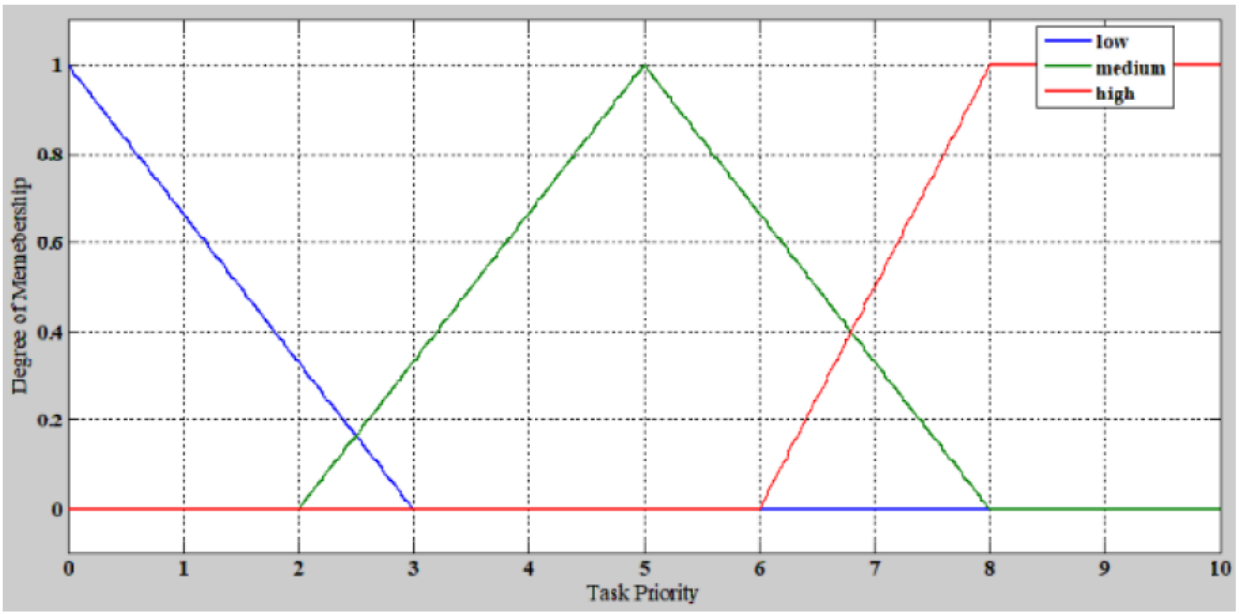

Рисунок 2.4 – Нечітке правило для пріоритету завдання (TP)

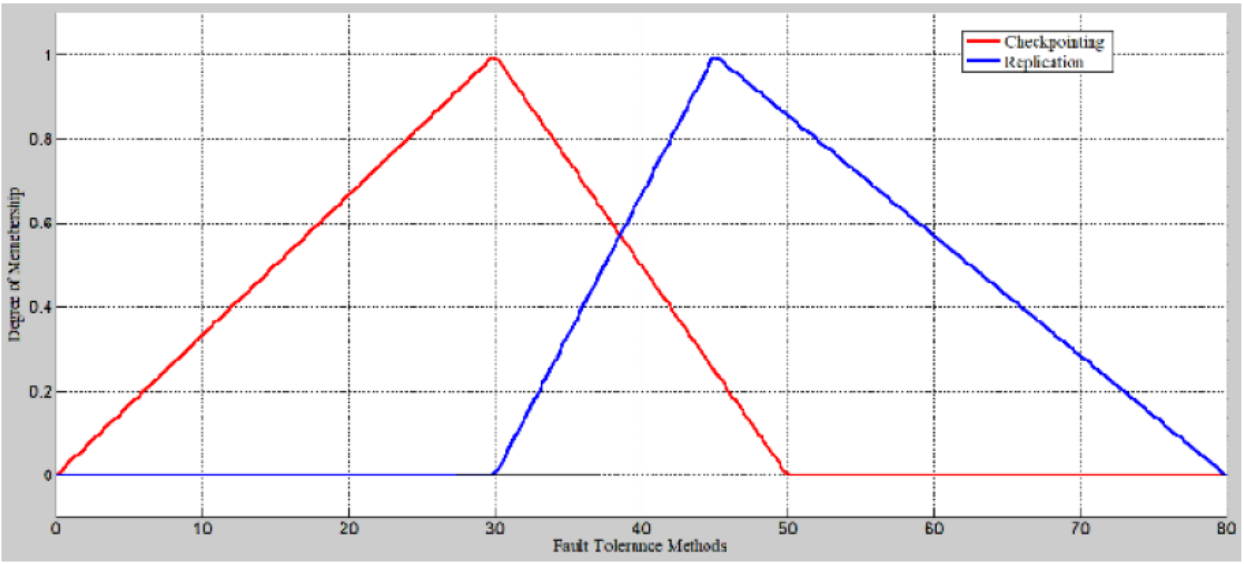

Рисунок 2.5 – Нечітке правило для вибору метода самовідновлення (FTM)

2.3 Нечіткі правила

Для визначення значення змінної FTM (вихідної змінної) на основі значень 3 вхідних змінних створено набір із 27 нечітких правил. Щоб сформувати нечіткі правила, ми дотримувалися певних умов:

Випадок 1: якщо доступність пам’яті на фізичній машині висока, а

пріоритет завдання, надісланого користувачем, високий, тоді, незалежно від значення ризику машини, для вирішення проблеми вибирається метод відмовостійкості реплікації.

Випадок 2: якщо доступність пам’яті на фізичній машині висока або середня, а пріоритет завдання, поданого користувачем, низький, тоді можуть бути дві підумови:

- якщо фізичний ризик машини високий, для вирішення проблеми вибирається метод відмовостійкості реплікації;
- якщо фізичний ризик машини низький, тоді для вирішення проблеми вибирається метод відмовостійкості контрольних точок.

Випадок 3. У разі низької доступності пам’яті на фізичній машині, незалежно від значення пріоритету завдання та ризику машини, для вирішення проблеми вибирається метод відмовостійкості контрольних точок.

27 правил нечіткого логічного висновку враховують усі можливі комбінації визначених вище умов (від випадку 1 до випадку 3) для вхідних змінних. Розглядаючи випадок 1, випадок 2 і випадок 3, для кожного розроблено три правила. Отже, загальна кількість рамкових правил становить 3*3*3 = 27. 27 правил для запропонованої нечіткої системи проілюстровано в таблиці 2.3.

Згодом, оцінюючи правила відповідно до заданих умов, виявили, що деякі правила можна об’єднати, щоб отримати мінімізований набір правил для подальшої оптимізації запропонованої нечіткої системи. Наприклад, правила 1-3 вказують на те, що якщо ризик машини низький, а доступність пам’яті менша, то, незалежно від пріоритету завдання, FT-метод, який буде використано, є контрольними точками. Таким чином, правила 1-3 у таблиці 2.3 об’єднані, щоб сформувати нове правило 1 у таблиці 2.3. Подібним чином, правила 10–12 у таблиці 2.3 об’єднано, щоб сформувати правило 8 у таблиці 4, а 19–21 об’єднано, щоб сформувати правило 15 у таблиці 2.4. Дотримуючись того самого підходу, правила з 22 по 24 і з 25 по 27 об’єднано, щоб сформувати правила 16 і 17. відповідно в таблиці 2.4. Таким

чином, після об’єднання кількох правил загальна кількість нечітких правил для запропонованої нечіткої системи становить 17.

Таблиці 2.3 – Пропоновані правила для нечіткої логіки

Правило	Машинний ризик	Доступність сховища	Пріоритет завдання	Метод самовідновлення
1	2	3	4	5
1	Низький	Низька	Низький	Контр. точка
2	Низький	Низька	Середній	Контр. точка
3	Низький	Низька	Високий	Контр. точка
4	Низький	Середня	Низький	Контр. точка
5	Низький	Середня	Середній	Контр. точка
6	Низький	Середня	Високий	Реплікація
7	Низький	Висока	Низький	Контр. точка
8	Низький	Висока	Середній	Контр. точка
9	Низький	Висока	Високий	Реплікація
10	Середній	Низька	Низький	Контр. точка
11	Середній	Низька	Середній	Контр. точка
12	Середній	Низька	Високий	Контр. точка
13	Середній	Середня	Низький	Контр. точка
14	Середній	Середня	Середній	Контр. точка
15	Середній	Середня	Високий	Реплікація
16	Середній	Висока	Низький	Контр. точка
17	Середній	Висока	Середній	Контр. точка
18	Середній	Висока	Високий	Реплікація
19	Високий	Низька	Низький	Контр. точка
20	Високий	Низька	Середній	Контр. точка
21	Високий	Низька	Високий	Контр. точка
22	Високий	Середня	Низький	Реплікація

Продовження таблиці 2.3

1	2	3	4	5
23	Високий	Середня	Середній	Реплікація
24	Високий	Середня	Високий	Реплікація
25	Високий	Висока	Низький	Реплікація
26	Високий	Висока	Середній	Реплікація
27	Високий	Висока	Високий	Реплікація

Таблиці 2.4 – Мінімізовані правила для нечіткої логіки

Правило	Машинний ризик	Доступність сховища	Пріоритет завдання	Метод самовідновлення
1	2	3	4	5
1	Низький	Низька	Н/С/В	Контр. точка
2	Низький	Середня	Низький	Контр. точка
3	Низький	Середня	Високий	Контр. точка
4	Низький	Середня	Низький	Контр. точка
5	Низький	Висока	Низький	Контр. точка
6	Низький	Висока	Середній	Реплікація
7	Низький	Висока	Високий	Контр. точка
8	Середній	Низька	Н/С/В	Контр. точка
9	Середній	Висока	Низький	Реплікація
10	Середній	Середня	Середній	Контр. точка
11	Середній	Середня	Середній	Контр. точка
12	Середній	Висока	Низький	Контр. точка
13	Середній	Висока	Середній	Контр. точка
14	Середній	Висока	Високий	Контр. точка
15	Середній	Низька	Н/С/В	Реплікація
16	Середній	Середня	Н/С/В	Контр. точка
17	Середній	Висока	Н/С/В	Контр. точка

Скорочений набір нечітких правил проілюстровано в таблиці 2.4. Крім того, малюнок 2.6 ілюструє дизайн запропонованої нечіткої системи в середовищі MATLAB. Фазифікатор перетворює вхідні атрибути в лінгвістичний набір термінів за допомогою функцій належності. Результат фазифікатора передається механізму логічного висновку, який використовує чітко визначені правила бази даних для забезпечення найкращого результату.

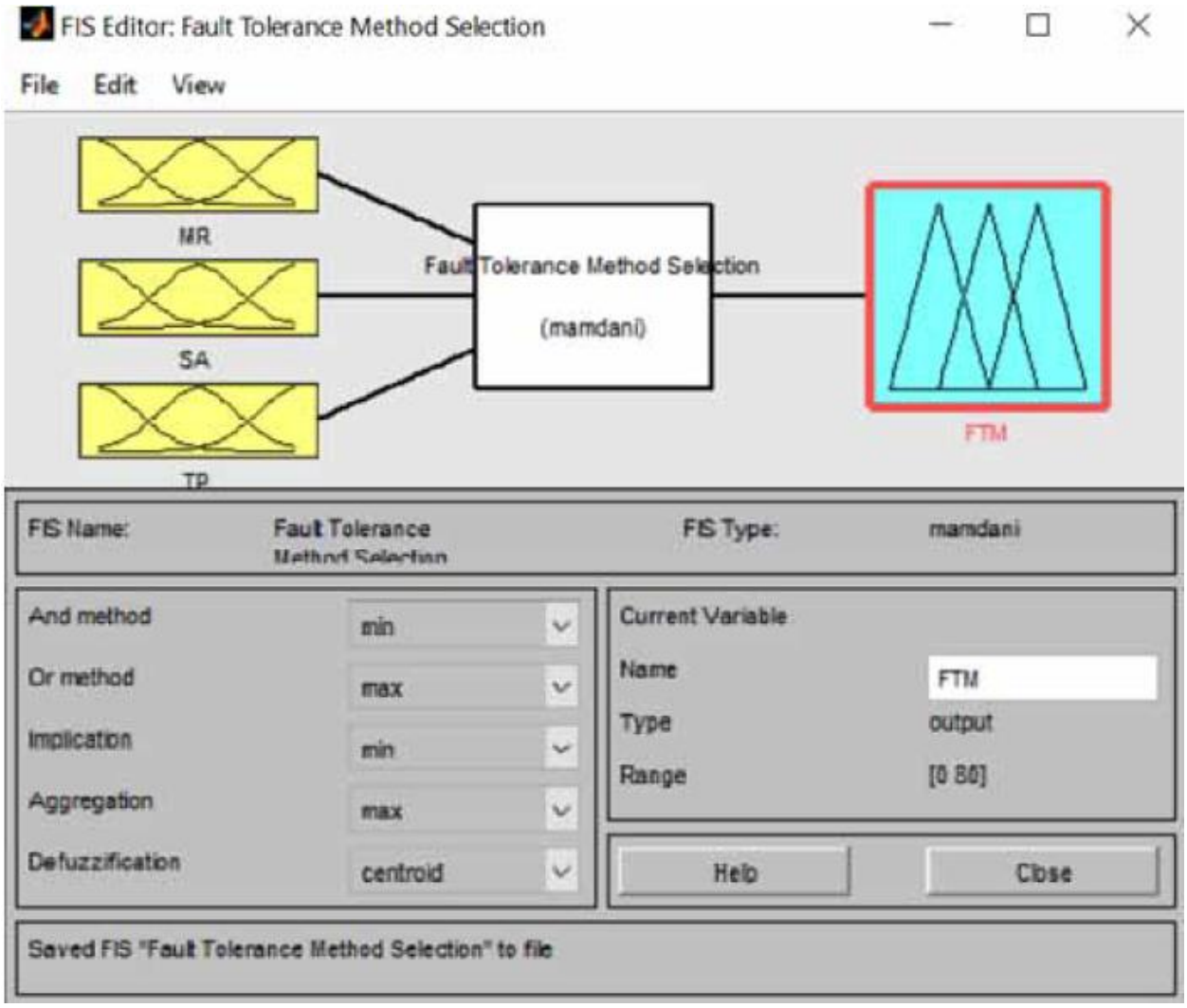

Рисунок 2.6 – Реалізація фазифікатора в MATLAB

Щоб представити кінцевий вихід системи, отриманий нечіткий результат надсилається до дефазифікатора, який перетворює результат у чітке значення. Дефазифікація виконується методом центроїда. FTM системи оцінюється в стані контрольних точок або реплікації на основі результату

дефазифікатора. Правила бази даних механізму висновку FTMSFIS, які були виконані в MATLAB, показані на рисунку 2.7.

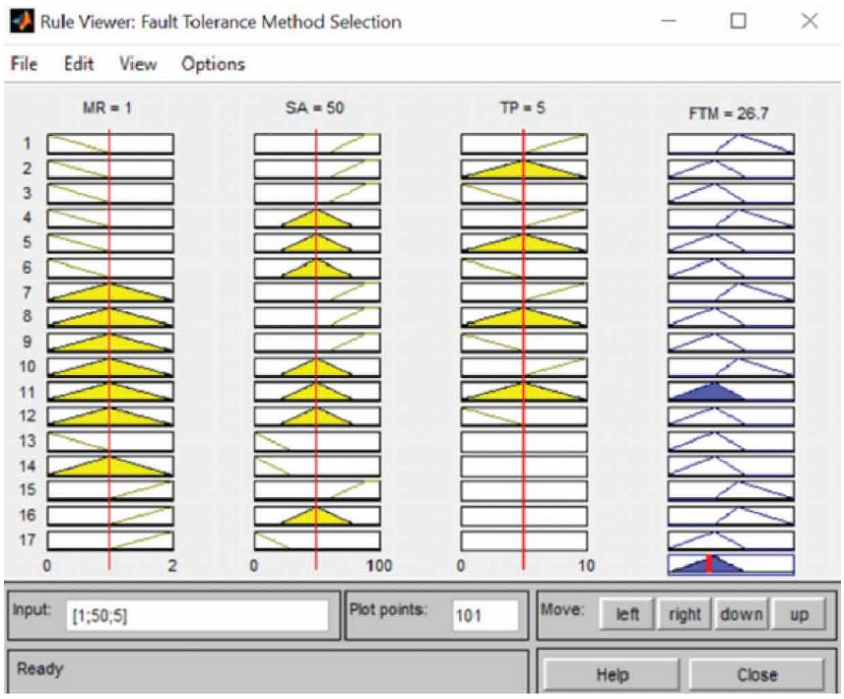

Рисунок 2.7 – Правила бази даних механізму формування висновку

На рисунку 2.8 зображено інтерпретацію передбаченої нечіткої системи за допомогою Surface Viewer.

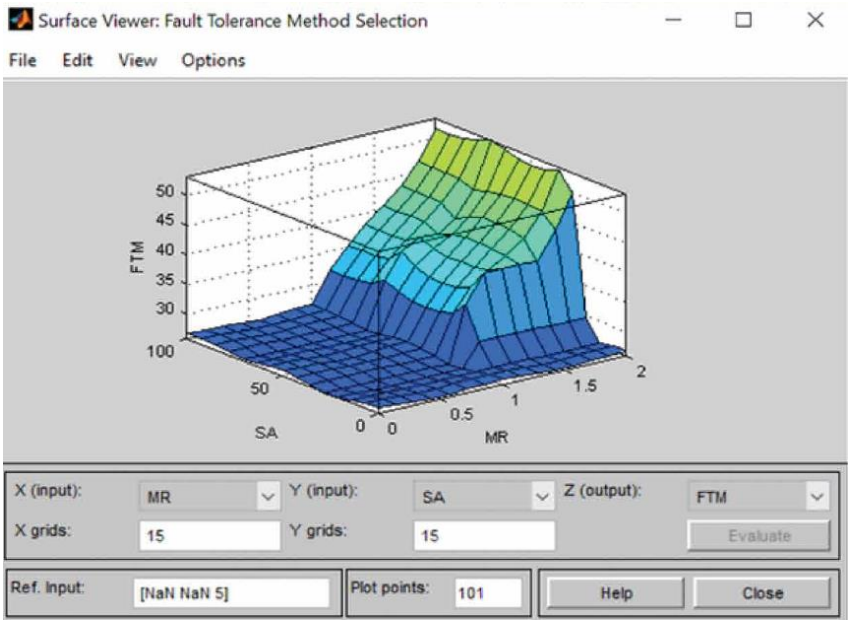

Рисунок 2.8 – Передбачення нечіткої системи

3 РЕАЛІЗАЦІЯ ТА МОДЕЛЮВАННЯ

3.1 Параметри та метрики експериментів

У цьому розділі ефективність запроектованої схеми було оцінено за допомогою експериментів на основі моделювання. Запропонована схема порівнюється з методом реплікації та контрольних точок щодо загальної кількості успішних відновлень, а також максимального споживання пам’яті. Симуляції виконуються на ПК з процесором Intel® Core (TM) i5-7200, що працює на частоті 2,7 ГГц, і 8 ГБ оперативної пам’яті. Ми припускаємо, що кожна PM може мати від 0 до 4 віртуальних машин для виконання поставлених завдань.

Розглянемо параметри оцінювання.

Відновлюваність: цей показник використовується для розрахунку відсотка успішного відновлення віртуальної машини та визначається у рівнянні:

$$R = \frac{N_S}{N_F}, \tag{3.1}$$

де N_S – кількість успішних відновлювань;

N_F – кількість збоїв.

Максимальне споживання пам’яті (MSC): цей показник використовується для обчислення максимального обсягу пам’яті, що споживається PM під час виконання методів контрольних точок і реплікації, а також запропонованого методу. MSC визначається у рівняннях

$$M_{repl} = \sum_{1}^{n} pm_i \bullet v \bullet s \ , \tag{3.2}$$

$$M_{check} = \sum_{1}^{n} pm_i \bullet v \bullet cpu \ , \tag{3.3}$$

$$M_{prop} = \sum_{1}^{h} pm_i \bullet v \bullet s + \sum_{j}^{l} pm_j \bullet v \bullet cpu \ , \tag{3.2}$$

де – n представляє загальну кількість PM;

v – означає кількість віртуальних машин на кожному PM;

s – позначає розмір кожної віртуальної машини;

cpu – позначає розмір зображення контрольної точки кожної віртуальної машини;

h – позначає загальну кількість машин високого ризику;

l – позначає загальну кількість машин з низьким рівнем ризику.

3.2 Організація експериментів та аналіз результатів

Хмарні системи, що самовідновлюються, значною мірою залежать від інновацій і технологічних досягнень, таких як автоматизація, машинне навчання та штучний інтелект, щоб підтримувати стійкість і безперервність роботи. Ці системи використовують машинне навчання для розуміння хмарної атмосфери, де її конфігурація дозволяє розпізнавати потенційні збої. Вони також оцінюють автоматизовані процедури відновлення, щоб відобразити їх ефективність у покращенні результатів, продуктивності та надійності системи. У цьому контексті можна спостерігати деякі ключові показники, такі як скорочення часу простою, покращення безперервності обслуговування та підвищення ефективності роботи. Однак під час оптимізації самовідновлювальних хмарних систем можуть виникнути проблеми через різноманітність хмарних платформ, погані застарілі системи, проблеми з масштабованістю та виправлення безпеки.

На рисунку 3.1 показано клас ініціалізації системи самовідновлення та його методи.

```python
import random

class SelfHealingSystem:
    def __init__(self):
        self.downtime = 0
        self.performance = 100  # percentage

    def monitor_performance(self):
        # Simulate random performance drops
        self.performance -= random.randint(0, 20)
        return self.performance

    def automated_heal(self):
        if self.performance < 80:
            self.downtime += 1
            self.performance = 100  # Restore performance
        return f"Downtime: {self.downtime}, Performance: {self.performance}"

# Example usage
shs = SelfHealingSystem()
shs.monitor_performance()
shs.automated_heal()
```

Рисунок 3.1 – Клас ініціалізації системи самовідновлення

Стратегія впровадження самовідновлюваних хмарних систем включає в себе прийняття важливих стратегій, таких як використання машинного навчання та штучного інтелекту для виявлення аномалій, інтеграція розширених інструментів моніторингу та автоматизація процесів відновлення. Потенційні проблеми, з якими можна зіткнутися під час розвитку хмарних систем, що самовідновлюються, такі як складності в різноманітних хмарних середовищах, проблеми з масштабованістю алгоритмів відновлення, погане планування та негативні операційні перешкоди. Крім того, можна зрозуміти, що впровадження стратегічної оцінки підвищує подальший успіх і передбачає кращу системну інтеграцію та узгоджені сервісні засоби за допомогою хмарних сервісів, що самовідновлюються.

Ефективність роботи самовідновлюваних хмарних систем значною мірою залежить від необхідних дозволів адміністраторів хмари для

подальшого оновлення загальної інфраструктури. Ці системи також потребують адаптивного контролю доступності, за допомогою якого їх можна захистити від несанкціонованого доступу, щоб запобігти витоку даних і проблемам безпеки. Він також підкреслює застосування автоматичних реакцій для відхилення вразливостей даних. Наприклад, хмарні системи з самовідновленням вимагають безперервних автоматичних оновлень, які можуть відхиляти налаштування безпеки, а також створювати нові помилки програмного забезпечення, створені зловмисниками. Ці системи можуть бути узгоджені з інструментами моніторингу безпеки, щоб ідентифікувати потенційні загрози та пропонувати відповіді на них [10]. На рисунку 3.2 представлено клас забезпечення безпеки процесу виконання. Вони також здатні визначати відмінності між операційними проблемами та загрозами безпеці, щоб генерувати відповідні відповіді.

```python
class SecurityModule:
    def __init__(self):
        self.security_level = "high"

    def update_security(self):
        # Simulate security update that might cause issues
        self.security_level = "medium"
        return "Security level lowered due to update"

    def monitor_threats(self):
        if self.security_level == "medium":
            return "Potential threat detected"
        return "No threats detected"

    def restore_security(self):
        if self.monitor_threats() == "Potential threat detected":
            self.security_level = "high"
        return "Security level restored to high"

# Example usage
sm = SecurityModule()
sm.update_security()
sm.monitor_threats()
sm.restore_security()
```

Рисунок 3.2 – Клас забезпечення безпеки

Міркування стійкості також пов’язані з адаптивним обслуговуванням узгодженого надання послуг і вжиття заходів щодо майбутніх збоїв. Крім того, завдяки успішній оцінці історичних даних і результатів відновлення ці системи можуть оптимізувати свої алгоритми та впроваджувати стратегії для запобігання майбутнім збоям.

Хмарні системи з самовідновленням мають можливість адаптувати стратегії відновлення на основі даних у реальному часі та вносити зміни відповідно до умов. Будучи одним із ключових аспектів стійкості, ці системи використовують метод прогнозної аналітики для оцінки потенційних проблем і створення пом’якшених дій, щоб уникнути серйозних збоїв. Завдяки впровадженню машинного навчання системи самовідновлення аналізують попередні інциденти, щоб з часом покращити свою стійку тактику.

Метою експерименту є вимірювання кількості віртуальних машин, які успішно відновилися за певний період часу за допомогою запропонованого методу. У сценарії вибрано метод реплікації. Таким чином, ми порівнюємо результати з кількістю успішно відновлених віртуальних машин за допомогою методу контрольних точок. З результатів на рисунку 3.3 видно, що запропонована стратегія досягає вищої відновлюваності, ніж схема контрольних точок, у середньому приблизно на 4%.

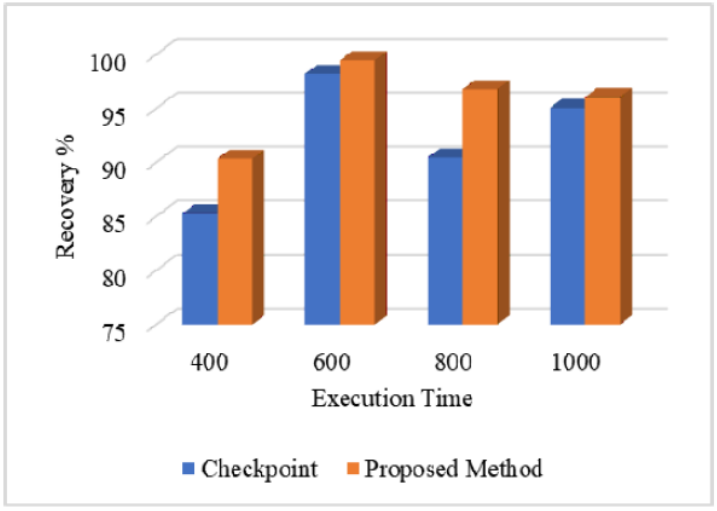

Рисунок 3.3 – Успішне відновлення віртуальних машин

Наступний набір експериментів виконується для оцінки кількості відновлень за різної частоти відмов (FR) віртуальної машини. FR означає, як часто PM або VM виходять з ладу за кількісно визначений період часу.

На рисунку 3.4 показано відновлюваність системи для FR як 0,01, 0,05, 0,1 і 0,5 для різних часів виконання. В обох наведених вище випадках запропонований підхід забезпечує вищу відновлюваність, ніж схема контрольних точок.

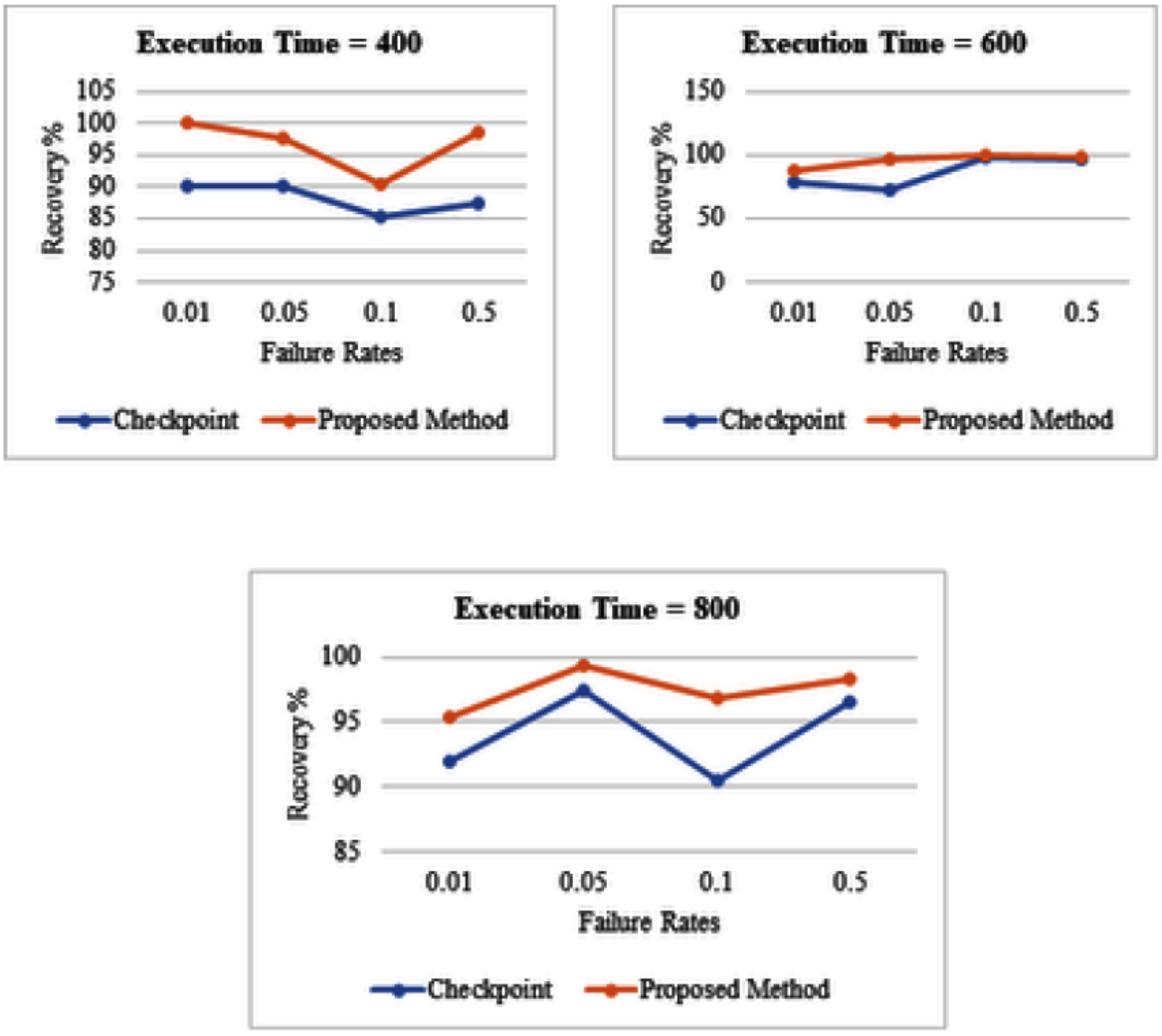

Рисунок 3.4 – Успішне відновлення проти частоти відмов

Результати запропонованої методології порівнюються з кількістю успішних відновлень віртуальної машини, виконаних за допомогою реплікації та стратегії контрольних точок у наступному експерименті. З результатів на рисунку 3.5 можна помітити, що запропонований метод

досягає вищої відновлюваності, у середньому майже 56% і 48%, ніж схеми реплікації та контрольних точок відповідно.

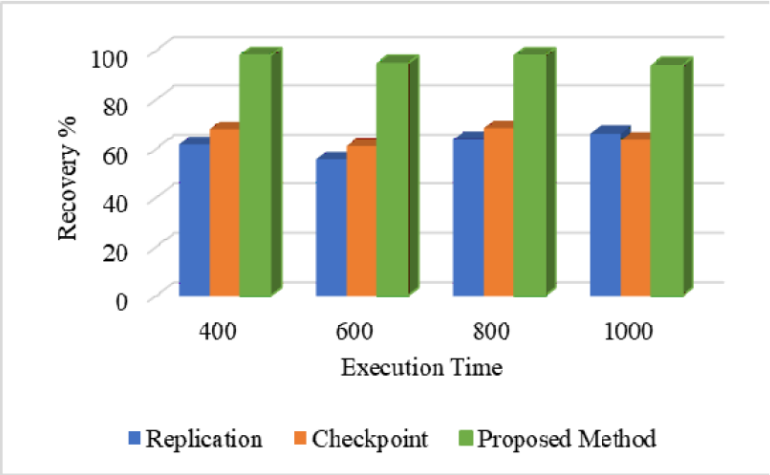

Рисунок 3.5 – Успішне відновлення віртуальних машин

Подальший набір експериментів оцінює кількість відновлень за різних інтенсивностей відмов (FR) VM. На рисунку 3.6 показано відновлюваність системи для FR як 0,01, 0,05, 0,1 і 0,5 для різних часів виконання.

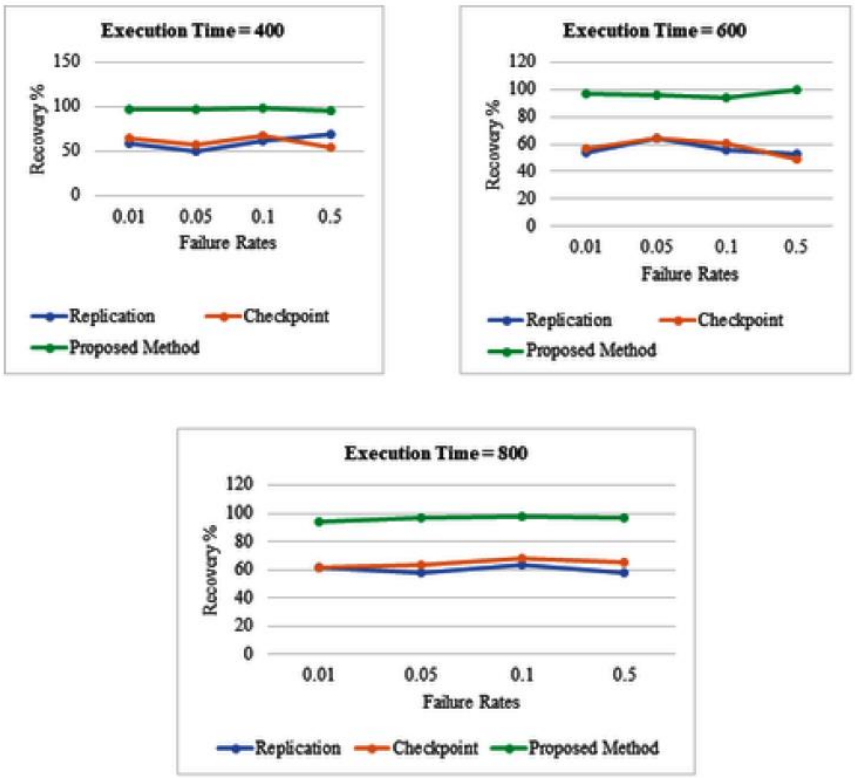

Рисунок 3.6 – Успішне відновлення проти частоти відмов

В обох наведених вище випадках запропонований підхід забезпечує вищу відновлюваність, ніж схема реплікації та контрольних точок. Те саме можна віднести до адаптивності запропонованої схеми в залежності від пріоритету завдання та даного хмарного середовища.

ВИСНОВКИ

З наведеного вище можна зробити висновок, що самовідновлювальні хмарні системи є ефективними інструментами вдосконалення в розробці програмного забезпечення, які допомагають покращити автономні служби та стійкість. Використання передових технологій, таких як машинне навчання та штучний інтелект, може допомогти цим системам виявити та вирішити потенційні проблеми, за допомогою яких загальна система може бути стійкою та мінімізувати час простою. Ці системи підтримують переваги стратегічного розгортання та оцінки продуктивності, за допомогою яких можна забезпечити операційну ефективність і постійне надання послуг, що призведе до подальшої стійкості та адаптивного хмарного середовища.

Динамічний характер середовища хмарних обчислень робить його вразливим до різних типів збоїв. Це призводить до важливості методів FT у впровадженні надійних хмарних систем. Работа представляє класифікацію та адаптивний підхід до FT у хмарі на основі нечіткої логіки. Він розглядає параметри, пов’язані з уподобаннями користувача, ризиком відмови фізичної машини та можливостями хмарної системи для визначення відповідного методу відмовостійкості, який буде використано для завдання, поданого користувачем. Тут розглядаються два типи методів FT: контрольні точки та реплікація. Ефективність запропонованого методу оцінюється за допомогою імітаційних експериментів.

Було видно, що запропонований адаптований підхід перевершує методи контрольних точок та реплікацій при окремому використанні. Впровадження адаптивного методу призводить до більшої кількості успішних відновлень, ніж інші методи; таким чином, що призводить до ефективного використання ресурсів, задоволення користувачів і економних прибутків для користувачів, а також хмарних провайдерів [15].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Asghar A., A New Paradigm for Proactive Self-Healing in Future Self-Organizing Mobile Cellular Networks. 2019. [Online]. Available: https://shareok.org/handle/11244/316814.

2. Волк М.О., Саранча С.М., Гора М.В., Ковтун Є.І., Лабазов В.Г., Полозов Д.М., Моделі ресурсів та програмних завдань для систем підтримки функціональної стійкості розподілених інформаційних систем. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 35(74) №3,Ч.1. 2024. С. 42-47. DOI: https://doi.org/10.32782/2663-5941/2024.3.1/08

3. Gadani N. The evolution of cloud architectures: from traditional virtualization to serverless and beyond. Journal of Emerging Technologies and Innovative Research. Vol. 10, Iss. 4. 2023. – P.181-190.

4. Волк М.О., Гора М.В. Модифікований метод самовідновлення розподіленого програмного забезпечення в гетерогенних комп'ютерних системах. Сучасний стан наукових досліджень та технологій в промисловості. 2024. №1 (27). С. 5-17. DOI: https://doi.org/10.30837/ITSSI.2024.27.005

5. Волк М.О., Бугрій А.М., Ковтун Є.В., Брестовицький Р.М., Соробей Б.В., Лобач Я.В. Оптимізація ресурсів у хмарних обчисленнях: гібридний підхід до автоматизації операцій та енергозбереження. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 35(74) №5 Ч.1. 2024. С. 91-96. DOI: https://doi.org/10.32782/2663-5941/2024.5.1/15

6. B. K. Ray, A. Saha, S. Khatua and S. Roy, "Proactive Fault-Tolerance Technique to Enhance Reliability of Cloud Service in Cloud Federation Environment," in IEEE Transactions on Cloud Computing, vol. 10, no. 2, pp. 957-971, 1 April-June 2022, doi: 10.1109/TCC.2020.2968522.

7. Prakash, S., Vyas, V. Analysis of Fault Tolerance Techniques in Virtual Machine Environment. ICT Analysis and Applications. 2022. – P. 121-131. DOI: 10.1007/978-981-16-5655-2_12.

8. Волк М.О., Гора М.В., Лабазов В. Г., Міщенко А.В., Барсуков А.І., Голець В.В. Журналізація стану програм для самовідновлення паралельних програмних систем. Системи управління, навігації та зв'язку, 2023, випуск 2(72), с. 80-87. DOI:https://doi.org/10.26906/SUNZ.2023.2.080

9. Y. Wu, G. Peng, H. Wang and H. Zhang. "A Two-Stage Fault Tolerance Method for Large-Scale Manufacturing Network. in IEEE Access, vol. 7. 2019.- pp.81574-81592. DOI: 10.1109/ACCESS.2019.2923660.

10. Amoon, M., El-Bahnasawy, N., Sadi, S., & Wagdi, M. (2019). On the design of reactive approach with flexible checkpoint interval to tolerate faults in cloud computing systems. Journal of Ambient Intelligence and Humanized Computing, 10(11), 4567–4577. DOI: 10.1007/s12652-018-1139-y

11. Gupta, Brij B & Agrawal, Dharma & Yamaguchi, Shingo & Sheng, Quan. Soft computing techniques for big data and cloud computing. Soft Computing. 24. 2020. – DOI: 10.1007/s00500-020-04766-2.

12. Rezaeipanah, A., Mojarad, M., & Fakhari, A. Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic. International Journal of Computers and Applications, 44(2). 2022.– P.1–9. DOI:10.1080/1206212X.2019.1709288

13. W. Dai, L. Riliskis, P. Wang, V. Vyatkin, and X. Guan, "A cloud-based decision support system for self-healing in distributed automation systems using fault tree analysis," IEEE Transactions on Industrial Informatics, vol. 14, no. 3, pp. 989-1000, 2018, doi: 10.1109/TII.2018.2791503.

14. R. Patriarca, J. Bergström, G. Di Gravio, and F. Costantino, "Resilience engineering: Current status of the research and future challenges," Safety Science, vol. 102, pp. 79-100, 2018, doi: 10.1016/j.ssci.2017.10.005.

15. Волк М.О., Бугрій А.М., Бітюкова Є.В. Галушка О.І., Брестовицький Р.М., Соробей Б.В., Розподілене моделювання гетерогенних

систем інтернету речей. Вісник Херсонського національного технічного університету. Том 2 No 1(92). 2025. - С. 45-50. DOI: https://doi.org/10.35546/kntu2078-4481.2025.1.2.6