

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»  
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ  
Кафедра інформаційної безпеки**

«На правах рукопису»

УДК 004.056

«До захисту допущено»

В.о. завідувача кафедри  
М.В.Грайворонський  
“ ” 2019 р.

## Магістерська дисертація на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: Технології самовідновлення для розподілених систем на базі штучного інтелекту

Виконав (-ла): студент (-ка) \_\_\_\_\_ курсу, групи \_\_\_\_\_  
(шифр групи)

Фокін Олександр Владиславович  
(прізвище, ім'я, по батькові)

(підпис)

Науковий керівник к.т.н., доц. Стьопочкіна Ірина Валеріївна  
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент \_\_\_\_\_

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Засвідчую, що у цій магістерській дисертації  
немає запозичень з праць інших авторів без  
відповідних посилань.

Студент \_\_\_\_\_  
(підпис)

Київ – 2019 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**  
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**  
**імені ІГОРЯ СІКОРСЬКОГО»**  
**ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**  
 Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 125 Кібербезпека («Системи і технології кібербезпеки»)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

\_\_\_\_\_ М.В.Грайворонський  
 (підпис)

«\_\_\_» \_\_\_\_\_ 2019 р.

**ЗАВДАННЯ**  
**на магістерську дисертацію студенту**

Фокін Олександр Владиславович  
 (прізвище, ім'я, по батькові)

1. Тема дисертації Технології самовідновлення для розподілених систем на базі штучного інтелекту  
 науковий керівник дисертації к.т.н., доц. Стьопочкіна Ірина Валеріївна,  
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «\_\_\_» \_\_\_\_\_ 2019 р. № \_\_\_\_\_

2. Термін подання студентом дисертації 10.12.2019 р.

3. Об'єкт дослідження \_\_\_\_\_  
 \_\_\_\_\_

4. Вихідні дані \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

5. Перелік завдань, які потрібно розробити \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

6. Орієнтовний перелік ілюстративного матеріалу \_\_\_\_\_  
 \_\_\_\_\_

7. Орієнтовний перелік публікацій \_\_\_\_\_

8. Дата видачі завдання \_\_\_\_\_

#### Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент

\_\_\_\_\_

(підпис)

\_\_\_\_\_

(ініціали, прізвище)

Науковий керівник дисертації

\_\_\_\_\_

(підпис)

\_\_\_\_\_

(ініціали, прізвище)

## РЕФЕРАТ

Обсяг роботи 102 сторінок, 39 ілюстрацій, 36 джерел літератури.

Об'єктом дослідження є технології самовідновлення компонентів системи.

Предметом дослідження є методи моніторингу та відновлення компонентів системи, та завдання забезпечення доступності, в яких вони беруть участь.

Методи дослідження – поєднання існуючих методів і технологій моніторингу та відновлення разом з власними детермінованими правилами, і штучним інтелектом.

Наукова новизна обґрунтована тим, що існуючі системи обслуговування клієнтів є небездоганними і не можуть забезпечувати нормативні вимоги безпеки в повній мірі або відновлюють систему з втратою частини функціоналу. Дана робота пропонує архітектуру розподіленої системи з властивістю самовідновлення, що має кращі показники в порівнянні з існуючими системами, завдяки поєднанню існуючих методів відновлення зі штучним інтелектом.

Результати роботи викладені у третьому розділі, в якому реалізована розподілена система обслуговування клієнтів з властивістю самовідновлення, використовуючи алгоритм машинного навчання Decision Tree, а також механізм Docker health check у поєднанні з детермінованими правилами.

Результати роботи можуть бути використані для побудови розподіленої системи, яка повинна задовольняти НД ТЗІ 2.5-004-99, а також для підвищення безпеки системи.

система самовідновлення, система моніторингу, відновлення системи, розподілена система, штучний інтелект, машинне навчання, детерміновані правила, обслуговування клієнтів

## ABSTRACT

Graduate work consists of 102 pages, 39 illustrations, 36 literature sources.

The object of the study is self-healing technologies for system components.

The subject of the study is the methods of monitoring and restoring the components of the system, and the task of ensuring the accessibility in which they participate.

Research Methods is a combination of existing monitoring, recovery methods and technologies together with our own determined rules and artificial intelligence.

Scientific novelty is justified by the fact that the existing systems of customer service are faulty and cannot fully the regulatory requirements of cyber security in full or restore the system with the loss of some functionality. This work proposes a distributed system architecture with a self-healing property that performs better than existing systems by combining current recovery methods with artificial intelligence.

The results of the work are presented in the third section, which implements a distributed customer service system is implemented with the property of self-healing using the Decision Tree machine learning algorithm, as well as the Docker health check mechanism in combination with the determined rules.

The results of the work can be used to build a distributed system that should satisfy the “HД T3I 2.5-004-99”, as well as to improve the security of the system.

self-healing system, monitoring system, system restoration, distributed system, artificial intelligence, machine learning, determination of rolling

## ЗМІСТ

Перелік умовник позначень, символів, одиниць, скорочень і термінів .....	9
Вступ .....	10
1 Аналіз типових рішень в реалізації систем самовідновлення .....	12
1.1 Теоретичні відомості про системи самовідновлення .....	12
1.2 Використання системи самовідновлення для забезпечення кібербезпеки .....	22
1.3 Аналіз існуючих рішень .....	23
1.4 Самовідновлення і моніторинг для Веб-сервісів .....	29
Висновки до розділу 1 .....	32
2 Аналіз архітектури та принципів роботи системи самовідновлення на базі штучного інтелекту .....	34
2.1 Використання AI у моніторингу та відновленні ОС .....	34
2.2 Вибір моделі нейронної мережі .....	37
2.3 Алгоритм прогнозування .....	39
2.4 Побудова детермінованих правил використовуючи алгоритм машинного навчання Decision Tree .....	40
2.5 Виявлення аномалій у часових рядах за допомогою згорткових нейронних мереж .....	41
2.6 Постановка проблеми .....	45
Висновки до розділу 2 .....	53
3 Самовідновлення компонентів розподіленої системи .....	54
3.1 Моніторинг та відновлення клієнтів на базі правил .....	54
3.2 Моніторинг та відновлення серверів обслуговування клієнтів .....	60
Висновки до розділу 3 .....	71
4 Розробка стартап-проекту .....	73

4.1 Опис ідеї проекту .....	73
4.2 Технологічний аудит ідеї проекту .....	77
Висновки до розділу 4 .....	98
Висновки .....	100
Перелік джерел посилання .....	102

## **ПЕРЕЛІК УМОВНИК ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ**

НД – нормативний документ

ТЗІ – технічний захист інформації

ОС – операційна система

РС – розподілена система

КС – компонент системи

ПЗ – програмне забезпечення

NN – neural network

ML – machine learning

TTL – time to life

SVM – support vector machine

CNN – convolutional neural network

DT – decision tree



## ВСТУП

У наш час технології самовідновлення все частіше використовуються в операційних системах. Це пов'язано з ростом вимог до сучасних систем, а саме до забезпечення критерію доступності цими системами. Ще однією причиною є те, що часто компоненти написані на мові, яка має недолік “undefined behavior”.

Оскільки системи самовідновлення ще далекі від ідеалу і вони використовуються не у всіх системах, питання покращення роботи таких систем є дуже актуальним.

**Актуальність роботи:** недосконалість методів, які використовуються у технологіях самовідновлення системи. Стрімкий ріст вимог до сучасних систем та технологій самовідновлення.

**Мета роботи:** аналіз проблем та недоліків сучасних систем самовідновлення. Запропонувати та розробити нову архітектуру моніторингу та відновлення компонентів системи.

**Завданнями даної роботи є:**

1. Проаналізувати існуючі рішення моніторингу та відновлення компонентів системи, їх переваги та недоліки.
2. Експериментальним шляхом довести недосконалість сучасних систем, які мають властивість самовідновлення.
3. Дослідити методи виявлення аномалій, які базуються на використанні штучного інтелекту.
4. Розробити та реалізувати багаторівневу розподілену систему з властивістю самовідновлення.
5. Проаналізувати отримані результати.

**Об'єкт дослідження:** технології самовідновлення компонентів системи.

**Предмет дослідження:** методи моніторингу та відновлення компонентів системи, та завдання забезпечення нормативних вимог безпеки, в яких вони беруть участь.

**Практичне значення роботи:** полягає у можливості використання розробленої архітектури та рекомендацій для створення розподіленої системи обслуговування клієнтів з властивістю самовідновлення на всіх рівнях.

**Наукова новизна одержаних результатів:** поєднання існуючих підходів моніторингу та штучного інтелекту для більш точного виявлення неправильної роботи компонентів системи, а також розробка власних алгоритмів відновлення для повернення системи в нормальний стан без втрати функціоналу. Створення багаторівневої системи моніторингу та відновлення.

**Практичне значення одержаних результатів:** результати роботи можуть бути використані організацією, яка займається обслуговуванням великої кількості клієнтів і хоче забезпечити критерій доступності 8.2 “Стійкість до відмов” і 8.4 “Відновлення після збоїв”, НД ТЗІ 2.5-004-99.

**Апробація результатів:** Результати роботи доповідалися на XVII Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні та прикладні проблеми фізики, математики та інформатики» та прийнято до друку в журналі «Theoretical and applied cybersecurity».

# 1 АНАЛІЗ ТИПОВИХ РІШЕНЬ В РЕАЛІЗАЦІЇ СИСТЕМ САМОВІДНОВЛЕННЯ

## 1.1 Теоретичні відомості про системи самовідновлення

У 2001 році, Paul Horn з компанії IBM, представив концепцію автономних обчислень. У своїй роботі він висвітлив проблеми експоненціальному росту кількості та складності систем, баз даних, взаємозв'язок між розподіленими системами та гетерогенними системами, ріст робочого навантаження та змінність навколишнього середовища. Складність систем, робить їх управління теж складним, дорогим і схильним до помилок:

- Завдання адміністрування занадто трудомісткі і вимагають висококваліфікованих ІТ спеціалістів.
- Ріст складності та налаштування складні.
- Люди, що знаходяться під тиском, здійснюють більше помилок і вимагають більше часу, щоб зрозуміти і приймати рішення.

У цьому контексті автономні обчислення виступають в якості цілісного підходу зі складністю інтеграції, управління та експлуатації обчислювальних систем. Комп'ютерна система повинна бути в змозі адаптувати себе для задоволення вимог, таких як продуктивність, відмовостійкість, надійність і безпеку. Для цього автономна система повинна:

- Включати автоматизовані механізми збору та агрегування даних для підтримки рішень ІТ-персоналу.
- Надавати поради людям, пропонуючи можливі напрямки дій.
- Бути зданими автоматично вибирати і вживати заходи з регулювання проблем.

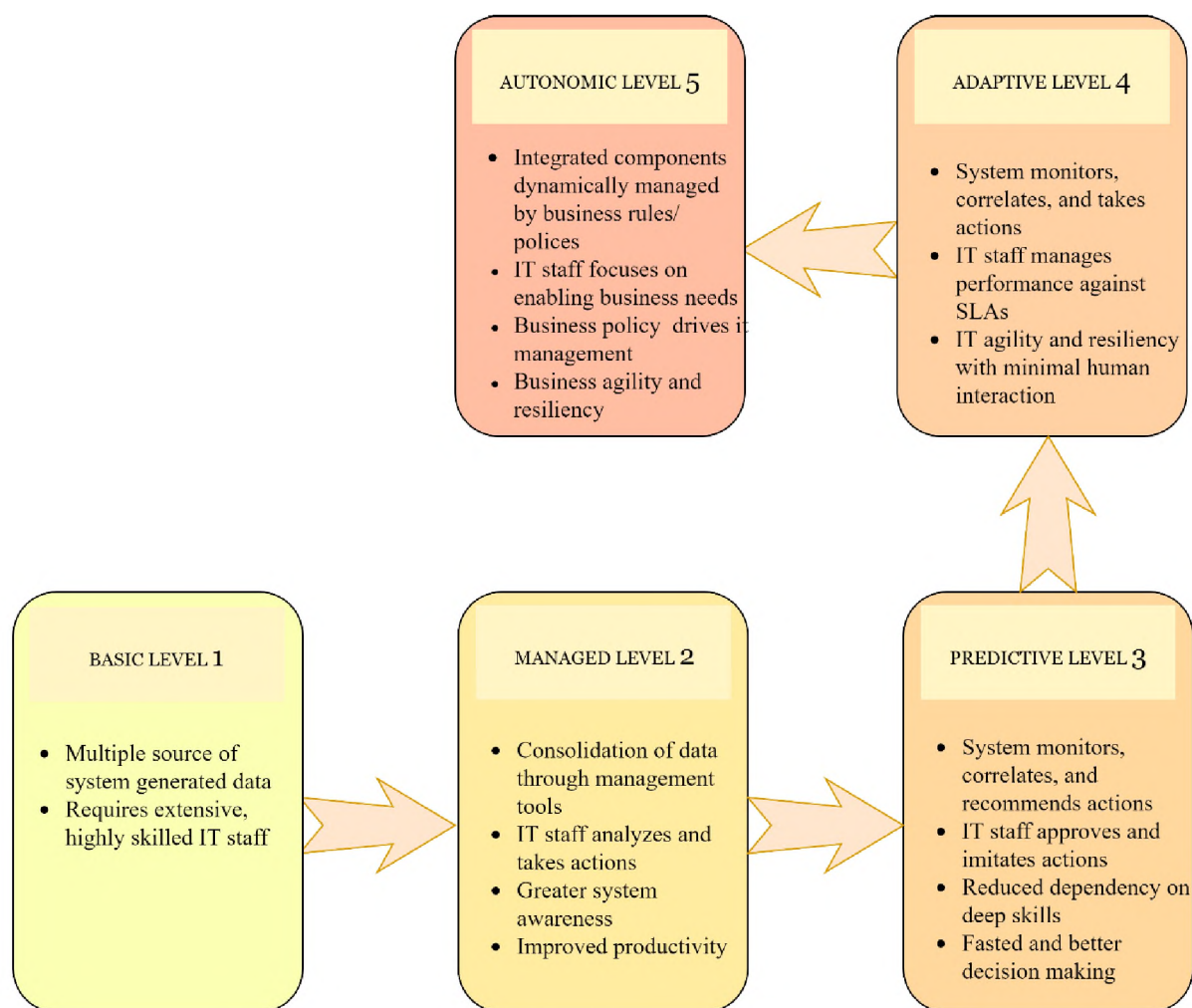


Рисунок 1.1 – Еволюція операцій автономних обчислень

Автономна система складається з одного або декількох автономних елементів. Автономний елемент складається з двох модулів. Функціональний модуль, керований елементом - який виконує необхідні послуги та функціональні можливості. Модуль управління - Autonomic Manager - який відстежує стан і контекст елемента, аналізує його поточні вимоги та надає плани адаптації для задоволення визначених вимог (наприклад: продуктивність, відмовостійкість, безпека)[1][5].

Системи самовідновлення – це системи, служби, які мають властивість виявляти, що компонент ОС працює некоректно та без втручання користувача вносять зміни для відновлення системи до нормального стану.

Можливості систем самовідновлення:

1. Моніторинг системи – полягає в перевірці правильності роботи усіх компонентів системи;
2. Відновлення системи – відновлення компоненту до попереднього стану, а якщо це неможливо припинення його роботи;
3. Сповіщення – повідомлення о проблемі розробнику компоненту.

Системи само-адаптації – це системи, які здатні або покращити свій статус, або підтримувати його в різних ситуаціях. Вони можуть дати оцінку свого поточному статусу і вони здатні вирішити, що потребують вдосконалення.

Системи само-оптимізації – це системи, які можуть знайти оптимальне рішення для задоволення конкретної мети самою системою і щоб система коригувала свій шлях відповідно до ресурсів, наданих для виконання цілей.

Системи само-моніторингу – це системи, які мають можливості та функції, необхідні для контролю за її внутрішніми функціями, а також за їх виконанням. Системи також можуть генерувати звіти, які мають зворотній зв'язок та можливості навчання або адаптації.

Само-менеджмент – це методи, за допомогою яких система управляє своїми функціями без втручання людини[2-7].

### **1.1.1 Існуючі методи моніторингу та відновлення компонентів ОС**

На сьогодні існує багато методів для моніторингу та відновлення ОС шляхом прийняття рішень при відмові одного з них. Такі технології застосовуються для забезпечення доступності операційної системи. Технології самовідновлення можуть бути реалізовані на рівні застосунків, системи або апаратному рівнях. Для моніторингу компонентів операційної системи можуть використовуватися такі підходи, а саме:

1. Час життя (TTL);
2. Перевірка доступності;

А для відновлення компонентів системи можна виділити наступні підходи, а саме:

1. Реактивне відновлення;
2. Профілактичне відновлення[3];

### **1.1.2 Підхід моніторингу на основі часу життя (TTL)**

Перевірки часу життя (TTL) припускають, що служба або додаток будуть періодично підтверджувати свою працездатність. Система, яка приймає сигнали TTL, відстежує останній повідомлений стан для даного TTL. Якщо цей стан не оновлюється протягом попередньо визначеного періоду, система моніторингу передбачає, що служба не виконана і її необхідно відновити до початкового стану. Якщо процес, в якому працює служба, завершиться невдало, він не зможе відправити запит, закінчиться TTL і будуть виконані заходи реагування. Основна проблема TTL – це те, що програми та сервіси повинні бути прив’язані до системи моніторингу. Впровадження TTL було б одним з анти-патернів мікро-сервісів, бо ми намагаємося спроектувати їх так, щоб вони були максимально автономними. Більш того, мікро-сервіси повинні мати чітку функцію і єдину мету. Реалізація запитів TTL всередині них збільшить додаткову функціональність і ускладнить розробку[3].

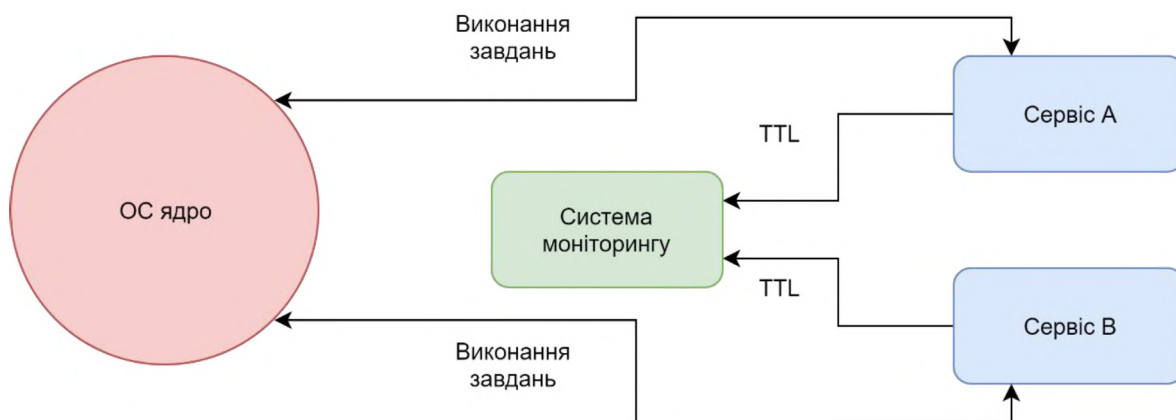


Рисунок 1.2 – Схема роботи системи моніторингу на основі TTL

### 1.1.3 Підхід моніторингу на основі перевірки доступності

Підхід на основі пінгування полягає в тому, щоб перевіряти зовнішній стан програми або служби. Система моніторингу повинна періодично перевіряти кожну послугу і якщо відповідь не отримана або зміст відповіді не є коректним, виконати заходи відновлення. Пінгування протилежне до TTL і по можливості найбільш прийнятний шлях перевірки стану окремих частин системи[3].

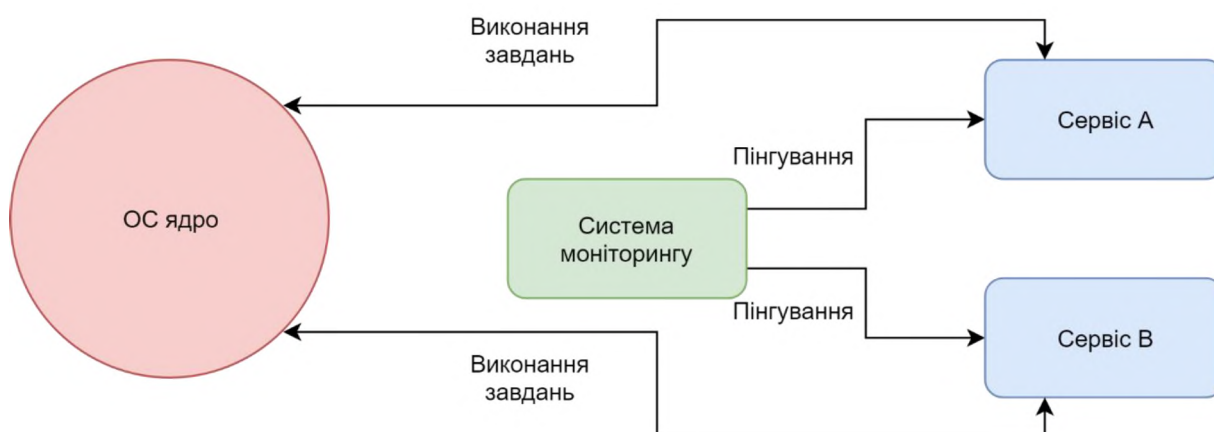


Рисунок 1.3 – Схема роботи системи моніторингу на основі пінгування

### 1.1.4 Підхід відновлення системи на основі реактивного самовідновлення

Підхід на основі реактивного самовідновлення полягає в тому, щоб після того як засоби моніторингу зафіксували неправильну поведінку компоненту системи або зупинення його роботи через непередбачену помилку, система відновлює свій стан до заданого. Доки у нас є всі перевірки, а також дії, які повинні бути виконані у разі збою, ми зводимо час простою системи до мінімуму[3].

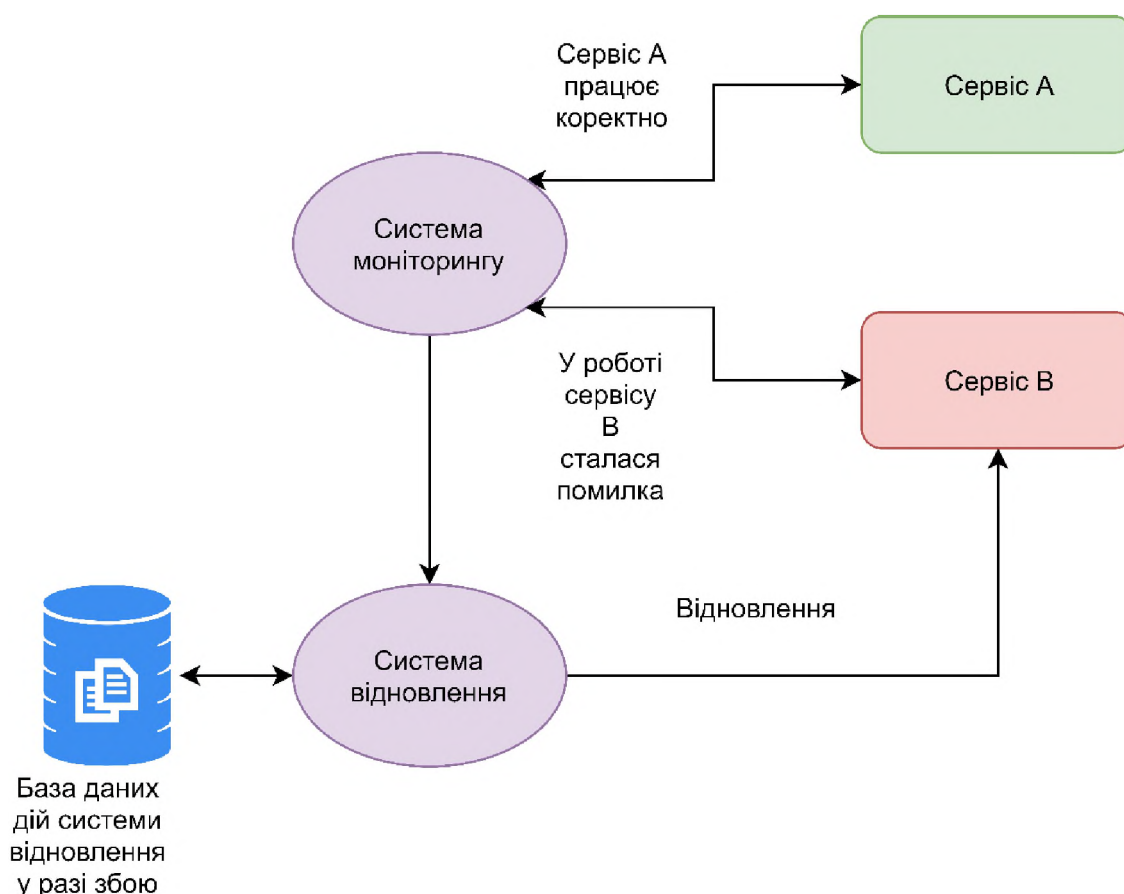


Рисунок 1.4 – Приклад роботи системи на основі реактивного самовідновлення



### **1.1.5 Підхід відновлення системи на основі профілактичного самовідновлення**

Ідея профілактичного лікування полягає в тому, щоб передбачити проблеми, які можуть виникнути в майбутньому і діяти таким чином, щоб уникнути цих проблем. Простий, але менш надійний спосіб прогнозування майбутнього полягає в тому, щоб засновувати припущення на даних в реальному часі. При такому підході аналізується час відгуку системи, завантаженість процесора, пам'яті та інше. Наприклад, ми могли б помітити, що протягом останньої години використання пам'яті неухильно зростало і досягло критичного рівня, скажімо, 90%. Це було б чіткою ознакою того, що послуга, що викликає таке збільшення, повинна масштабуватися. Система також може враховувати більш тривалий період часу і робити висновок про те, що щопонеділка відбувається раптове збільшення трафіку, і завчасно масштабувати послуги, щоб запобігти тривалій відповіді[3].

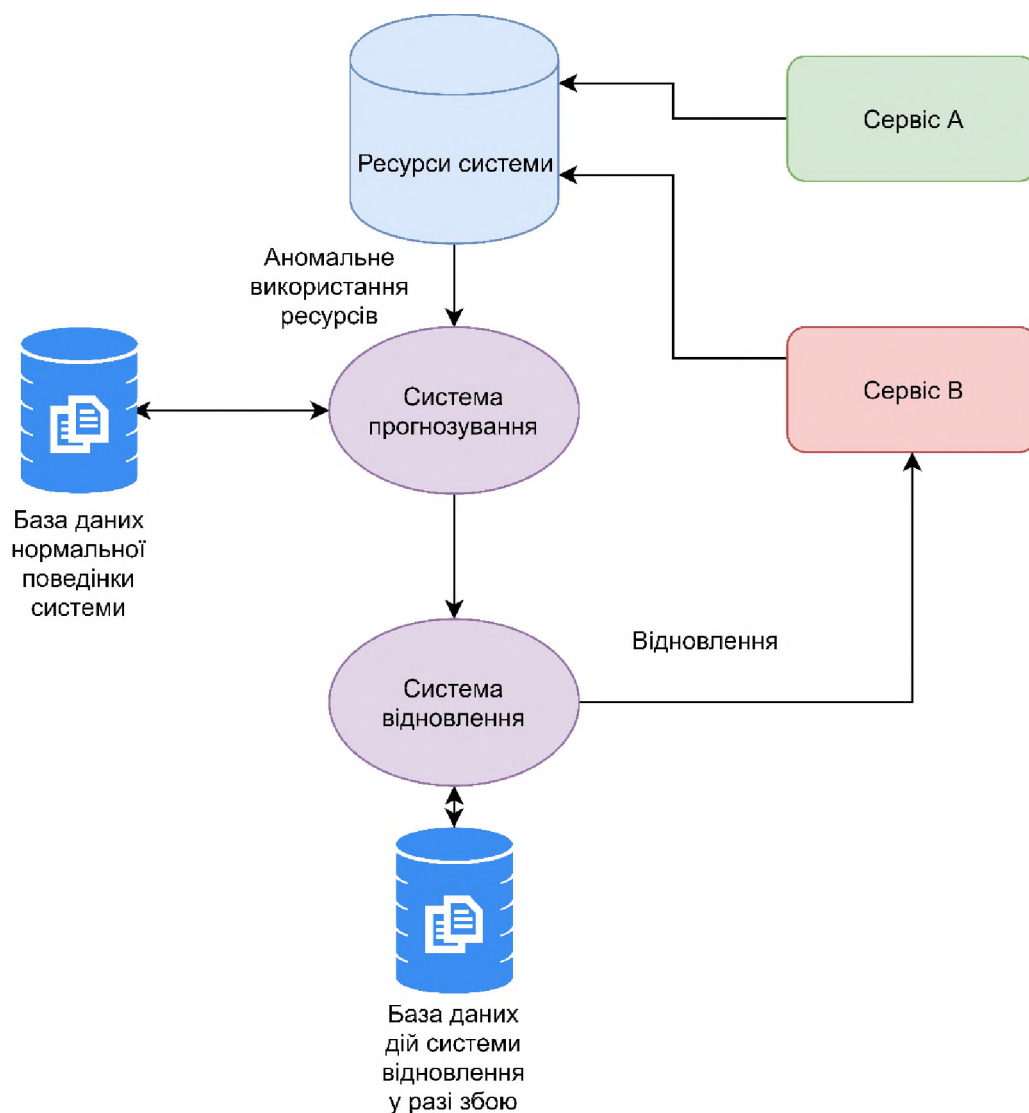


Рисунок 1.5 – Приклад роботи системи на основі профілактичного самовідновлення

### 1.1.6 Життєвий цикл систем самовідновлення

Життєвий цикл систем самолікування можна розділити на 5 фаз:

1. Моніторинг
2. Аналіз
3. Діагностика
4. Відновлення
5. Збереження подій

На етапі моніторингу система збирає інформацію з усіх модулів. Після цього інформація вводиться у використання для аналізу завдання.

Наступний етап – це аналіз. Аналіз завдання допомагає визначити дії, які потрібно виконати, порівнявши інформацію про стан із вимогами системи.

Діагностика – це процес фіксації всіх несправностей, помилок або змін у системі, використовуючи порівняння чи інших методів.

Етап відновлення – це коли система повертається до нормального стану, шляхом усунення несправностей.

Збереження подій – ця фаза є важливою, оскільки містить усі відповідні знання, спожиті та отримані попередніми чотирма завданнями[1-7].

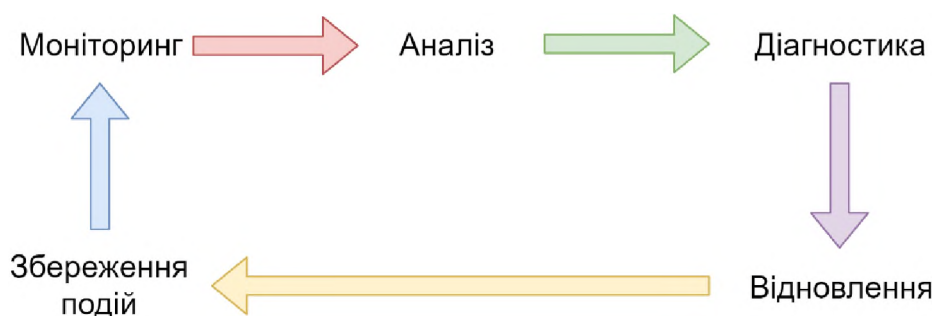


Рисунок 1.6 – Життєвий цикл системи самовідновлення

### 1.1.7 Дизайн для систем самовідновлення

Компанія Microsoft розробила рекомендації для додатків, які мають властивість самовідновлюватися. Рекомендації складають шість пунктів, які можна використовувати для побудови системи самовідновлення.

1. Повторювати невдалі операції. Тимчасові збої можуть виникнути через миттєву втрату підключення до мережі, перерване з'єднання з базою даних або підчас коли служба зайнята. Так для багатьох сервісів Azure клієнт SDK реалізує автоматичне повторення спроб.

2. Ізоляція критичних ресурсів. Збій в одній підсистемі іноді може каскадно впливати на інші системи. Це може статися, якщо збій спричиняє не своєчасне звільнення деяких ресурсів, що призводить до їх вичерпування. Щоб цього уникнути, треба розділяти систему на ізольовані групи, щоб помилка в одному розділі не призвела до краху всієї системи.

3. Вирівнювати навантаження. Раптові сплески навантаження можуть впливати на продуктивність системи. Щоб цього уникнути можна використовувати шаблон вирівнювання навантаження на основі черги, щоб робочі елементи працювали асинхронно. Черга діє як буфер, який згладжує піки навантаження.

4. Деградуйте витончено. Іноді ви не можете вирішити проблему, але ви можете забезпечити зменшений функціонал, який все ще корисний. Розгляньте додаток, що показує каталог книг. Якщо програма не може отримати мініатюрне зображення для обкладинки, воно може відображати пусте зображення. Цілі підсистеми можуть бути некритичними для програми. Наприклад, на веб-сайті електронної комерції показ рекомендацій щодо товару, ймовірно, менш критичний, ніж обробка замовлень.

5. Тест на ін'єкцію несправності. Занадто часто шлях до успіху добре перевірений, але не шлях невдачі. Система може працювати в нормальному режимі протягом тривалого часу, перш ніж станеться її відмова. Треба використовувати ін'єкцію несправностей для перевірки стійкості системи до відмов, або запускаючи фактичні збої, або імітуючи їх.

6. Використовувати інженерію хаосу. Техніка хаосу розширює поняття введення несправності, шляхом випадкового введення несправностей або аномальних умов у виробничі екземпляри[8].

## **1.2 Використання системи самовідновлення для забезпечення кібербезпеки**

Системи самовідновлення допомагають забезпечити один із критеріїв оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу, а саме нормативного документу НД ТЗІ 2.5-004-99, критерію доступності 8.2 “стійкість до відмов”. Згідно цього критерію система повинна мати політику стійкості до відмов та визначити компоненти, до яких ця політика відноситься і типи відмов після яких система в змозі продовжувати функціонування. Системи самовідновлення можуть захищати компоненти системи від відмов, які можуть призводити до недоступності послуг, а також спроможні повідомити адміністратора (розробника), про відмову будь-якого захищеного компонента.

Системи з властивістю самовідновлення забезпечують (ДС-2 або ДС-3). Тобто такі системи мають стійкість з погіршенням характеристик обслуговування або без погіршення. Система самовідновлення може відновити модуль, або якщо це тимчасово неможливо, відключити його.

Також системи самовідновлення забезпечують послугу “відновлення після збоїв”. Використовуючи технології самовідновлення можна повернути комп'ютерну систему у відомий захищений стан після відмови. За НД ТЗІ 2.5-004-99 технології самовідновлення виконують автоматизоване відновлення (ДВ-2). Система після відмови має здатність визначити можливість повернення системи до нормального стану в автоматичному режимі (без втручання користувача) і якщо це можливо, повернути КС до нормального функціонування.

Обидві послуги “стійкість до відмов” та “відновлення після збоїв” потребують реалізацію рівня НО-1, тобто розподіл обов'язків, в системі повинні бути визначені ролі користувача та адміністратора[9].

Технології самовідновлення можуть забезпечити відновлення системи після атаки шкідливого програмного забезпечення, системних помилок при оновленні, а також підтримувати модулі, які безпосередньо відповідають за захист системи, файрволи, модулі антивірусного захисту, модулі захисту персональних даних та інше. Щодо розподілених систем, технології самовідновлення також можуть захищати системи від Dos атак. Така атака в системі обслуговування великої кількості клієнтів може бути виконана не тільки зловмисником, а і самими клієнтами або розробником ПО.

### **1.3 Аналіз існуючих рішень**

На сьогоднішній день не важко знайти систему, яка б не мала модуль самовідновлення. Оскільки чим система більша, складніша та має змогу підключення нових модулів та систем, потрібність модулю самовідновлення.

#### **1.3.1 Самовідновлення в системі Linux**

Розпочнемо з популярної операційної системи Linux, а саме з Ubuntu 16.04.3 LTS. Для того щоб, показати наявність та роботи системи самовідновлення, будемо імітувати виникнення несправностей в роботі одного із процесів системи, а саме memory leak. Також на систему встановлено Jupyter Notebook, python та додаткові frameworks. Після чого розпочнемо навчання нейронної мережі, задавши їй параметри великого розміру, а саме batch size[10].

```

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.3 LTS"
NAME="Ubuntu"
VERSION="16.04.3 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.3 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial

```

Рисунок 1.7 – Інформація о системі

```

1 [ 2.4%] 9 [ 0.0%] 17 [
2 [ 0.0%] 10 [ 0.0%] 18 [
3 [ 3.1%] 11 [ 0.0%] 19 [
4 [ 0.0%] 12 [ 0.0%] 20 [
5 [ 0.0%] 13 [ 0.0%] 21 [
6 [ 0.0%] 14 [ 0.0%] 22 [
7 [ 0.0%] 15 [ 0.0%] 23 [
8 [ 0.0%] 16 [ 0.0%] 24 [
Mem[ 1.17G/62.7G] Tasks: 2934, 1488 thr; 1 running
Swp[ 11.8G/14.9G] Load average: 0.13 0.06 0.02
Up-time: 332 days(1), 01:39:49

```

Рисунок 1.8 – Стан системи до початку експерименту

Для експерименту була обрана модель нейронної мережі SeqtoSeq, оскільки при тренуванні вона використовує великий розмір оперативної пам'яті. Як видно на рисунку 1.8, до запуску моделі всі ресурси системи, які необхідні для навчання були вільні.

```

BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 256
steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
embedding_dim = 256
units = 1024
vocab_inp_size = len(inp_lang.word_index)+1
vocab_tar_size = len(targ_lang.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train, ta
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

```

Рисунок 1.9 – Параметри нейронної мережі

Після запуску навчання нейронної мережі, Jupyter почав активно використовувати оперативну пам'ять серверу, доки процес не використав майже всі його ресурси.

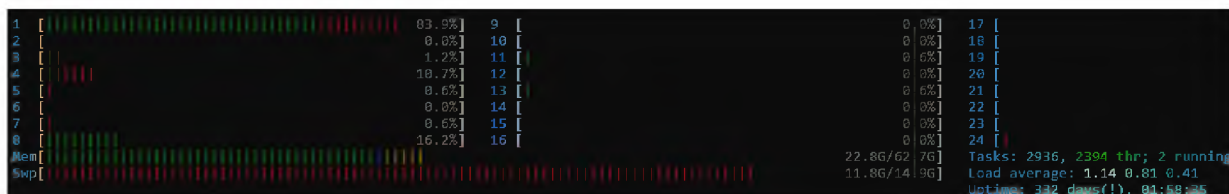


Рисунок 1.10 – Використання Jupyter ресурсів системи

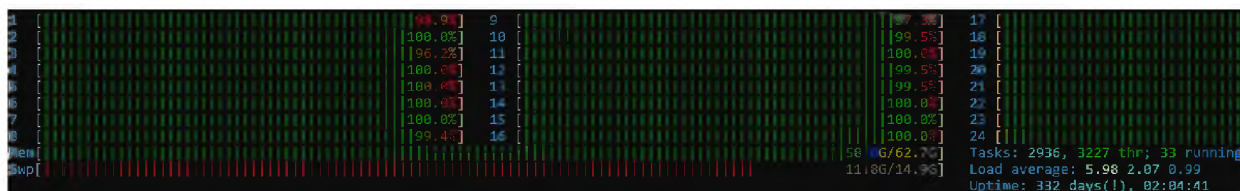


Рисунок 1.11 – Критична точка системи перед початком відновлення

Як тільки система зафіксувала, що процес зайняв практично всі ресурси системи, для того щоб продовжити нормально функціонувати, терміново завершила процес звільнивши ресурси.

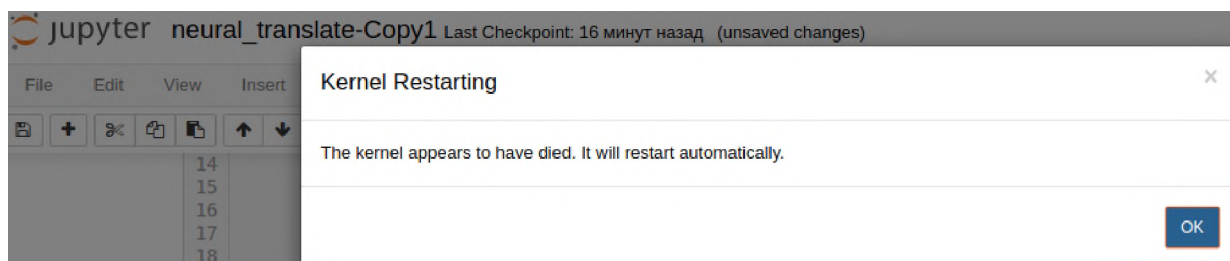


Рисунок 1.12 – Linux завершив процес Jupyter



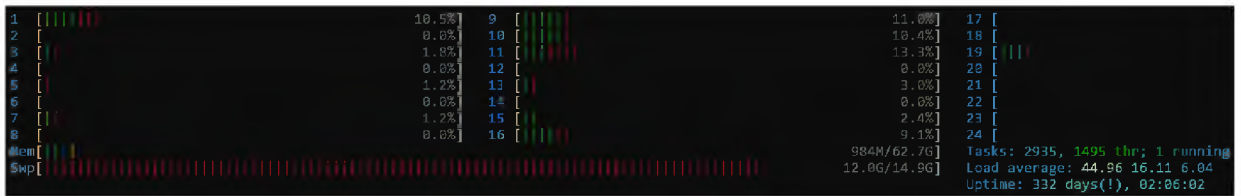


Рисунок 1.13 – Система Linux після відновлення

Як можна бачити система змогла запобігти повній відмові в обслуговуванні шляхом пожертвування функціоналом, повернувши себе до нормального стану.

### 1.3.1 Самовідновлення в системі Android

Система самовідновлення в одній з найпопулярніших оперативних систем Android працює за іншим алгоритмом для своїх системних сервісів ніж ОС Linux. Різниця в тому, що при виникненні схожої ситуації, система Android намагається повернути сервіс до початкового стану, після чого викликає його знову. Для експерименту було використано Android версії 6, після чого була внесена штучна помилка в один із її компонентів, а саме модуль Bluetooth[11].

```
done;;
// Be nice and restore the old value of that byte
*(data) = previous_byte;

// Remove the signal byte from our transmitted length, if it was actually written
if (transmitted_length > 0)
    --transmitted_length;
uint8_t * print_data;
uint8_t * print_data2;
print_data = (uint8_t*)malloc(1000);
print_data = data;
for (int i = 0; i < 100000; i++)
{
    print_data2 = (uint8_t*)malloc(1000000);
    LOG_INFO("%s LogbyAlexprintmass %02x", __func__, print_data[i] );
}
return transmitted_length;
```

Рисунок 1.14 – Код доданий в базову прошивку Android

Було з'ясовано, що система дійсно спочатку спробувала відновити сервіс до початкового стану, після чого почала викликати його знову і знову намагаючись отримати від нього відповідь, що в свою чергу привело до повного відказу ОС. Це пов'язано з тим, що в системі не було реалізовано обмежень на спроби викликати сервіс, а також відсутність такого механізму як в системі Ubuntu, яка б просто завершила процеси, які зайняли практично всі ресурси. Помилка привела до автоматичного перезавантаження ОС.

### 1.3.2 Самовідновлення у нереляційних базах даних

Прикладом реалізації системи самовідновлення у нереляційних базах даних, може стати MongoDB з використанням AWS сервісів.

MongoDB — документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

MongoDB дозволяє створити кластер, що характеризується високою надійністю за допомогою ReplicaSets та Sharding:

- ReplicaSets – це набір однакових вузлів, на які копіюються введені в базу даних документи JSON.
- Шардінг – дозволяє записувати кожен набір документів JSON (Колекції) на інший сервер. Це забезпечує кращу пропускну здатність як для читання, так і для письма.

Використовуючи MongoDB та AWS сервіси можна реалізувати базу даних з самовідновленням, а саме:

- Autoscaling group – це послуга AWS для управління та масштабування груп екземплярів. Можна визначити тип екземпляра та кількість застосованих екземплярів. AWS запусить їх, відстежить їх стан та, якщо необхідно, скасує їх, а також запусить нові.
- Route 53 – AWS DNS service
- Lambda Functions - виконує програмний код за конкретних умов та відповідає за автоматичне виділення необхідних ресурсів.
- SQS Queues - високонадійні черги повідомлень.
- AWS CodeDeploy - служба, керована AWS, для довільного розгортання коду через встановлений на машині агент, який інтегрується з групами автоматичного масштабування. Розгортання виконується при запуску нового примірника.

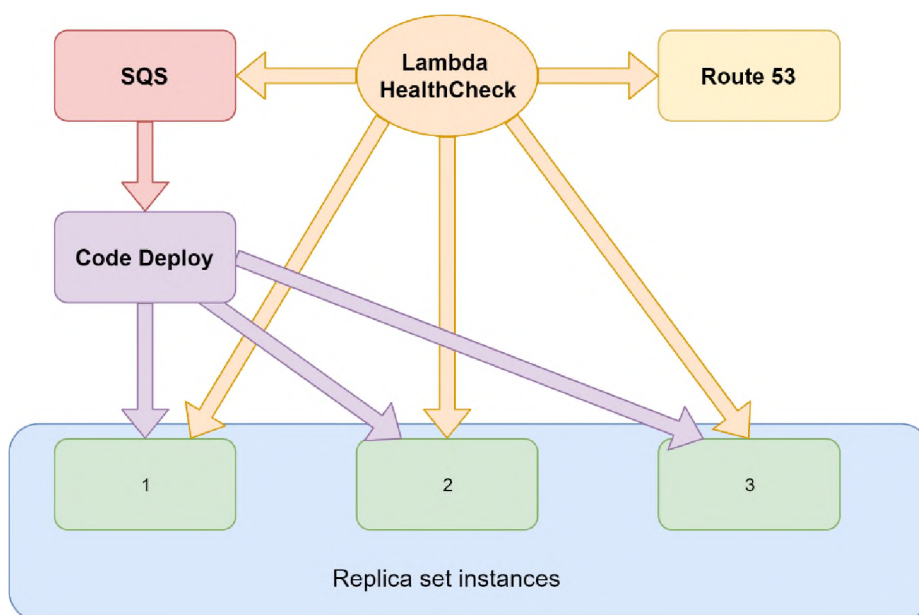


Рисунок 1.15 – Модель самовідновлення у MongoDB з використанням AWS сервісів

Для того, щоб система самовідновлення працювала, машини реєструються в групу автоматичного несення шкали за допомогою перевірки стану EC2.

Таким чином, у разі відмови машини, AWS створить новий екземпляр з AMI, визначеного за шаблоном групи автоматичного масштабування. Далі створена лямбда функція, яка діє як система моніторингу для машин ReplicaSet, контролює їх стан за допомогою відправки до них команди replSetGetStatus.

Якщо вузол не відповідає на запит, лямбда записує повідомлення у чергу SQS, яке містить ім'я DNS, машини яка не відповідає на запити, а також відокремлює її від групи автоматичного масштабування. Після чого EC2 створює новий екземпляр, а CodeDeploy робить усі необхідні налаштування[12].

#### **1.4 Самовідновлення і моніторинг для Веб-сервісів**

Будь-яка система, яка працює бездоганно, може бути зупинена великою кількістю несправностей. Залежно від завдань, які виконує система ці несправності можуть мати різноманітні форми. Сукупний ефект цих несправностей – зменшення продуктивності системи і, отже, падіння її ефективності. Система вже не функціонує як раніше до виникнення несправностей. Вони можуть виникати на різних рівнях в архітектурі системи, крім того, деякі несправності можуть викликати виникнення інших несправностей. Таким чином, несправність у системі - це явище, яке призводить до певного відхилення від очікуваної поведінки системи. Відмова одного із сервісів через програмні помилки або апаратні, а також проблеми с мережею є основною загрозою для розподілених систем. На сьогоднішній день, існує багато підходів для моніторингу та відновлення сервісів.

Несправності, які виникають у розподілених системах можна поділити на три основних типи:

1. **Перехідні несправності:** ці помилки виникають і потім зникають. Наприклад, повідомлення відправлене по мережі не досягло пункту призначення через роз'єднання, але проблема зникає при повторній спробі.

2. **Переривчасті несправності:** Постійні несправності, які повторюється. Це найбільш дратуючі недоліки і виникають в основному через несправності компонентів або неправильної між компонентної роботи.

3. **Постійні помилки:** Цей тип відмови є стійким і він буде існувати до тих пір, поки несправний системний компонент невідновлений або повністю не замінений.

Системи самовідновлення можуть знаходити помилки у розподілених системах та відновлювати їх роботи, повертаючи ефективність такої системи до нормальної. Звичайні способи до усунення помилок включали б ведення журналу, моніторинг на основі моделей та підходи на рівні компонентів, а за відновлення нормальної роботи, відповідала б людина на базі отриманих даних. Ці підходи підтримують деякі частини процесу самолікування, але не весь процес, що включає моніторинг, фільтрацію, аналіз, діагностику, прийняття рішення та відновлення.

Одним з основних підходів самовідновлення для веб сервісів є Rule-Based. Головною перевагою такого підходу є те, що можна швидко оновлювати алгоритм роботи системи.

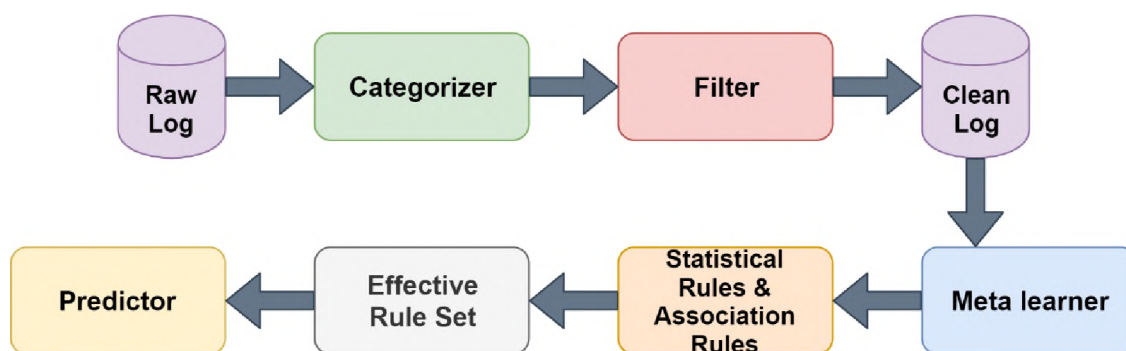


Рисунок 1.16 – Схема прогнозування відмов

Журнали RAS не можна використовувати напряду, через нерівномірність подання однотипних повідомлень про помилки. Data Scrubber обробляє і робить очищення даних таким чином, щоб зробити їх корисними для обробки та аналізу. Цей крок також допомагає зменшити обсяг статистичних даних для їх зберігання. По завершенню роботи Data Scrubber надає список унікальних подій для прогнозування відмов.

Categorizer розбиває дані, які до нього надходять по категоріям. Він відображає проблему на бінарну модель, де конкретна подія або відбувається, або ні. Також проводить сортування за важливістю. Це допомагає розпізнавати помилки у системі ефективніше.

Розподілені системи мають тенденцію писати повідомлення про одну і ту ж саму подію декілька разів. Для того щоб позбавитись дублікатів, у модель було добавлено Filter, який залишає тільки унікальні події.

Тепер відфільтровані та структуровані дані можуть бути оброблені Prediction Engine для аналізу, щоб виявити помилки у роботі системи. Для прогнозування використовується Association Rule Based Learning. Очищенні дані подаються до “Weka”, яка після обробки даних повертає правила. Після чого дані аналізуються за алгоритмом “Apriori”. Якщо модуль фіксує, що нода працює некоректно, запускається процес відновлення і система повертається до нормального стану.

Також можна використовувати аналітичні моделі ефективності, для прогнозування реакції системи на різні конфігурації та зміни. Вони складаються з набору обчислювальних алгоритмів, які використовують вимірювані параметри робочого навантаження для обчислення експлуатаційних характеристик системи, використовуючи різні аналітичні моделі можна оцінити час відповіді.

Queuing Network модель – це один із способів створення аналітичної моделі продуктивності системи. QN являє собою сукупність взаємопов'язаних черг, які включають ресурс, що надає послугу, так і лінію очікування для

доступу до цього ресурсу. QN використовується для моделювання системи та оцінки впливу на проектні рішення. При створенні моделей QN використовуються два типи параметрів: інтенсивність навантаження та обслуговування. Моделі QN можна використовувати для представлення багатокласових систем. Таким чином таку модель можна використовувати як систему моніторингу[13-16],[21].

### **Висновки до розділу 1**

В даній роботі були розглянуті підходи моніторингу на основі перевірки доступності та часу життя TTL. Кожен з цих підходів має свої переваги та недоліки. Так наприклад, підхід на основі моніторингу TTL ускладнює розробку сервісів і це пов'язано з тим, що необхідно реалізувати TTL всередині самих компонентів системи. Цю проблему вирішує підхід на базі перевірки доступності, але його не завжди можна реалізувати у зв'язку з архітектурними особливостями деяких компонентів.

Проаналізовано забезпечення системами самовідновлення нормативних вимог безпеки, таких як, НД ТЗІ 2.5-004-99. Розглянуті функції систем самовідновлення, які відповідають за забезпечення кібербезпеки.

Також було розглянуто існуючі рішення, такі як система самовідновлення у операційних системах Android і було доведено її недосконалість та розглянуто дистрибутив операційної системи Linux Ubuntu. Робота також торкнулася систем самовідновлення для розподілених систем, а саме архітектуру самовідновлення для MongoDB на базі AWS сервісів і нові підходи моніторингу системи для виявлення неправильної поведінки компоненту.

Аналізуючи усі підходи, можна зробити висновок о необхідності розробки системи, яка б була гнучка, тобто могла аналізувати з яким типом компоненту вона працює та який оптимальний підхід обрати для моніторингу та

відновлення у разі збоїв. Система повинна відновлюватися без втрати частини свого функціоналу, тобто виконувати автоматизоване відновлення (ДВ-2).



## **2 АНАЛІЗ АРХІТЕКТУРИ ТА ПРИНЦИПІВ РОБОТИ СИСТЕМИ САМОВІДНОВЛЕННЯ НА БАЗІ ШТУЧНОГО ІНТЕЛЕКТУ**

### **2.1 Використання AI у моніторингу та відновленні ОС**

Використання AI для моніторингу та відновлення системи дуже важливо. На сьогодні існує багато різноманітних сервісів, які відрізняються не тільки задачами, які вони виконують, але і своєю архітектурою. Неправильно застосовувати одні та ті ж методи та параметри для різних сервісів ОС. Система моніторингу повинна будувати гнучкою та аналізуючи усі дані підбирати оптимальні параметри роботи усіх своїх компонентів. Саме використання AI може розв'язувати цю проблему контролюючи роботу системи, будуючи портрети сервісів на основі даних з модулів моніторингу та зберігання їх у базі даних, фіксувати неправильну роботу та приймати рішення з відновлення ОС.

Розглянемо приклад використання AI з об'єднанням підходів для моніторингу TTL та пінгування. У нашому випадку навчання має відбуватися лише з одним класом даних. Система самолікування отримує ці дані в тренувальному режимі, коли ОС знаходиться в нормальному стані. Перевірка стану компонентів системи повинна проводитися за допомогою TTL та методу перевірки доступності. Після тренування AI, TTL та модуль перевірки доступності можна відключити для підвищення продуктивності. Якщо модуль AI виявляє аномалію, то TTL та модуль перевірки доступності можуть використовуватися як додаткові інструменти для перевірки стану компонентів системи.

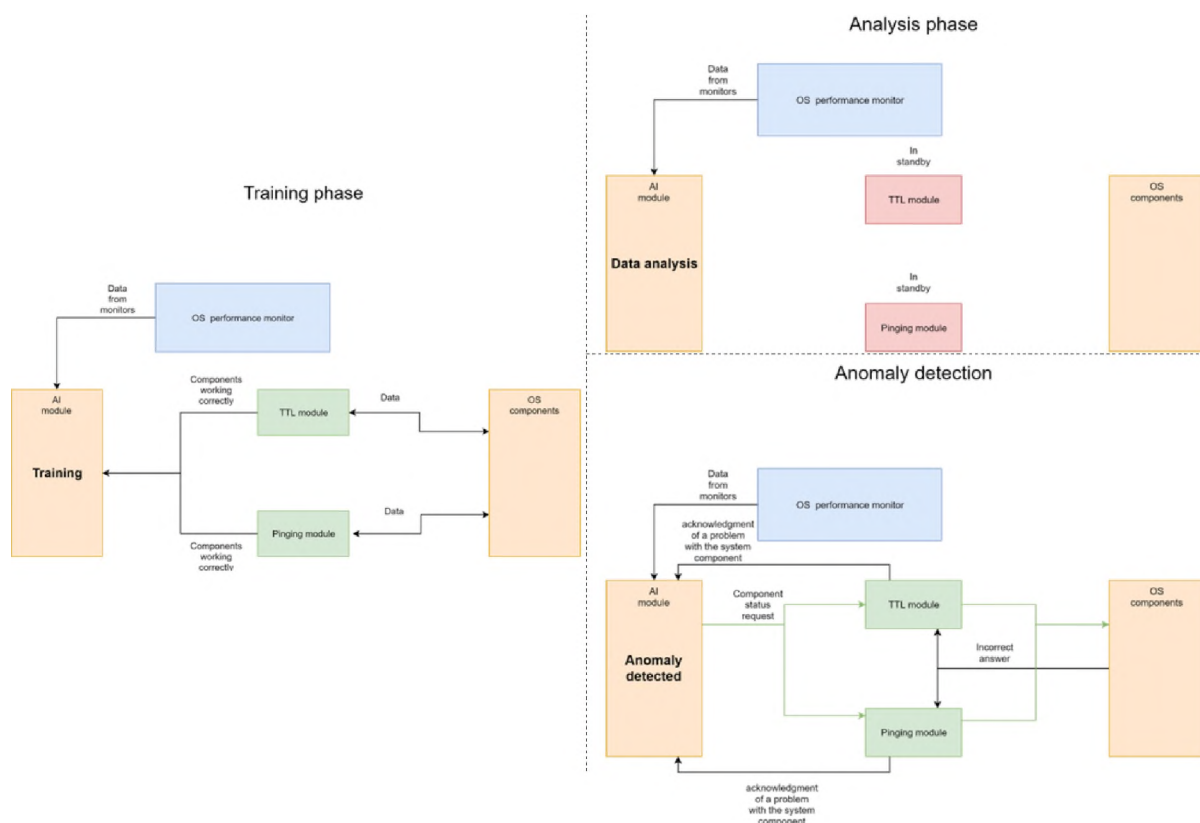


Рисунок 2.1 – Модель у режимі тренування, аналізу, та знаходженні аномалій

На базі усіх переваг та недоліків різних підходів можна побудувати нову модель системи моніторингу та відновлення ОС.

Після того, як у системі запускається новий компонент, запропонована модель працює по наступному алгоритму:

1. Модуль “Resource monitorin” повідомляє модулю “General function”, що у системі почав працювати новий компонент.

2. “General function” відправляє запит до компонента з метою отримати доступні підходи для моніторингу і в залежності від відповіді підключає TTL модуль, або модуль pinging. Модуль “Resource monitoring” підключається завжди, незалежно від відповіді компоненту системи.

3. Після підключення модулів моніторингу, “General function” відправляє данні до модуля AI, який у свою чергу повертає параметри моніторингу та починає обробляти данні з систем моніторингу. У разі фіксування неправильної

поведінки компоненту, він відправляє команду до “General function” на відновлення системи.

4. Якщо відновлення компоненту пройшло не вдало, система повинна прийняти рішення, щодо завершення процесу. Якщо компонент є критичним для системи і його завершення приведе до її відмови, параметр “Ignore” повинен дорівнювати “True”.

5. “General function” починає обробляти команду, починаючи з параметр “Ignore”. Далі модуль перевіряє значення параметру “Critical” і якщо він буде дорівнювати значенню “true”, система негайно виконає процедуру “kill process”. У випадку коли параметр “Critical” дорівнює значенню “false” модуль відправляє повідомлення користувачу системи для прийняття рішення.

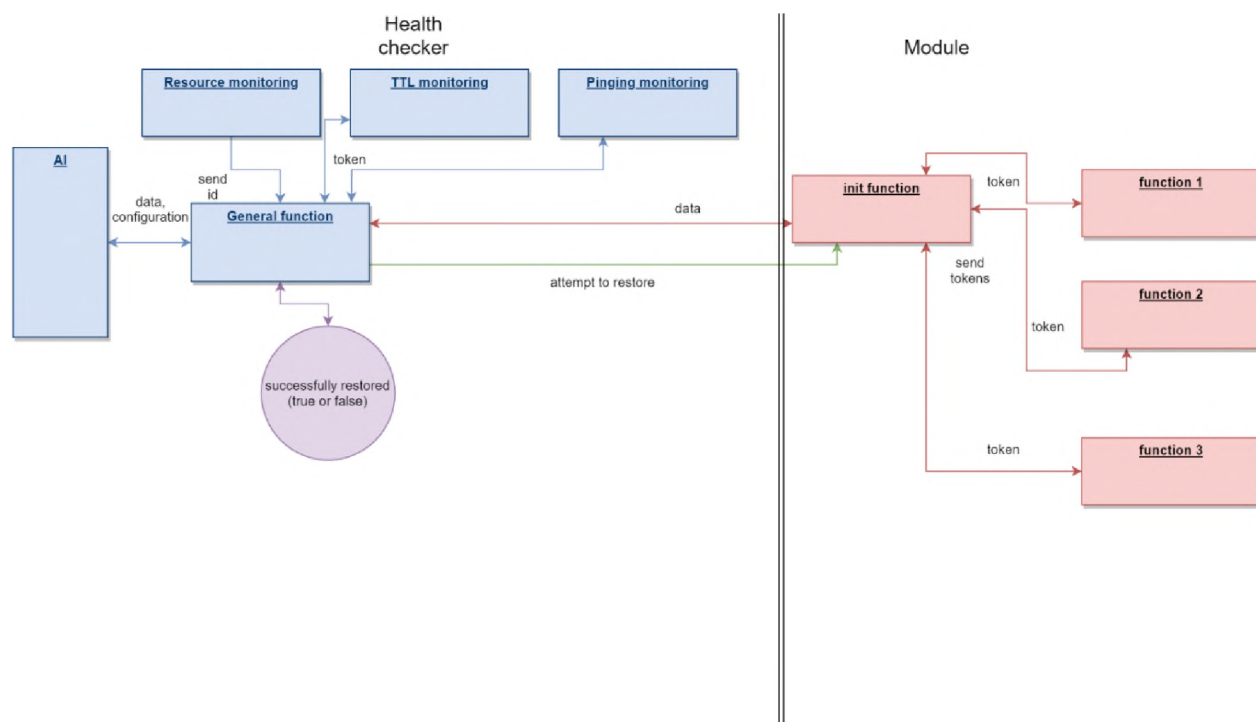


Рисунок 2.2 – Схема роботи моделі при відновленні компоненту

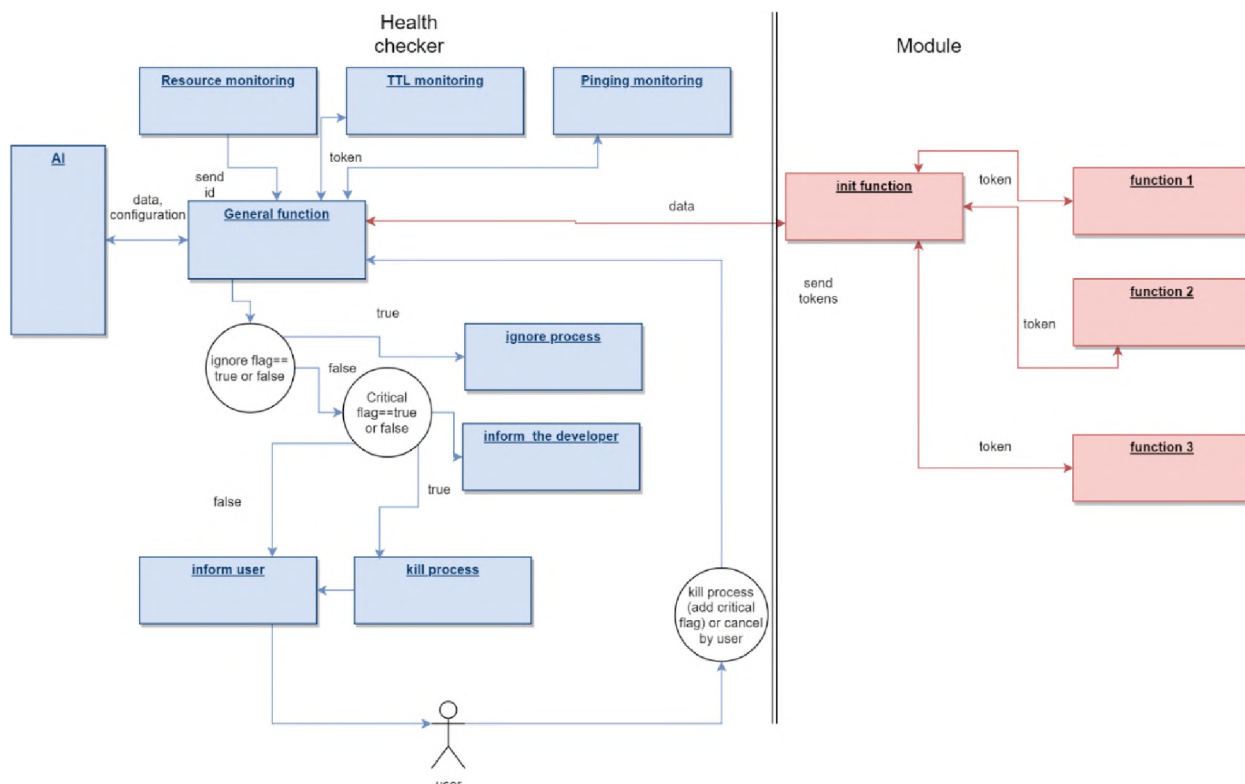


Рисунок 2.3 – Схема роботи моделі при невдалому відновленні компоненту

## 2.2 Вибір моделі нейронної мережі

В аналізі даних, виявленні аномалій (або виявленні викидів) називається знаходження та ідентифікація елементів, подій або спостережень, що не відповідають очікуваній поведінці або іншим елементам набору даних.

Є кілька головних питань, які потрібно вирішити перед початком проектування та побудови автоматизованої системи виявлення аномалій:

### 1. Своєчасність

Як швидко система повинна визначати, чи є щось аномалією чи ні? Чи потрібно виявляти аномалії в реальному часі або система може це виявляти через день, тиждень, рік.

Оскільки ОС повинна буде приймати рішення миттєво, а саме одразу після того як у роботі одного з сервісів сталася помилка, система виявлення аномалій повинна працювати у реальному часі.

## 2. Масштаб

Чи потрібно системі обробляти сто метрик чи мільйони? Кількість доступних ресурсів?

Оскільки ми говоримо про ОС, її продуктивність займає одну з основних вимог. Це означає, що система виявлення аномалій буде мати дуже обмежені ресурси.

## 3. Надійність без нагляду

Чи може система покладатися на зворотній зв'язок із людиною в деяких варіантах моделювання (наприклад, виборі алгоритмів або параметрів? Чи вона повинна отримувати надійні результати з абсолютно невідконтрольними параметрами? Оптимальним варіантом при модулюванні такої системи буде пошук параметрів, які б працювали однаково ефективно на всіх системах. Але треба обов'язково реалізувати систему оновлення, для того щоб була змога їх швидко змінити розробнику без участі користувача.

## 4. Динамічність даних

Чи мають тенденцію дані динамічно змінюватися, чи система змінюється відносно статично?

Для виявлення аномалій, коли відомі лише дані при нормальній роботі системи, можна використовувати алгоритми на базі однокласового методу опорних векторів (SVM): Protractor, EDR-, LCS-, DTW-ядро на базі SVM; Метод опорних векторів (SVM) завжди був корисним у виявленні аномалій через свою здатність забезпечувати нелінійну класифікацію за допомогою функції ядра[29-30]. Однокласові SVM використовуються для відокремлення даних одного конкретного цільового класу, від інших даних. Вони навчаються лише з позитивних прикладів, тобто точок даних з цільового класу.

Алгоритми машинного навчання, які були розглянуті вище мають один основний недолік, а саме архітектура “чорного ящика”. Дуже важко пояснити після навчання таких моделей, чому вона приймає ті чи інші рішення. Також основною проблемою таких моделей це час навчання та кількість необхідних ресурсів для їх роботи. Тому доцільно буде використовувати щось середнє між

детермінованими правилами та машинним навчанням і прикладом такого алгоритму є “Decision Tree”. Такий алгоритм швидко навчається і не використовує багато ресурсів, а також рішення, які даний алгоритм буде зрозумілі для людини[17-20].

### 2.3 Алгоритм прогнозування

У розділі 1.3 цієї роботи було розглянуто моніторинг та самовідновлення для веб-сервісів. Розглянемо алгоритм прогнозування на базі правил для системи, яка зображена на рисунку 1.16.

На основі набору правил створюються два списки:

1. Список помилок
2. Список запущених подій

$$TE - List = \{f_i \rightarrow \{e_{i1}, e_{i2}, \dots, e_{ik}\}: 1 \leq i \leq N_f\}$$

$$TE - List = \{e_m \rightarrow \{f_{m1}, f_{m2}, \dots, f_{mn}\}: 1 \leq m \leq N_e\}$$

Де  $f_i$  – це фатальні події, а  $e_i$  – не фатальні. Під час прогнозування, коли відбувається подія  $e$ :

1. Додати  $e$  в набір подій прогнозування

$$E = \{e_1, e_2, \dots, e_n\}$$

Де події сортуються у порядку зростання та видалення  $e_i$ , коли

$$|T_E - T_i| > T_{predict\_window}$$

2. Отримати потенційні збої, які можуть бути викликані  $e$  згідно

$$TF - List: e \rightarrow \{f_1, f_2, \dots, f_k\}$$

3. За кожну невдачу у наборі  $\{f_1, f_2, \dots, f_k\}$ , обробити список подій згідно з

$$TF - List: f_i \rightarrow \{e_{i1}, e_{i2}, \dots, e_{ik}\}$$

4. Якщо  $\{e_{i1}, e_{i2}, \dots, e_{ik}\} \subseteq E$ , тоді провести попередження о відмові  $f_i$ , яке може статися на протязі  $f_{prediction\_window}$

Оціночна метрика:

- Точність – це ймовірність того, що (випадково обраний) знайдений документ актуальний  $P = \frac{T_p}{T_p + F_p}$

- Повнота, це ймовірність того, що (випадково обраний) відповідний документ витягується в результаті пошуку.  $P = \frac{T_p}{T_p + F_n}$  [13]

## 2.4 Побудова детермінованих правил використовуючи алгоритм машинного навчання Decision Tree

Дерево прийняття рішень – це інструмент підтримки прийняття рішень, який використовує деревовидний граф або модель рішень і їх можливі наслідки, включаючи випадкові події, витрати ресурсів і корисність. Це один із способів відображення алгоритму, який містить тільки умовні оператори управління. Найчастіше після навчання з учителям результатом є дерево з правилами у вузлах і прогнозом. Вирішальне правило – це деяка функція від об'єкта, що дозволяє визначити, в яку з дочірніх вершин потрібно помістити даний об'єкт.

Алгоритм побудови:

1. Перевірити критерій – це основа алгоритму. Якщо він виконується, обрати для вузла прогноз, що можна зробити декількома способами.

2. Інакше треба розбити множину на декілька, які не перетинаються. У загальному випадку в вершині  $t$  задається вирішальне правило  $Q_t(x)$  приймає деякий діапазон значень. Цей діапазон розбивається на  $R_t$  непересічних множин об'єктів,  $S_1, S_2, \dots, S_{R_t}$ , де  $R_t$  - кількість нащадків у вершини, а кожне  $S_i$  - це безліч об'єктів, які потрапили в  $i$ -го потомка.

3. Множина в вузлі розбивається відповідно до обраного правилом, для кожного вузла алгоритм запускається рекурсивно.

Вирішальні правила. Найчастіше в якості  $Q_t(x)$  беруть просто одну із ознак, тобто  $x^{i(t)}$ :

- $S_t(j) = \{x \in X : h_j \leq x^{i(t)} \leq h_{j+1}\}$  для обраних  $h_1, \dots, h_{j+1}$
- $S_t(1) = \{x \in X : \langle x, v \rangle \leq 0\}; S_t(2) = \{x \in X : \langle x, v \rangle > 0\}$  перевірка вузла
- $S_t(1) = \{x \in X : p(x, x_0) \leq h\}; S^t(2) = \{x \in X : p(x, x_0) > h\}$  де відстань  $p$  визначено в деякому метричному просторі (наприклад,  $p(x, y) = |x - y|$ ).
- $S_t(1) = \{x \in X : x^{i(t)} \leq h\}; S_t(2) = \{x \in X : x^{i(t)} > h\}$  предикати,  $\langle x, v \rangle$  – скалярний добуток векторів

В цілому, взяти можна будь-які вирішальні правила, але краще - інтерпретовані, оскільки їх легше налаштовувати. Особливого сенсу брати щось складніше предикатів немає, так як вже з їх допомогою можна отримати дерево з високою точністю на навчальній вибірці.[22-23]

## 2.5 Виявлення аномалій у часових рядах за допомогою згорткових нейронних мереж

Згорткові нейронні мережі – це один із алгоритмів машинного навчання, який найчастіше використовується для аналізу зображення.

Часовий ряд можна розглядати як одновимірне зображення де єдиний вимір це час, тоді як типове зображення має дві характеристики: ширину і висоту. Багатовимірний часовий ряд може мати довільну кількість каналів, які можуть мати різні властивості та кореляцію один до одного. Підхід пропонує наступну архітектуру для сегментації часових рядів, як показано на рисунку 2.4



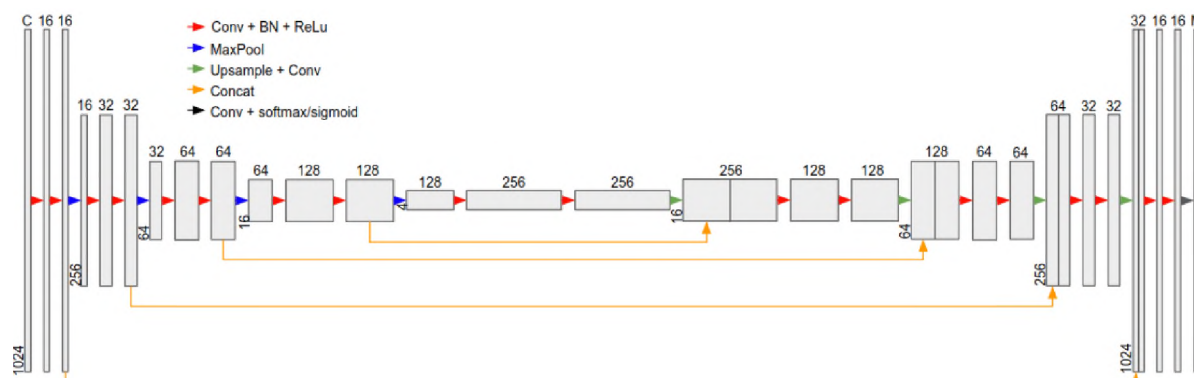


Рисунок 2.4 – Архітектура U-Net для сегментації часових рядів

Часовий ряд з довжиною каналів 1024 і кількістю каналів  $C$ , закодований п'ятьма секціями згорткових шарів. Кожний розділ включає два шари блоків, кожен з яких включає згортковий шар, шар нормалізації та шар ReLu. Для шарів згортання розмір ядра дорівнює 3. Кількість фільтрів у кожному шарі згортки збільшуються від 16 до 256 з коефіцієнтом 2. Для кожного шару згортання, застосовується заповнення нулями і крок рівний 1. Між розділами кодування, max pooling шари з розміром пулу, що дорівнює 4, використовуються для зменшення вибірки рядів. Після цього слідує чотири розділи декодування, де кожен розділ включає шар з підвищеною частотою дискретизації та два conv + BN + ReLu блоки. Шар з підвищеною частотою дискретизації також дорівнює 4, а шари згортання використовують таку ж кількість фільтрів, що і їх аналог кодування, роблячи архітектуру симетричною. Важливою особливістю U-Net є пропуск каналів між відповідні розділи кодування та декодування. Це реалізовано шляхом об'єднання виходу кожного max pooling шару з виходом шару з підвищеною частотою дискретизації та виконання згортки над об'єднаною ознакою ряду.

Останній розділ містить шар згортки з ядром розмір якого дорівнює одиниці та активаційний шар. Іноді різні класи аномалій не є взаємовиключними, а отже існує проблема з багато-класовими даними, тобто часова точка може бути віднесена до декількох класів одночасно. Наприклад, сплеск періодичного ряду одночасно є адитивною аномалією та сезонною. У

такому випадку кількість фільтрів при остаточному шарі згортки повинен дорівнювати кількості класів аномалій  $M$ , а кінцева функція активізації сигмоїдальною. Якщо всі класи аномалії взаємовиключні, вихід має глибину  $M+1$  і додаткову колонку для номінального класу (тобто немає аномалії).

При розгортанні натренованої моделі для роботи в потоковому режимі, треба регулярно отримувати пакети останніх даних та запускати модель на них. Частота отримання даних з сенсорів повинна бути як мінімум така, щоб кожна точка часу оцінювався моделлю як мінімум один раз. Можна використовувати частоту подавання даних в кілька разів вище ніж мінімальна межа, щоб кожен момент часу оцінювався моделлю декілька разів. Таким чином ми отримаємо декілька результатів за один і той самий проміжок часу, щоб отримати більш надійне виявлення.

Також модель має фіксований вхід 1024. Не потрібно завжди використовувати це значення як розмір даних одного пакету. Довжина ряду у пакеті повинна бути визначена за частотою потоку і часовою шкалою аномальної поведінки.

Стратегія розсувного вікна також може працювати з моделлю CNN, яка класифікує послідовності за аномальністю. Однак вибір довжини пакету, який подається на вхід буде проблематичним. Якщо ряд у пакеті буде занадто довгий, ряд з аномалією з великою вірогідністю буде класифікований неправильно. З іншого боку, занадто короткий ряд у пакеті буде містити мало контексту, що теж приведе до поганих результатів класифікатора.

Щодо нормалізації даних, відмінно від звичайних проблем обробки зображень для яких зазвичай використовують U-net моделі, значення RGB каналу завжди має значення від 0 до 255. Часові ряди можуть мати довільно великі або малі значення. Тому постає потреба у нормалізації вхідних даних.

Мала кількість подій пов'язаний з помилками у наборі даних, часто обмежує модель навчання. Трансферне навчання – це стратегічне вирішення питання про обмеженість даних. Підхід трансферного навчання використовує ваги базової моделі натренованої на доступному великомасштабному наборі

даних, а потім точно налаштовує ваги моделі використовуючи невеликий набір даних. Процес налаштування гіпер-параметрів використовує переваги моделі натренованої на великомасштабному наборі даних, таким чином отримуючи корисні особливості із вхідних даних, а тому стає необхідність у різкому зменшенні набору даних для забезпечення сходження без перетренування моделі.

Для трансферного навчання інших універсальних завдань можна залишити архітектуру моделі як показано на рисунку 2.1, але шар виходу повинен змінитися відповідно до числа класів у цільовому завданні. Ваги усіх шарів крім секції виходу повинна бути ініціалізована з вагами попередньо навчаної моделі. Існує дві стратегії тонкого налаштування з хорошими показниками на тестах. Перший – це встановити різні множники швидкості навчання для 10 розділів (5 секції кодування, 4 секції декодування та вихідного розділу) 0.01, 0.04, 0.09, ..., 0.81, 1.0. Другий – це залишати незмінними ваги у перших двох розділах і налаштовувати ваги інших розділів, після чого почати змінювати ваги першого розділу налаштовуючи усі інші ваги.

Використання U-Net архітектури у вирішенні багатоваріантних задачах може бути проблематичним. Ядро першого шару згортки мало б іншу форму, оскільки  $C$  не дорівнює 1. Якщо ці ядра ініціалізовані випадковим чином, тоді перенесення ваг глибших шарів марно, тому що функції найнижчого порядку виконуються по-різному. Якщо ми повторимо матрицю ваг  $3 \times 1 \times 16$  в першому шарі  $C$  разів і створимо ініціалізацію вагової матриці  $3 \times C \times 16$  для передачі, математично це еквівалентно передачі суми всіх рядів. Поки сума RGB-каналів може зберігати важливі властивості зображень, сума каналів часових рядів як правило, безглуздо, наприклад, канали різні датчиків в системі IoT, такі як тиск і температура.

Тому виникає необхідність у модифікованій архітектурі U-Net для передачі ваг з натренованої моделі при виконанні багатоваріантних задач, як показано на Рисунку 2.5

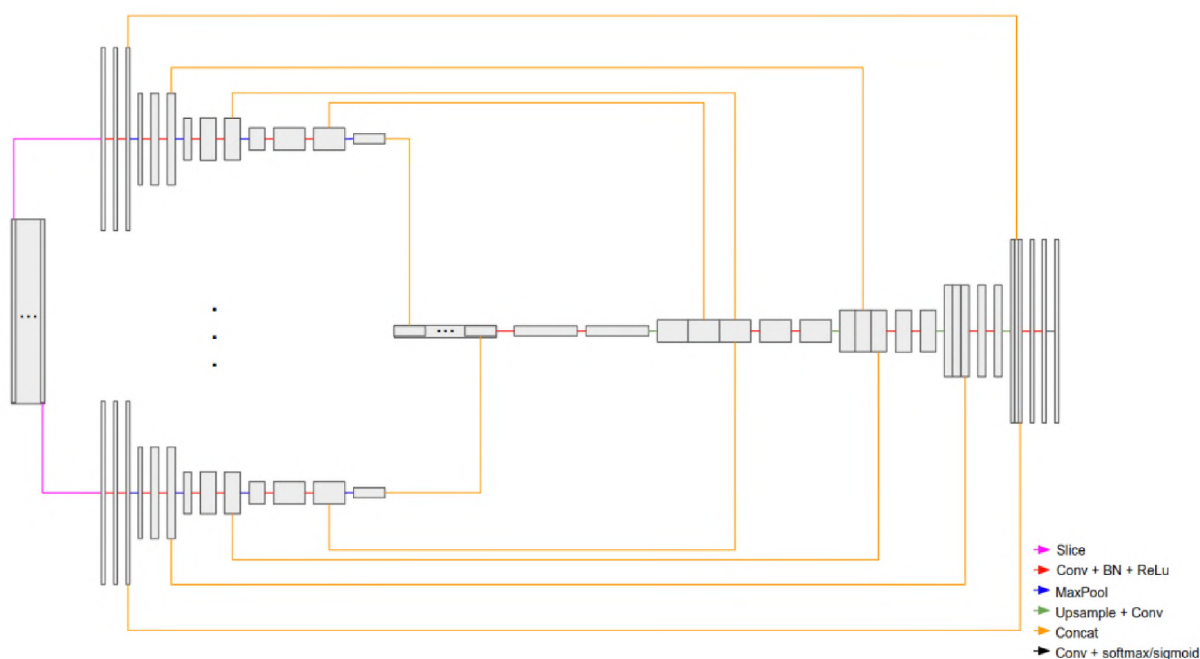


Рисунок 2.5 – Архітектура MU-Net, мережевої передачі на основі U-Net для багатоваріантних завдань.

Подібно до U-Net, нова архітектура також має структуру кодування-декодування. Однак канали  $C$  вхідних рядів розділені спочатку шаром нарізки, а потім кожен канал має свої власні уніваріантні секції кодування, як і перші чотири розділи кодування в U-Net. Вихід з четвертого розділу кодування над усіма каналами буде об'єднаний, перед інтеграцією у п'ятий розділ кодування, за яким слідує чотири секції декодування і остаточний вихідний розділ, такий самий, як у U-Net. [24-27]

## 2.6 Постановка проблеми

Розпочнемо огляд проблеми з вимог до системи. Нехай є  $n$  телевізорів (клієнтів), які мають  $k$  модулів. Модулі повинні в свою чергу отримувати оновлення та конфігурацію з серверу, а також сервер повинен мати можливість

відправляти команди до них, а саме команди типу: Install, Uninstall, Execute. Система повинна збирати інформацію про успішність виконання команд відправлених до клієнта за період  $t$  та мати можливість перевірити коректність роботи кожного з компонентів на стороні клієнта. Уся зібрана інформація обробляється на стороні серверу. Для зручного керування клієнтами, система повинна розділяти їх на групи.

Розглянемо систему з наступною архітектурою, яка показана на рисунку 2.6

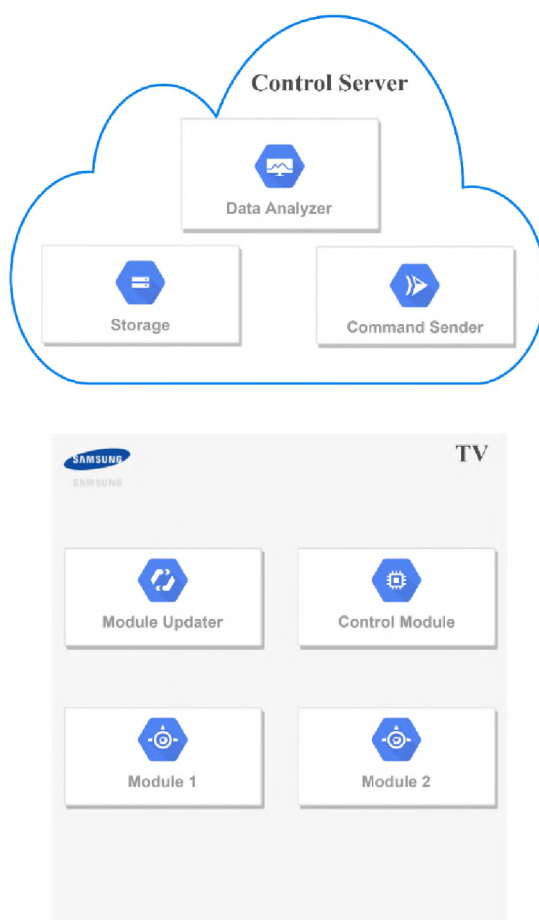


Рисунок 2.6 – Архітектура системи.

Система має архітектуру клієнт-сервер. На стороні клієнта розташовані:

1. Нефіксована кількість модулів (плагінів). Будемо їх позначати “Module №”

2. Module Updater – це модуль який відповідає за оновлення конфігурації модулів типу “Module №”.

3. Control Module – модуль приймає команди з сервера та відправляє їх до модулю Module Updater або Module + № або виконує її в залежності від команди, яку він отримав.

На стороні сервера розташовані три основні модуля:

1. Storage – це сховище даних, а саме конфігурацій та оновлень, до якого може звертатися “Module Updater” для їх завантаження.

2. Command Sender – це модуль для відправки команд до “Control Module”, який розташований на стороні клієнта.

3. Data Analyzer – модуль, який отримує дані від клієнтів та аналізує їх.

Розглянемо більш детально процес відправки команд до клієнтів.

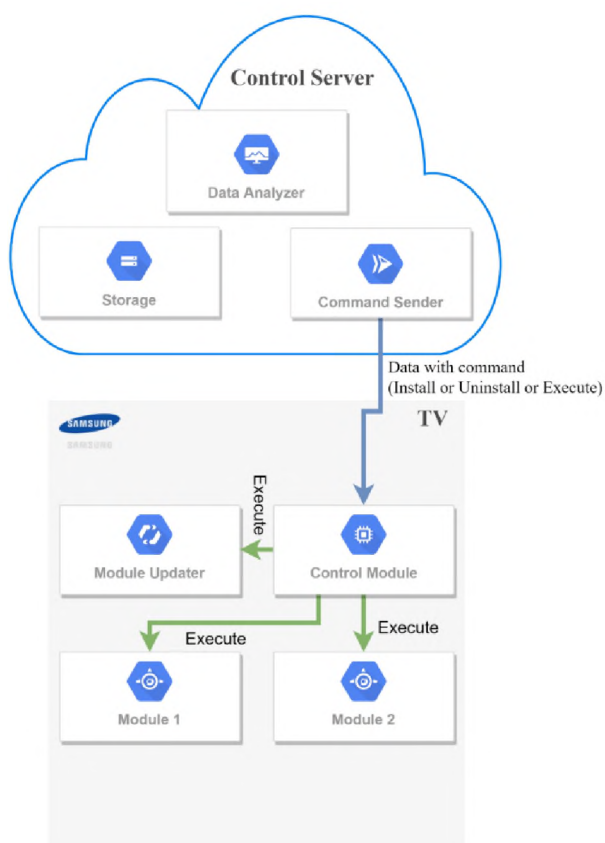


Рисунок 2.7 – Схема відправки команд від сервера до клієнта

Як показано на рисунку 2.7, команди відправляються с серверу через модуль “Command Sender”. Команди приймає на стороні клієнта “Control Module”. Якщо до нього було надіслано команду типу “Install Module №”, тоді “Control Module ” виконує встановлення Module №, аналогічно для команди Uninstall. Команди типу “Execute Module\_Name Function p”, де Module\_Name – це ім’я модулю, до якого “Control Module ” повинен передати команду. Команду Execute також може виконати і сам “Control Module ”. Параметр Function у команді – це інструкція, що саме модуль до якого адресована команда повинен зробити. p – це додаткові (необов’язкові) параметри для команди.

Стосовно оновлення модулів на телевізорі воно розділяється на два типи. Повне оновлення модулю та оновлення конфігурації. При повному оновленні модулю “Control Module ” автоматично завантажить нову версію “Module №” як тільки вона з’явиться у “Storage” і версія “Module №” буде більша за ту, яка встановлена на клієнті. Після чого “Control Module ” буде чекати команди на встановлення нової версії. При оновленні конфігурації, до “Control Module ” повинна прийти команда типу “Execute Module\_Updater DownloadUpdate”. “Control Module ” відправить цю команду до “Module Updater”, який у свою чергу завантажить конфігурацію з серверу (“Storage”) і оновить конфігурацію цільових “Module №”. Версія модулю при оновленні конфігурації не змінюється. Детальна схема роботи зображена на рисунку 2.8

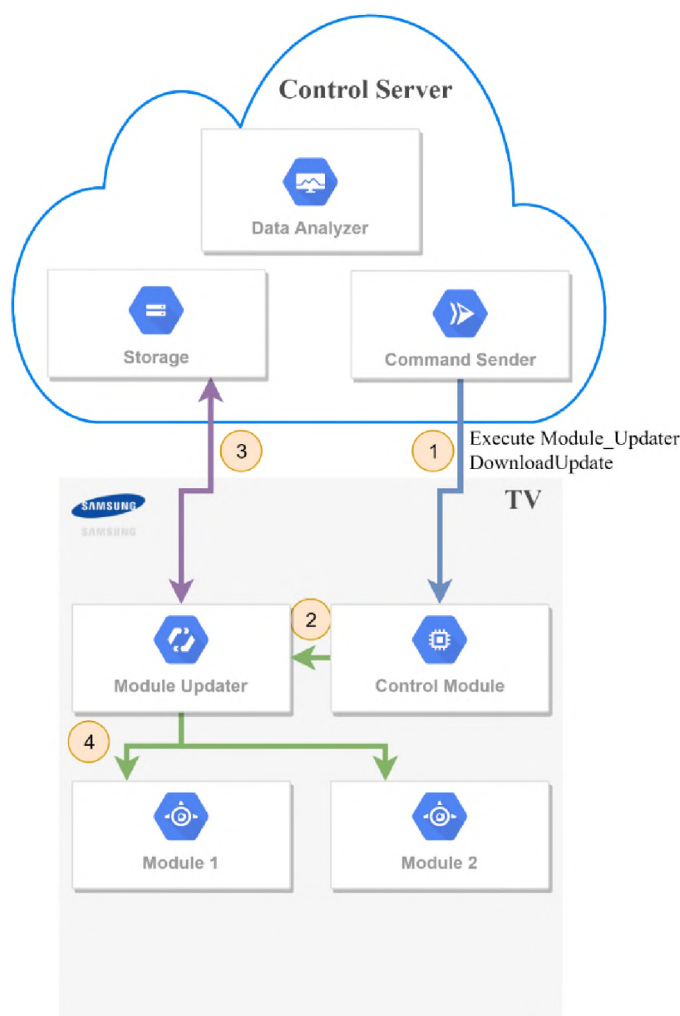


Рисунок 2.8 – Схема оновлення конфігурації модулів

Система повинна мати функцію моніторингу та відновлення у разі неправильної роботи одного з модулів на стороні клієнта. На стороні сервера розташуймо модуль "Data Analyzer", який використовуючи модуль "Command Sender". Він може відправити команду "Execute Control\_Module GetCommandStatus t", де параметр t – це період. Отримавши дану команду, "Control Module" повинен зібрати інформацію о всіх отриманих командах від серверу, а саме успішність їх виконання за період t. Якщо наприклад за період t до клієнта надходила команда "Install Module №", тоді він повинен відправити до "Command Sender" звіт про успішність або неуспішність виконання цієї команди. Аналогічно "Data Analyzer", може надіслати команду до "Command Sender" для отримання статусу модулю, щоб перевірити його стан. Тобто, чи



працює модуль правильно. Команда повинна мати вигляд: “Execute Control\_Module GetModuleStatus”.

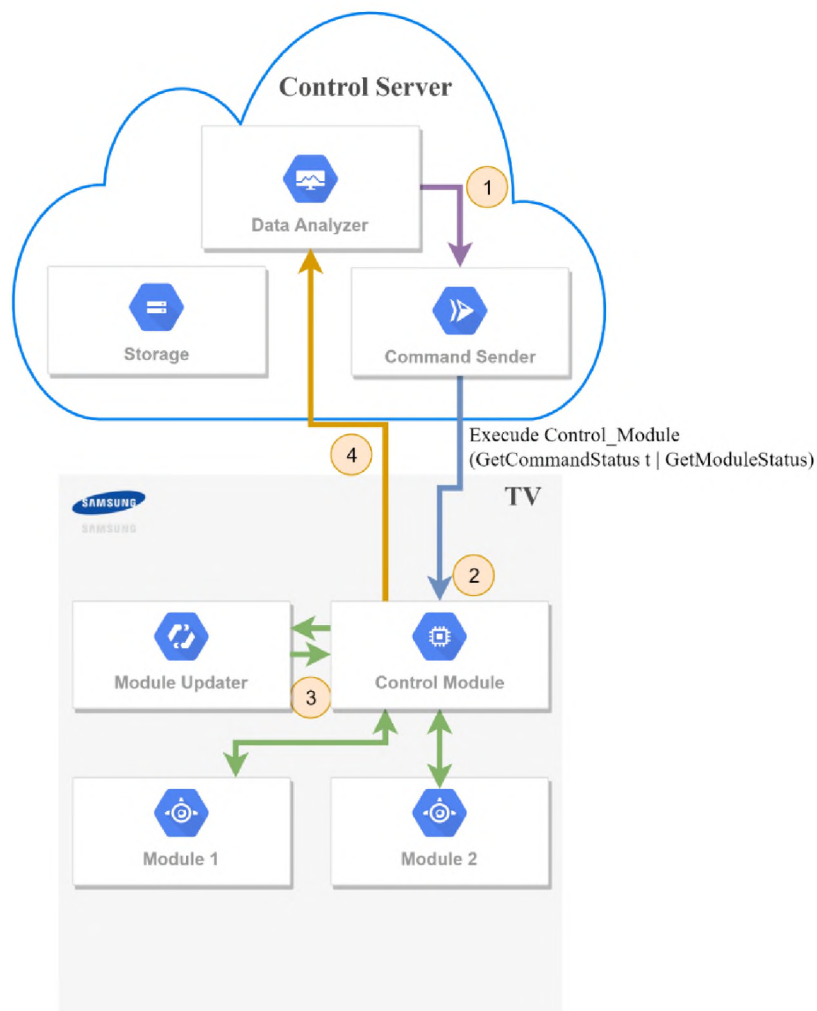


Рисунок 2.9 – Схема моніторингу системи

Точна кількість клієнтів невідома, але треба враховувати витрату трафіку для пересилання даних між клієнтом та сервером. Тобто необхідно максимально стискати дані до відправки.

Після збору даних з клієнтів у “Data Analyzer”, система повинна обробити їх і розділити клієнтів на групи. Розділення на групи дає нам можливість відновлювати нормальну роботу телевізорів з однаковою проблемою. Тобто, якщо на  $n/2$  телевізорах не працює “Module 1”, ми віднесемо їх всіх до групи

“b”, а телевізори, які працюють в нормальному режимі до групи “a”. Після цього система може почати відновлення всіх клієнтів, які відносяться до групи “b”. Тому кожен клієнт повинен мати універсальний ідентифікатор для розділення їх на групи.

Після того, як система виявила несправність і віднесла всіх користувачів з нею до однієї групи, вона повинна прийняти рішення щодо відновлення базуючись на інформації, яка надійшла до “Data Analyzer”. Вона повинна попередити про помилку розробників та автоматично прийняти міри по відновленню, шляхом оновлення конфігурації, повного оновлення модулю, повернення модулю до попередньої версії або його відключення якщо проблему неможливо вирішити автоматично. Якщо роздивлятися процес відновлення детальніше, краще описати логіку її роботи при виявленні аномалії наступними правилами:

1. Спробувати повторно встановити модуль для групи “b”, якщо модуль працює нормально для групи “a” і версій модулів, конфігурації та платформи співпадають.
2. Спробувати оновити конфігурацію модулю для групи “b” на таку саму як у групи “a”, якщо вони мають різні конфігурації та модуль у групи “a” працює в нормальному режимі і версій модулів та платформи співпадають.
3. Повернути конфігурацію до попередньої версії, якщо операція (2) закінчилася невдачею або версії конфігурацій групи “a” та “b” різні, але вони мають однакову версію платформ і однакову версію модуля, або групи “a” не існує.
4. Спробувати встановити версію модулю для групи “b”, яка встановлена для групи “a”, якщо у групі “a” модуль працює нормально, версія модуля у групі “a” і групі “b” різні, версії платформ однакові.
5. Повернути модуль до попередньої версії, якщо група “a” та “b” мають різні платформи або операція (4) закінчилася невдачею, або операція (1) закінчилася невдачею, або операція (3) закінчилася невдачею.
6. Вимкнути модуль, якщо операція (5) закінчилася невдачею.

Також є необхідність у вирішенні іншої проблеми. Оскільки системою можуть користуватися  $n$  клієнтів, ми повинні мати  $f$  “Control Server” для їх обслуговування. Як і клієнти, ці сервери повинні автоматично відновлювати свою роботу при неправильній роботі. Оскільки вони повинні приймати величезну кількість інформації від клієнтів, обробляти її, зберігати та відправляти відфільтровані логи до серверу розробників. Повна архітектура такої системи зображена на рисунку 2.10

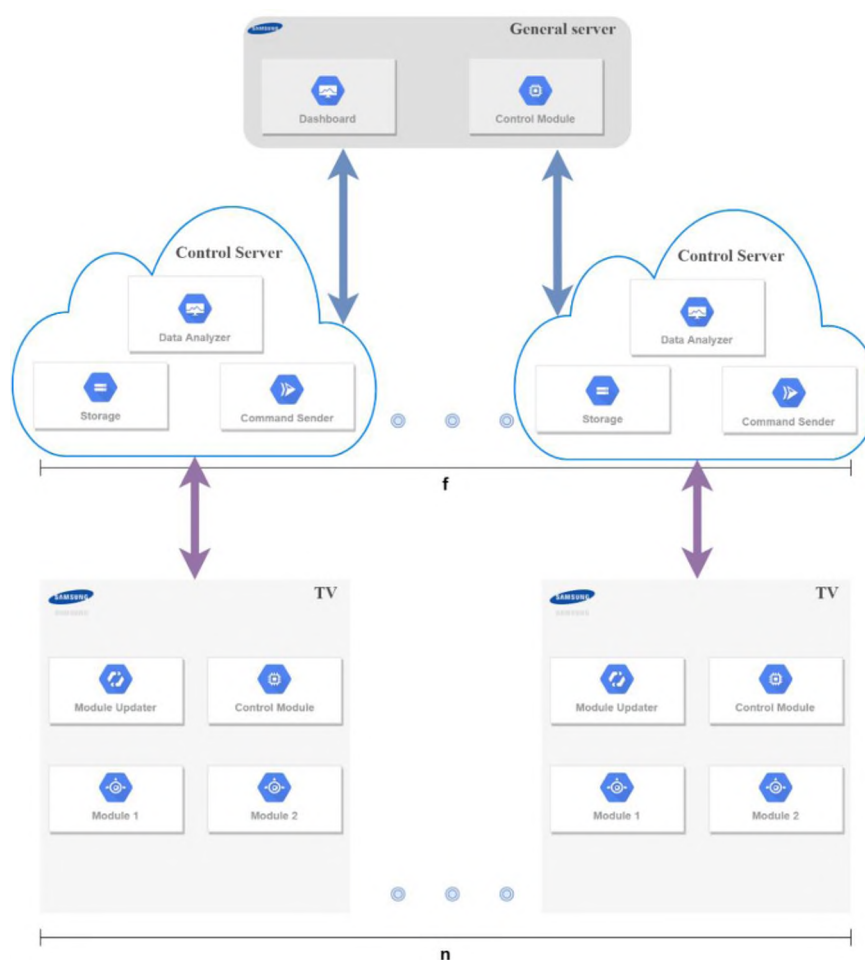


Рисунок 2.10 – Повна схема системи

На сервері розробників “General server”, повинна бути реалізована система моніторингу, автоматичного відновлення “Control Server”.

## Висновки до розділу 2

В даному розділі було розглянуто приклад поєднання підходів моніторингу, пінгування та TTL з використанням штучного інтелекту. Таким чином ми змогли усунути недоліки кожного з підходів об'єднавши їх і використовуючи для задач, де ці підходи показують максимальну ефективність. Також було розглянуто підхід на базі правил для моніторингу системи на наявність помилок при її роботі. Запропоновано використовувати згорткові мережі у поєднанні з CNN[28] або іншими класифікаторами для моніторингу часових рядів у реальному часі.

Розглянуто алгоритм машинного навчання Decision Tree як спосіб класифікації компоненту системи, оскільки інші алгоритми машинного навчання, які було розглянуто, мають один основний недолік, а саме архітектуру “чорного ящика”. Дуже важко пояснити після навчання таких моделей, чому вона приймає ті чи інші рішення. Також основною проблемою таких моделей це час навчання та кількість необхідних ресурсів для їх роботи. Decision Tree швидко навчається, не потребує великої кількості ресурсів для своєї роботи, але без правильної обробки даних перед їхнім поданням на вхід моделі, важко отримати високі результати точності.

Проаналізовано і побудовано архітектуру системи з властивостями самовідновлення, яка буде забезпечувати нормативні вимоги безпеки, оскільки система розрахована на велику кількість клієнтів. Запропонована система має властивості самовідновлення, які були визначені у першому розділі даної роботи, оскільки вона має функцію моніторингу компонентів системи, може приймати рішення по їх відновленню та сповіщає розробників при виявленні проблем у компонентах. Запропонована система повинна мати високу точність детектування неправильної роботи компонентів системи, а також відновлювати їх без втрати функціоналу.

### **3 САМОВІДНОВЛЕННЯ КОМПОНЕНТІВ РОЗПОДІЛЕНОЇ СИСТЕМИ**

Метою цієї частини є розробка розподіленої системи з властивостями самовідновлення. Для забезпечення критерію доступності, оскільки даною системою, архітектура якої зображена на рисунках 2.6 – 2.10, буде користуватися велика кількість клієнтів. Тому постає необхідність у реалізації системи моніторингу встановлених модулів на стороні клієнта та моніторингу і відновлення сервісів на стороні “Control Server”. Всі команди, які передаються між клієнтом та сервером, повинні стискатися алгоритмом без втрат для економії трафіку.

#### **3.1 Моніторинг та відновлення клієнтів на базі правил**

Для системи, архітектура та вимоги якої були описані у розділі 2.5 було реалізовано відправку команд до клієнтів. Відправку команд до клієнтів можна здійснити двома способами. Через DBoard або їх відправить модуль “Data Analyzer” через модуль “Command Sender” в автоматичному режимі, якщо якийсь компонент працює некоректно. Саме модуль “Data Analyzer” забезпечує нашій системі самовідновлення модулів на стороні клієнтів.

Для відправки команди на телевізор першим способом потрібно зайти на DBoard – це веб-сторінка, через котру можна згенерувати команду за зразком або написати в ручному режимі. Усі команди генеруються у JSON форматі та передаються до клієнтів де в свою чергу “Control Module” її обробляє. Також через DBoard можна створювати групи, дивитися усі зареєстровані клієнти та дивитись різні репорти, які надходять від клієнтів. Сам DBoard та приклад генерації команди зображено на рисунку 3.1

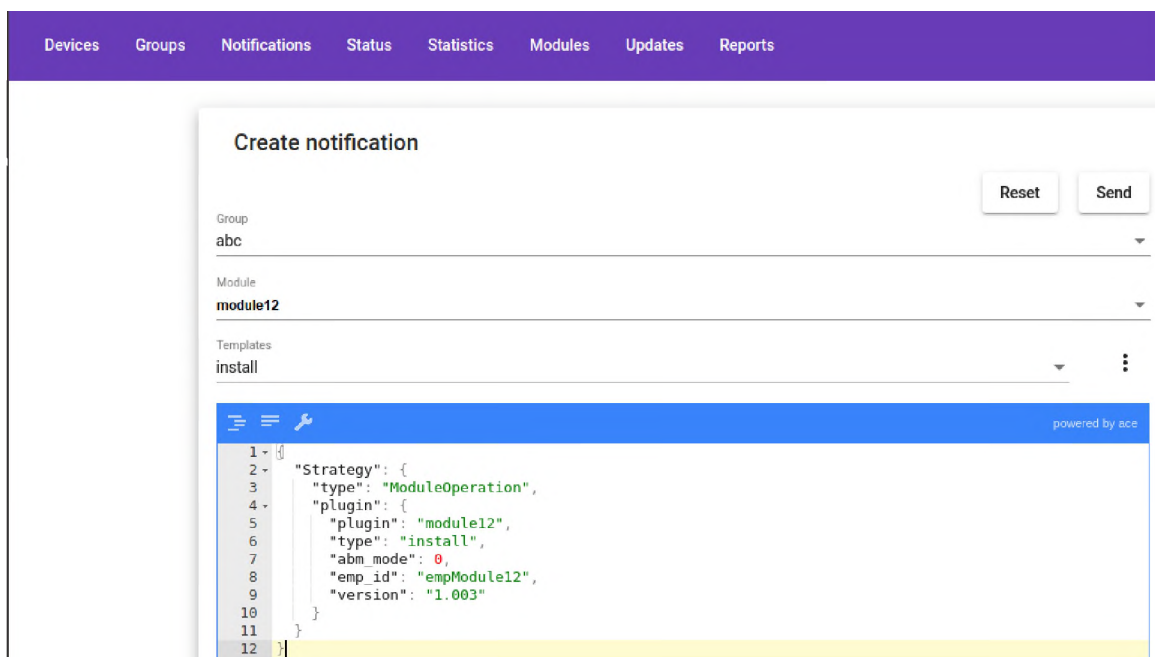


Рисунок 3.1 – веб-сторінка для контролю груп “DBoard”

Після натиску на кнопку “send”, команда у JSON форматі передається до клієнтів, які належать до групи “abc”. Отримання команди можна перевірити підключившись до телевізора по “ssh” та подивившись логи, які пише “Control Module” використовуючи “dlogutil”.

Після того як “Control Module” зафіксував неправильну роботу одного з модулів, він створює репорт та відправляє його до “Control Server”. Перевірка модулів на стороні клієнта виконується шляхом комбінування методів перевірки доступності та TTL, які описані у розділі 1.1.2 та 1.1.3. “Control Server” обробляє автоматично усі прийняті репорти та розподіляє їх на групи. У нашому випадку на групу де модуль працює нормально і на групу де модуль має проблему. Дві групи цільова та еталонна відправляються до “HealManager” з даними версії модулю, версії платформ, версії конфігурації. Після чого “HealManager” викликає “Healalgorithm”, який відновлює модуль цільової групи на базі правил. “Healalgorithm” працює за наступним алгоритмом, який представлений у вигляді псевдокоду:

Таблиця 3.1 Псевдокод алгоритму роботи модуля “Healgorithm”

Customer group healing procedure

Input: Two object a,b where a - reference group, b - target group. Objects have fields: ver\_module - version of target module,

ver\_plat - version of platform where module install, ver\_conf - version of target module configuration.

Output: Result code

steps = [1-6]{0}

if a.ver\_module == b.ver\_module and a.ver\_plat == b.ver\_plat and a.ver\_conf == b.ver\_conf:

result = Send-Command-To-Group-B-For-Reinstall-Module()

if result == success:

return 1

else:

steps[1] = 1

if a.ver\_conf != b.ver\_conf and a.ver\_plat == b.ver\_plat and a.ver\_module == b.ver\_module:

result = Send-Command-To-Group-B-For-Update-Configuration-that-equal-to-Group-A()

if result == success:

return 2

else:

steps[2] = 1

if (steps[0] == 1 or steps[1] == 1) or a == NULL:

result = Send-Command-To-Group-B-For-Revert-Configuration-to-previous-version()

if result == success:

return 3

else:

steps[3] = 1

if a.ver\_module != b.ver\_module and a.ver\_plat == b.ver\_plat:

result = Send-Command-To-Group-B-For-Install-Module-Version-that-equal-to-Group-A()

if result == success:

return 4

else:

steps[4] = 1

if a.ver\_plat != b.ver\_plat or steps[3] == 1:

result = Send-Command-To-Group-B-For-Revert-Module-to-previous-version()

if result == success:

return 5

else:

steps[5] = 1

if steps[5] == 1

result = Send-Command-To-Group-B-For-Turn-off-module()

if result == success:

return 6

Для того, щоб показати роботу “HealManager” та “Healalgorithm”, було зібрано дві прошивки, але в одній із них, модуль звертається до файлу за неправильним шляхом і тепер він завжди повертає статус “error”. Таким чином на телевізорі з цією прошивкою “Control Module” зможе повідомити “Control Server”, що модуль працює неправильно, а “Control Server” у свою чергу вважає, що на стороні клієнта сталася помилка і запустить процедуру відновлення. Телевізор з нормальною прошивкою буде виступати як еталон. Результатом роботи “Healalgorithm” повинно бути відновлення конфігурації, фактично неправильний шлях повинен замінитися на той, який встановлено на нормальній прошивці. Після завантаження прошивки на телевізори, підключаємося до серверу по ssh. Використовуючи “dlogutil” та “grep” виведемо та відфільтруємо логи з “HealManager” та “Healalgorithm”. Результати роботи наведені на рисунку 3.2

```
HealManager:INFO: group: <abc>, plugin: <module12>, status: <fail>
HealManager:INFO: founded reference group <a>
HealManager:INFO: group: <a> , plugin: <module12>, status: <normal>
HealManager:INFO: starting Healalgorithm() Tue Oct 15 15:30:17 UTC 2019
Healalgorithm:INFO: Init step 1 Tue Oct 15 15:30:18 UTC 2019
Healalgorithm:INFO: step 1 failed Tue Oct 15 15:38:54 UTC 2019
Healalgorithm:INFO: Init step 2 Tue Oct 15 15:38:55 UTC 2019
Healalgorithm:INFO: step 2 success Tue Oct 15 15:41:03 UTC 2019
HealManager:INFO: SendReport to [REDACTED]
```

Рисунок 3.2 – Приклад роботи модулів відновлення на базі правил

Після завершення роботи “HealManager” підключаємося до телевізора через “sdb”. Легше всього перевірити результат відновлення шляхом пошуку спеціального файлу “Policy.json” у який “Control Module” пише поточний стан всіх модулів на телевізорі. Через cat виводимо вміст файлу, до і після відновлення. Поле state 8 – означає що модуль працює правильно, 6 – модуль встановлено але він має проблеми і не працює.



```

sh-3.2# cat Policy.json
{
  "Control Module" : {
    "plugins" : {
      "module1" : {
        "emp" : "empnModule1",
        "id" : "057382",
        "state" : 8,
        "ver" : "1.002"
      },
      "module2" : {
        "emp" : "empnModule2",
        "id" : "087367",
        "state" : 8,
        "ver" : "1.007"
      },
      "module3" : {
        "emp" : "empnModule3",
        "id" : "023654",
        "state" : 8,
        "ver" : "1.003"
      },
      "module12" : {
        "emp" : "empnModule12",
        "id" : "096789",
        "state" : 6,
        "ver" : "1.003"
      }
    }
  }
}
sh-3.2#

sh-3.2# cat Policy.json
{
  "Control Module" : {
    "plugins" : {
      "module1" : {
        "emp" : "empnModule1",
        "id" : "057382",
        "state" : 8,
        "ver" : "1.002"
      },
      "module2" : {
        "emp" : "empnModule2",
        "id" : "087367",
        "state" : 8,
        "ver" : "1.007"
      },
      "module3" : {
        "emp" : "empnModule3",
        "id" : "023654",
        "state" : 8,
        "ver" : "1.003"
      },
      "module12" : {
        "emp" : "empnModule12",
        "id" : "096789",
        "state" : 8,
        "ver" : "1.003"
      }
    }
  }
}
sh-3.2#

```

Рисунок 3.3 – Policy.json до і після відновлення цільового модулю

Всі команди, які передаються між сервером та клієнтом стискаються алгоритмом без втрат з використанням словника RLE[31]. За цим алгоритмом дані розбиваються на слова, а потім замінюються на індекс в словнику. Даний словник знаходиться як на стороні сервера так і на стороні клієнта. Після генерації команди в json форматі вона передається на сервері в zip module, у цьому модулі вона стискається, передається до клієнта після чого дані конвертуються до попереднього стану і обробляються “Control Module”. Наприклад за цим алгоритмом команда, яка зображена на рисунку 3.1, “Strategy” заміниться на 4 що значно коротше. Враховуючи що сервер відправляє в середньому 3 команди до 1000000 клієнтів, без використання алгоритму стискання витрачається приблизно 621 мб в день, з використанням

приблизно 228 мб, тобто за допомогою цього алгоритму вдалося зменшити дані, які передаються від сервера до клієнта на 63 відсотка.

Таблиця 3.2 Стискання команд за алгоритмом RLE

Original command	Command in byte representation	Command after compression in byte representation
<pre> {   "Strategy": {     "type": "ModuleOperation",     "plugin": {       "plugin": "module12",       "type": "install",       "abm_mode": 0,       "emp_id": "empModule12",       "version": "1.003"     }   } } </pre>	<pre> 123 10 32 32 34 83 116 114 97 116 101 103 121 34 58 32 123 10 32 32 32 32 34 116 121 112 101 34 58 32 34 77 111 100 117 108 101 79 112 101 114 97 116 105 111 110 34 44 10 32 32 32 32 34 112 108 117 103 105 110 34 58 32 123 10 32 32 32 32 32 32 34 112 108 117 103 105 110 34 58 32 34 109 111 100 117 108 101 49 50 34 44 10 32 32 32 32 32 32 34 116 121 112 101 34 58 32 34 105 110 115 116 97 108 108 34 44 10 32 32 32 32 32 32 34 97 98 109 95 109 111 100 101 34 58 32 48 44 10 32 32 32 32 32 32 34 101 109 112 95 105 100 34 58 32 34 101 109 112 77 111 100 117 108 101 49 50 34 44 10 32 32 32 32 32 32 34 118 101 114 115 105 111 110 34 58 32 34 49 46 48 48 51 34 10 32 32 32 32 125 10 32 32 125 10 125 0 </pre>	<pre> 123 34 49 34 58 32 123 34 50 34 58 32 34 49 48 34 44 34 51 34 58 32 123 34 52 34 58 32 34 50 53 34 44 34 53 34 58 32 34 49 53 34 44 34 54 34 58 32 48 44 34 55 34 58 32 34 56 48 34 44 34 56 34 58 32 34 49 46 48 48 51 34 125 125 125 0 </pre>
Size	207 byte	76 byte

### 3.2 Моніторинг та відновлення серверів обслуговування клієнтів

Як зазначалося раніше, у розподіленій системі, яка зображена на рисунку 2.10, встановлено f “Control Server”. Вони відповідають за моніторинг та відновлення клієнтів, оновлення їх модулів, розсилання команд. Оскільки клієнтів у системі може бути багато, ці сервери також повинні мати властивість самовідновлення. Вони потенційно можуть мати проблеми, які пов’язані із заповненням пам’яті, а також можливі проблеми такого типу як заповнення бази даних що приведе до стрімкого росту часу, яке необхідно для обробки запиту, відмова одного з сервісів, системний збій.

За моніторинг та відновлення “Control Server” відповідає “General server” сервер. Детальна його архітектура зображена на рисунку 3.4

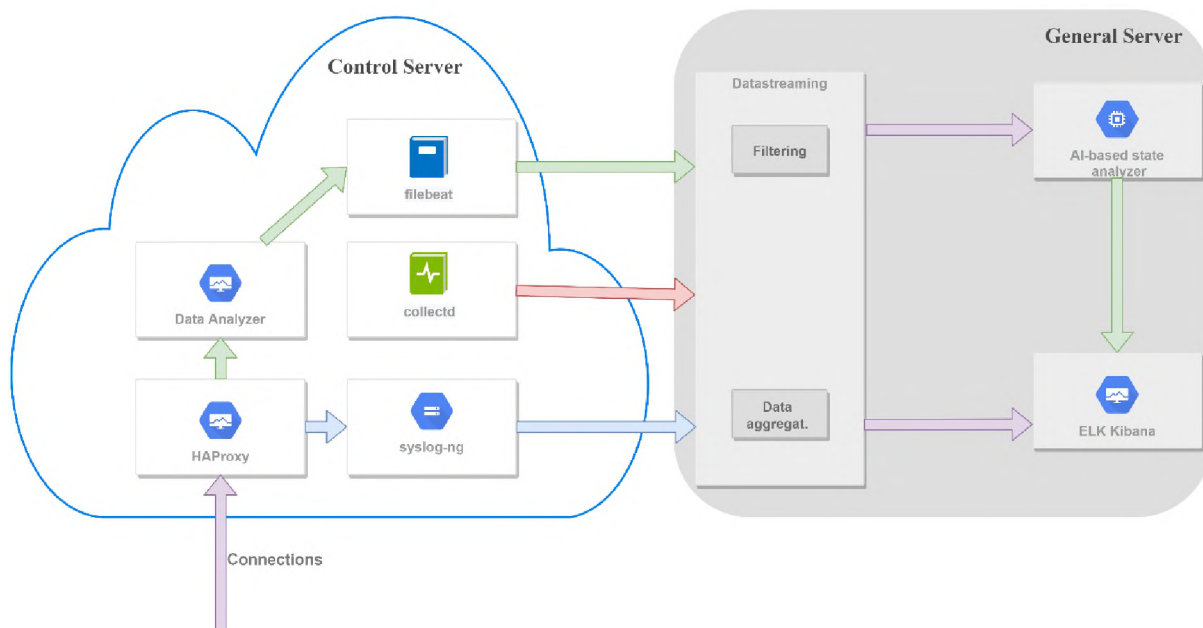


Рисунок 3.4 – Схема моніторингу “Control Server”

На стороні “Control Server” був встановлений “HAProxy” – програмне забезпечення, яке забезпечує балансування навантаження і проксі-сервер для

додатків на основі TCP і HTTP, які розподіляють запити між кількома серверами, а також були встановлені сенсори: “syslog-ng” – приймає, обробляє та відправляє логи, “collectd”[34] – це маленький демон, який збирає статистику про використання ресурсів системи, “filebeat”[32] – служить для відправки логів з модулю “Data Analyzer”. Після цього всі логи фільтруються та над ними проводиться агрегація, після чого відправляються до аналізатора на базі штучного інтелекту і відображаються на “ELK Kibana”[32]. Результат роботи сенсорів та “ELK Kibana”[32] зображено на рисунку 3.5

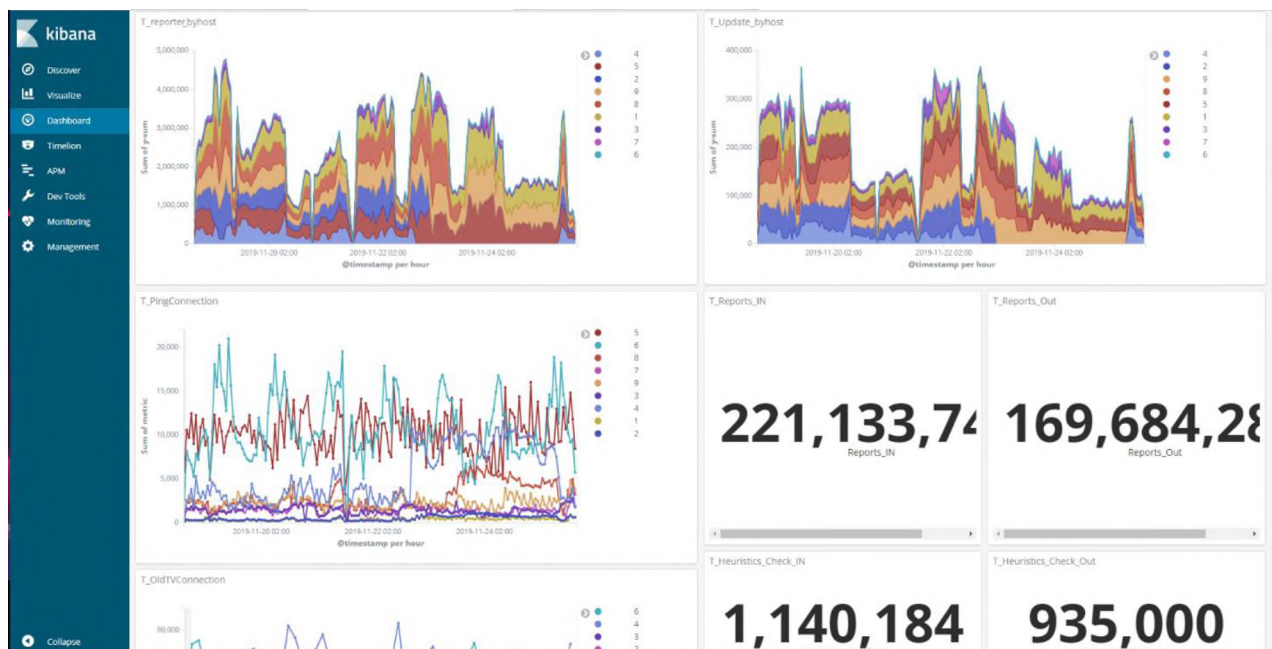


Рисунок 3.5 – Результат роботи сенсорів через “ELK Kibana”

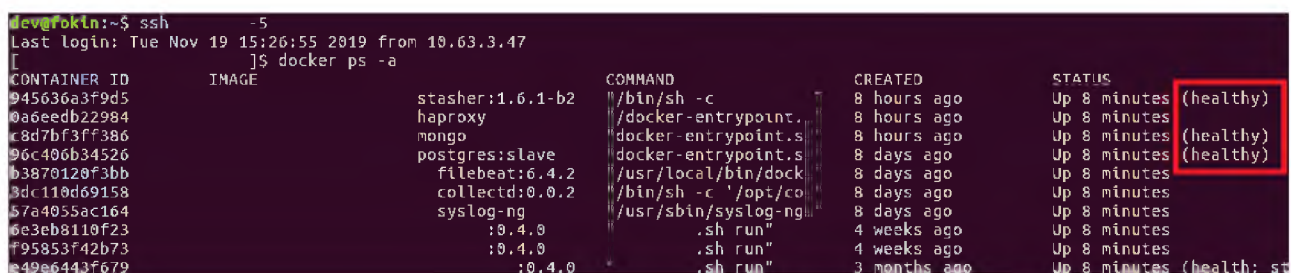
Усі сервіси на стороні “Control Server” розгорнуті в docker[36] для зручного керування, оскільки він має зручну функцію перевірки стану своїх контейнерів, так наприклад для перевірки стану бази даних mongo, яка використовується Data Analyzer сервісом, docker може відправляти запит до бази і якщо вона працює правильно отримає відповідь. Таким чином ми маємо можливість реалізувати багаторівневу систему моніторингу та відновлення сервісів на стороні контролюючого серверу. Спочатку усі сервіси контролює

docker і рішення по відновленню приймаються на базі правил, а іншим рівнем моніторингу та відновлення виступає штучний інтелект. Для реалізації перевірки стану docker контейнерів було додано в Docker compose file команди приклад яких можна побачити в таблиці 3.2.

Таблиця 3.2 Код моніторингу стану docker контейнерів

```
healthcheck:
  test: echo 'db.stats().ok' | mongo localhost:27017/main --quiet
  interval:  $t_1$ 
  timeout:  $t_2$ 
  retries:  $n$ 
```

У полі “interval”,  $t_1$  – це час через котрий Docker буде перевіряти стан контейнеру,  $t_2$  – це максимальний час відповіді контейнеру,  $n$  – це кількість повторних запитів, тобто якщо сервіс не відповів  $n$  разів підряд, docker буде вважати його “unhealthy”. Приклад роботи Docker health status наведено на рисунку 3.6.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
945636a3f9d5	stasher:1.6.1-b2	/bin/sh -c	8 hours ago	Up 8 minutes (healthy)
0a6eedb22984	haproxy	/docker-entrypoint.s	8 hours ago	Up 8 minutes (healthy)
c8d7bf3ff386	mongo	docker-entrypoint.s	8 hours ago	Up 8 minutes (healthy)
96c406b34526	postgres:slave	docker-entrypoint.s	8 days ago	Up 8 minutes
b3870120f3bb	filebeat:6.4.2	/usr/local/bin/dock	8 days ago	Up 8 minutes
3dc110d69158	collectd:0.0.2	/bin/sh -c '/opt/co	8 days ago	Up 8 minutes
57a4055ac164	syslog-ng	/usr/sbin/syslog-ng	8 days ago	Up 8 minutes
6e3eb8110f23	:0.4.0	.sh run"	4 weeks ago	Up 8 minutes
f95853f42b73	:0.4.0	.sh run"	4 weeks ago	Up 8 minutes
e49e6443f679	:0.4.0	.sh run"	3 months ago	Up 8 minutes (health: st

Рисунок 3.6 – Перевірка стану Docker контейнерів

Стан контейнерів відстежує на кожному сервері спеціальний процес для швидкого відновлення. Часто контейнери мають статус unhealthy через велику кількість запитів до сервісу або інших передбачених помилок. Цей процес працює за наступним алгоритмом, спочатку він пробує повторно створити

контейнер використовуючи спеціальні конфігурації. Фактично він пробує виконати команду “docker-compose up –d –force-recreate --no-deps service\_name”, після чого знову перевіряє стан контейнеру. Якщо крок перший проблему не вирішив, процес спробує перезавантажити сервіс Docker та виконати попередній шаг. У випадку коли після першого та другого кроків проблема залишилася, процес перезавантажує сервер. Таким чином використовуючи Docker health check ми відсіяли більшу кількість помилок та несправностей, які виникають у системі при реальному навантаженні і які легко вирішуються простими детермінованими правилами. Якщо процес, котрий контролює стан контейнерів дійшов до третього етапу відновлення, а саме перезавантаження серверу, відповідні дані відправляються до “ELK Kibana”. Приклад відображення цих даних можна побачити на рисунку 3.7

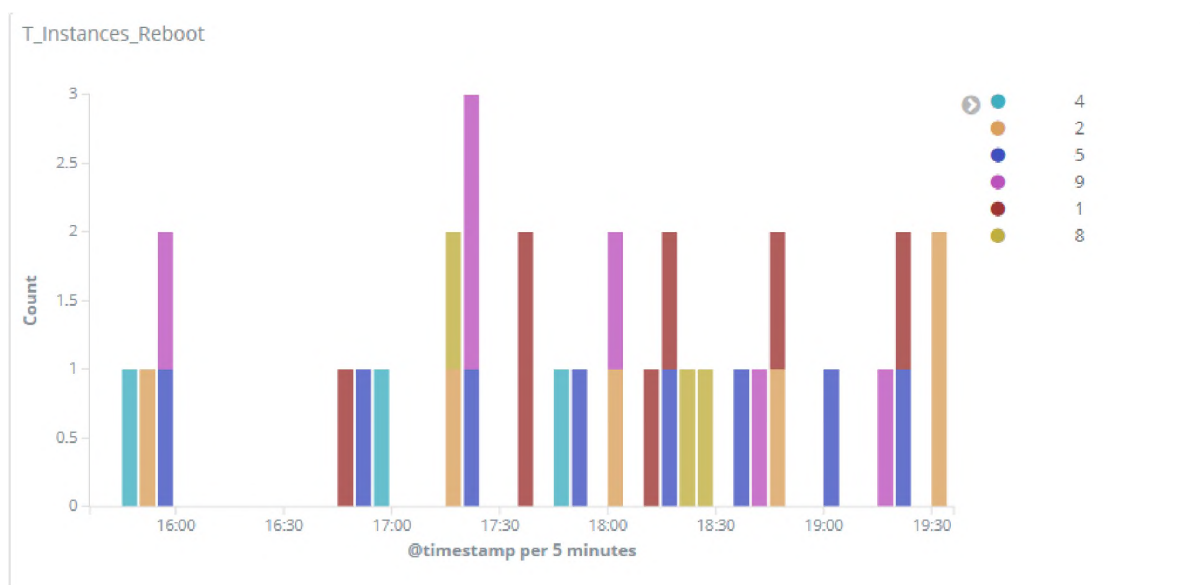


Рисунок 3.7 – Перезавантаження сервісів на базі Docker health check

Після того як дані з сенсорів записуються в базу даних вони аналізуються штучним інтелектом (класифікатором) та відображаються на “ELK Kibana”[32]. Дані типу, кількість запитів на оновлення модулів на стороні клієнта та кількість успішних оновлень, розмір бази даних, а також інші показники

сервісів на стороні “Control Server” повинні використовуватись не тільки штучним інтелектом але і відображатися на “ELK Kibana”[32] для швидкого аналізу розробником ефективності роботи системи в цілому, коректності роботи сервісів та систем моніторингу і відновлення. Модель класифікатора, яка слугує другим рівнем моніторингу та відновлення, відображена на рисунку 3.8, після чого модель відновлює сервіс на сервері, якщо вона зафіксувала його неправильну роботу.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

#Loading Data
col_names = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
# Load dataset
data = pd.read_csv("data.csv", header=None, names=col_names)
data.head(0)

#Split dataset in features and target variable
feature_cols = ['1', '5', '6', '8', '2', '3', '7']
X = data[feature_cols] # Features
y = data.label # Target variable

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test

# Create Decision Tree Classifier object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifier
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Рисунок 3.8 – Модель для класифікації стану серверу

Алгоритм Decision Tree є дуже простим та швидким алгоритмом машинного навчання, але якщо подати на вхід цієї моделі дані з сенсорів, навіть незважаючи на те що вони є попередньо нормалізовані та апроксимовані точність моделі складає приблизно 70%. Такий результат є непоганий для цієї моделі якщо враховувати те що на вхід подається величезна кількість даних, а дані для тренування збиралися в ручному режимі людиною і при розмітці вона допускала велику похибку. Але для запропонованої системи такий результат є дуже низьким, тому було прийнято рішення створити систему, яка буде

допомагати підготовлювати дані для тренування та тестування моделі, а також допомагати робити розмітку даних мінімізуючи похибку людини.

### **3.2.1 Збір даних для тренування моделі Decision Tree**

В першу чергу були проаналізовані сенсори, графіки який знаходяться на “Kibana”[32] для того щоб залишити тільки ті сенсори, які максимально точно передають стан серверу і їх буде достатньо для аналізу. Було обрано графіки, які відображають параметри та ресурси, які відповідають за виконання основної функції серверу, а саме обслуговування клієнтів. Сенсори стану бази даних (розміру), сенсор кількості прийнятих та відправлених репортів і сенсор кількості розданих оновлень сервером.

Дані з розміром бази даних приходять до системи кожні 20 хвилин, а саме останній розмір даних. Стан серверу було прийнято розділяти на 3 умовних категорії, а саме на: 1 – система працює в нормальному режимі, 2 – система працює зі зниженою ефективністю або на межі своїх можливостей але продовжує виконувати свої функції, 3 – на сервері сталася помилка. Дані розміру бази було розмічено як показано на рисунку 3.9. Система автоматичного очищення бази даних працює відмінно, тому статус 2 присвоїти не вдалося. Стан 3 не ставився в ручну, оскільки система, яка буде описана в розділі 3.2.2 зробить це в автоматичному режимі, тому всі данні отримали лейблу 1.



T\_MongoDB\_Size\_byhost

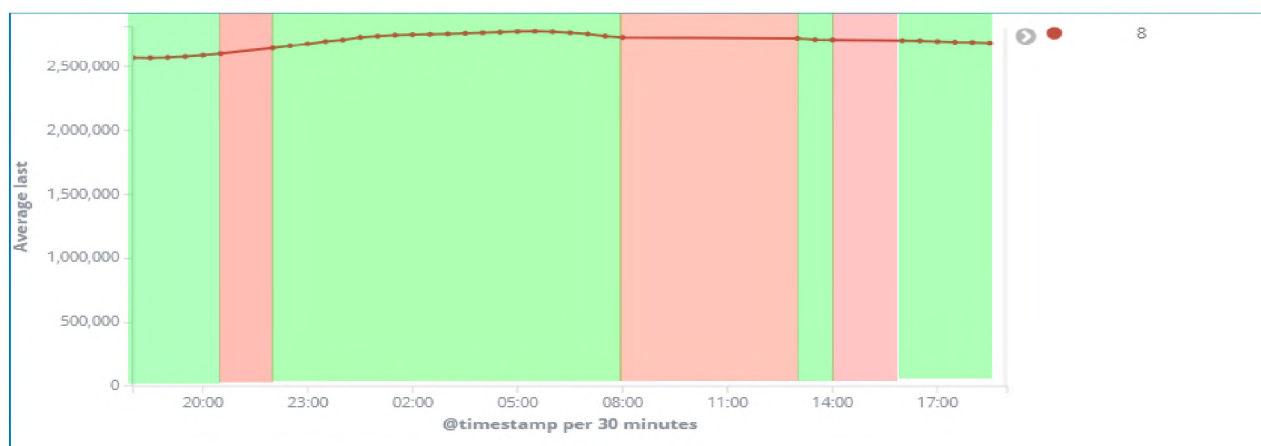


Рисунок 3.9 – Розмітка даних розміру бази даних для тренування моделі

Щодо сенсору отриманих та прийнятих репортів, вдалося розмітити всі три категорії. Категорія 3 ставилася в тому випадку, коли дані після падіння досягали нуля, це означає що сервер не виконував у цей час одну із основних функцій, а саме приймання та відправку даних. Категорія 2 було виставлено тільки для одного проміжку часу, коли сервер обробляв менше 200000 репортів.

T\_reporter\_byhost

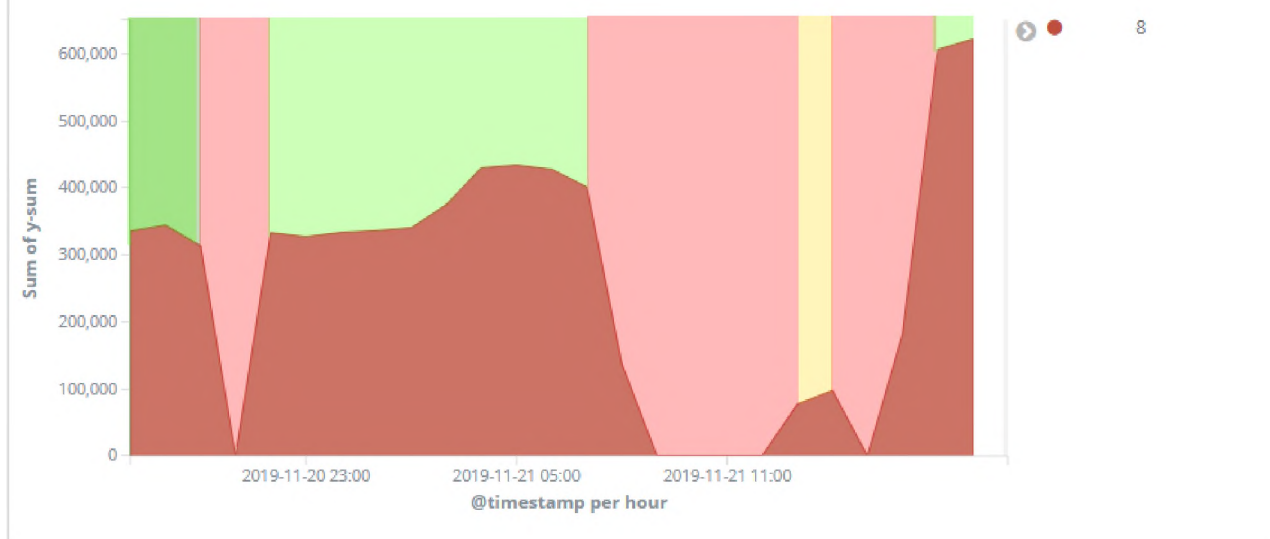


Рисунок 3.10 – Розмітка даних кількості отриманих та відправлених репортів для тренування моделі

З першого погляду графіки сенсорів “Reporter” та “Updater” схожі. Це тому що, оновлення є підмножиною репортів, але нехтувати даними по кількості розданих оновлень не можна, оскільки це основна функція і якщо сервер перестане роздавати оновлення, це не сильно відобразиться на графіку репортів.

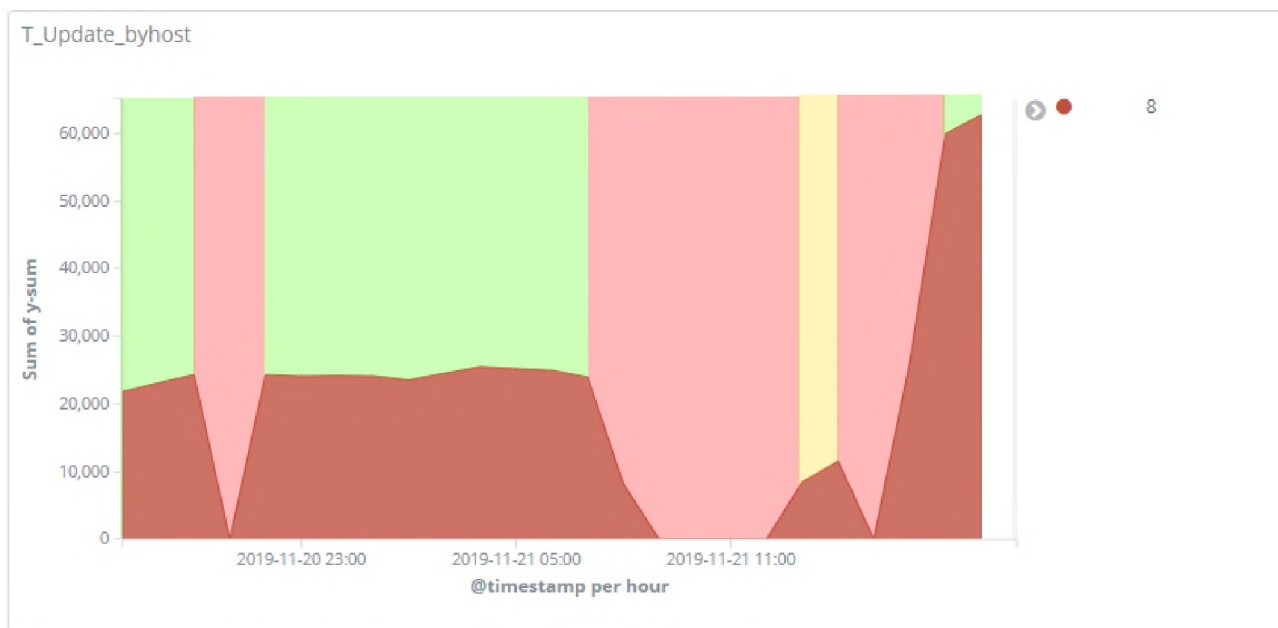


Рисунок 3.11 – Розмітка даних кількості розданих оновлень для тренування моделі

Після експортування необроблених даних з “ELK Kibana”, шляхом використання вбудованого інструменту, було отримано три файли з наборами даних, які складається з наступних полів:

- Час отримання даних розміру бази MongoDB (File\_1)
- Розмір бази (File\_1)
- Час отримання даних з кількості отриманих/ відправлених репортів (File\_2)
- Кількість отриманих/відправлених репортів (File\_2)
- Час отримання даних з кількості розданих оновлень (File\_3)

- Кількість розданих оновлень (File\_3)

Над даними проводиться розмітка, шляхом додавання нового поля в кожному файлі (status), яке приймає наступне значення:

- 1 – якщо модуль працює правильно
- 2 – якщо модуль працює неефективно, але продовжує виконувати свої функції
- 3 – якщо модуль не працює

### 3.2.2 Обробка часових рядів для Decision Tree

Після розмітки даних до роботи залучається модуль для обробки часових рядів. Він може працювати в двох основних режимах, для обробки даних для тренування та для обробки даних для тестування моделі. Модель читає свій файл конфігурації в якому присутні поля:

1. Train data: (True/False)
2. Window time size: 40
3. Shift: 2

Основна ідея цього модулю це розбиття даних на вікна за розміром, яке зазначено в конфігураційному файлі, а також синхронізація даних з різних модулів за часом. Для нашої системи було обрано 40 хвилин. Після цього обраховуються спеціальні значення і плаваюче вікно зсувається на час  $t$ , яке було обрано 2 хв.

Як було описано в пункті 3.2.2 даної роботи, на вхід до цього модулю подаються три вагові групи даних з сенсорів. Для даних зв'язних з розмірами бази, модуль робить автоматичну корекцію розмітки. Нам відомо що дані з розміром бази повинні надходити кожні 20 хвилин і тому у часовому вікні повинні бути присутні два значення. Якщо наприклад у вікно дані не потрапляють система ставить відмітку 3. Також система автоматично



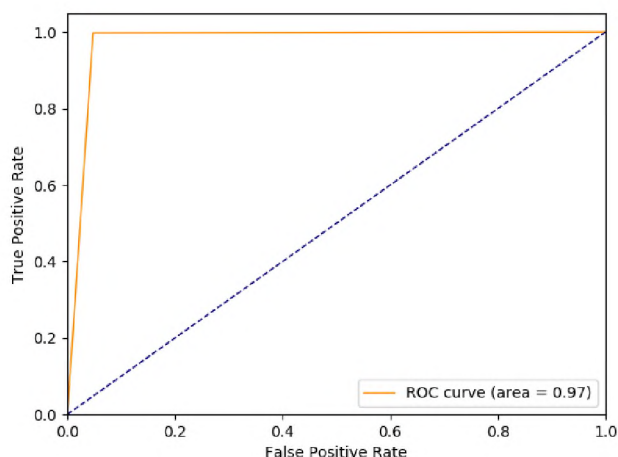


Рисунок 3.13 – ROC крива моделі класифікації

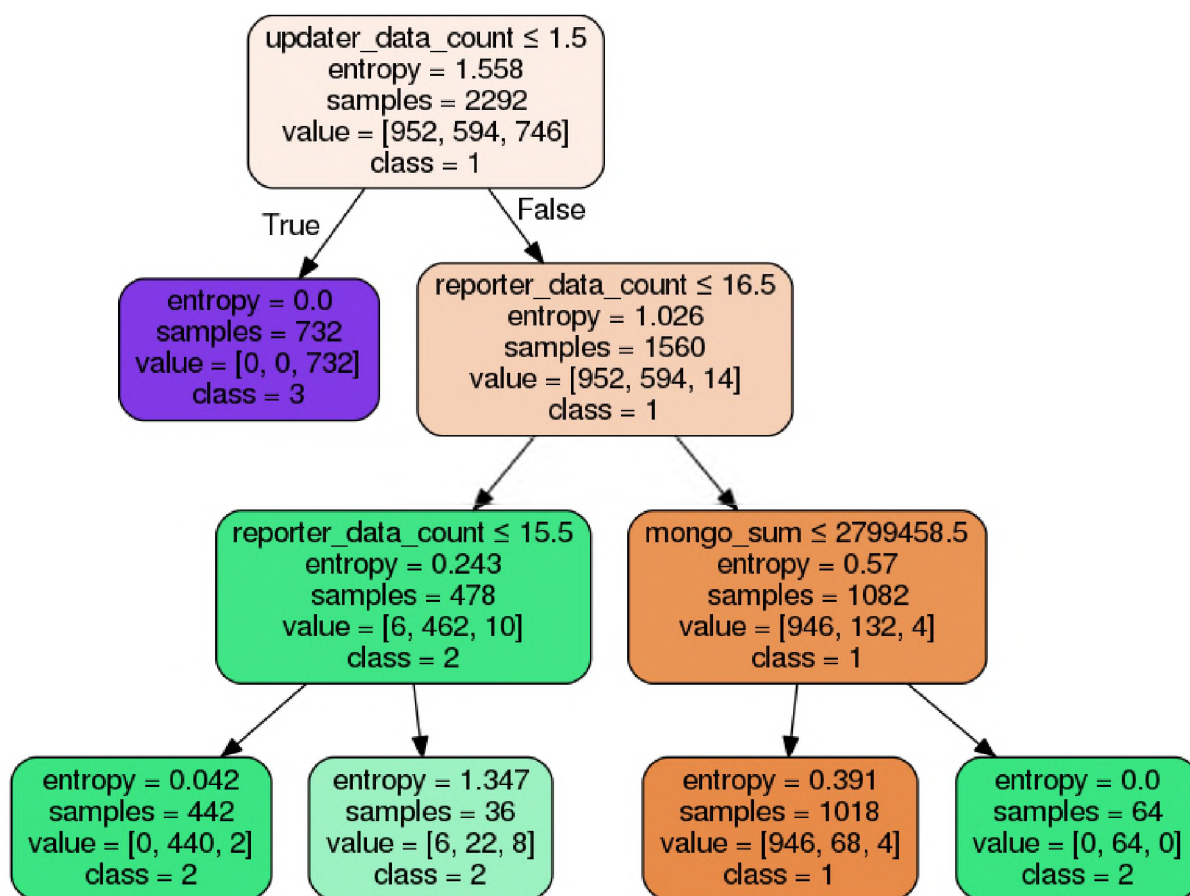


Рисунок 3.14 – Правила побудовані моделлю для класифікації стану серверу

Як видно з рисунку 3.13 модель Decision Tree в першу аналізує кількість відправлених оновлень і якщо ця кількість менша за визначене значення,

приймає рішення, що система в поганому стані. Якщо ні, перевіряє кількість отриманих / відправлених репортів і якщо ця кількість менша за визначене значення, приймає рішення, що стан системи середній. Якщо кількість отриманих / відправлених репортів більша за визначене значення, класифікатор перевіряє розмір бази даних і якщо вона менша за визначене значення, приймає рішення, що стан системи нормальний. В іншому випадку стан системи визначається як середній.

### **Висновки до розділу 3**

Проаналізовано і побудовано систему з властивостями самовідновлення, яка забезпечує нормативні вимоги безпеки, а також підвищує характеристики захищеності системи, шляхом підтримки модулів, які захищають систему від кіберзагроз на стороні клієнта і серверів обслуговування. Запропонована система має властивості самовідновлення, які були визначені у першому та другому розділі даної роботи, оскільки вона має функцію моніторингу компонентів системи, може приймати рішення по їх відновленню, сповіщає розробників при виявленні проблем у компонентах.

Самовідновлення на стороні клієнта було реалізовано шляхом поєднання методів перевірки стану об'єктів системи TTL та метод пінгування. Рішення про відновлення модулів на стороні клієнта приймається з використанням власного алгоритму на базі детермінованих правил "Healalgorithm", а також розподіл клієнтів на групи, що забезпечило відновлення системи на стороні клієнта без втрати функціоналу. Всі дані, які передаються між клієнтом та сервером стискаються алгоритмом RLE[31] для економії трафіку.

Реалізовано багаторівневу систему відновлення, завдяки використанню Docker health check і детермінованих правил, а також алгоритм для класифікації

стану серверів Decision Tree. Точність алгоритму машинного навчання Decision Tree склала 97% завдяки спеціальній обробці даних перед поданням їх на вхід моделі. Обробка часових рядів виконується за принципом “часового вікна”, а також апроксимація даних з сенсорів за вагами.

## 4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

Розділ має на меті проведення маркетингового аналізу стартап проекту задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження.

### 4.1 Опис ідеї проекту

В цьому підрозділі аналізується зміст ідеї, що пропонується, можливі напрямки застосування, основні вигоди, що може отримати користувач товару (за кожним напрямком застосування) (таблиця 4.1) та відмінність від існуючих аналогів та замінників (таблиця 4.2)

Таблиця 4.1 – Опис ідеї стартап-проекту

<i>Зміст ідеї</i>	<i>Напрямки застосування</i>	<i>Вигоди для користувача</i>
Архітектура багаторівневої системи самовідновлення для розподілених систем на базі правил та штучного інтелекту	1. ІТ компанії, що займаються розробкою ПЗ	1. Швидкість детектування неправильної поведінки одного з компонентів системи як на стороні клієнта так і на стороні компанії, яка надає послуги



Кінець таблиці 4.1

<i>Зміст ідеї</i>	<i>Напрямки застосування</i>	<i>Вигоди для користувача</i>
	2. ІТ компанії, що займаються обслуговуванням великої кількості клієнтів	2. Швидкість відновлення компонентів системи після збоїв
		3. Легкість в розгортанні системи
		4. Низькі потреби до продуктивності обладнання на якому дана система розгортається

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

<i>n</i> / <i>n</i>	<i>Техніко- економічні характерис тики ідеї</i>	<i>(потенційні) товари/концепції конкурентів</i>			<i>W (слабка сторона)</i>	<i>N (нейтр альна сторон а)</i>	<i>S (сильна сторон а)</i>
		<i>Мій проект</i>	<i>Google</i>	<i>Facebook</i>			
1	Швидкість виявлення неправильної роботи компонент у системі на стороні клієнта	Не залежить від кількості компонентів системи на стороні клієнта	Не залежить від кількості компонентів системи на стороні клієнта	Залежить від кількості клієнтів	Дуже залежить від кількості компонентів системи	Лінійна залежність	Виявлення у реальному часі
2	Відновлення у реальному часі неправильної роботи компонент у системі на стороні клієнта	S = 89% N = 8% W = 3%	S = 10% N = 80% W = 10%	S = 8% N = 90% W = 2%	Автоматичне відновлення неможливе і потребує втручання розробника для генерації оновлення	Жертвування функціоналом системи	Повне відновлення системи без втрати функціоналу

Кінець таблиці 4.2

<i>n</i> / <i>n</i>	<i>Техніко- економічні характери- стики ідеї</i>	<i>(потенційні) товари/концепції конкурентів</i>			<i>W (слабка сторона)</i>	<i>N (нейтр альна сторон а)</i>	<i>S (сильна сторон а)</i>
		<i>Мій проект</i>	<i>Google</i>	<i>Faceboo k</i>			
3	Швидкість виявлення неправиль- ної роботи компонент у системи на стороні сервера	Не залежит ь від кількост і компоне нтів системи на сервері	Не залежить від кількості компонен тів системи на сервері	Не залежит ь від кількост і компоне нтів системи на сервері	Залежить від кількості компонен тів системи	лінійна залежн ість	виявле ння у реальн ому часі
4	Відновлен ня у реальному часі неправиль- ної роботи компонент у системи на стороні сервера	S = 80% N = 0% W = 20%	S = 70% N = <1% W = 29%	S = 63% N = 1% W = 36%	Автомати чне відновлен ня неможлив е і потребує втручання розробник ах	Жертв ування функці оналом систем и	Повне віднов лення систем и без втрати функці оналу

## 4.2 Технологічний аудит ідеї проекту

В межах даного підрозділу був проведений аудит технології, за допомогою якої можна реалізувати ідею проекту та заповнена таблиця 4.3

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

<i>n/ n</i>	<i>Ідея проекту</i>	<i>Технології її реалізації</i>	<i>Наявність технологій</i>	<i>Доступність технологій</i>
1	Поєднання підходів TTL та методу перевірки доступності та їх використання на стороні клієнта	Використання TTL для перевірки стану модулів на стороні клієнта, використання методу перевірки доступності для демонів	TTL і перевірка доступності	TTL і перевірка доступності у вигляді псевдокоду доступні у відкритому доступі
2	Відновлення модулів на стороні клієнта за детермінованими правилами	Власна розробка алгоритму та його тестування	Відсутня і потребує власної розробки	Недоступна потребує розробки

Кінець таблиці 4.3

<i>n/ n</i>	<i>Ідея проекту</i>	<i>Технології її реалізації</i>	<i>Наявність технологій</i>	<i>Доступність технологій</i>
3	Моніторинг компонентів на стороні серверів	Використання сучасного програмного забезпечення та штучного інтелекту	Сервіси повинні розвернуті в окремих контейнерах, сервіси для збору даних налаштовані, нейронна мережа повинна бути побудована та натренована	Docker, filebeat, collectd, syslog-ng, Riemann [33], фреймворки tenserflow, keras знаходяться в відкритому доступі. ELK Kibana є пробний термін.
4	Відновлення компонентів системи на стороні сервера	Відновлення за допомогою детермінованих правил та штучного інтелекту	Детерміновані правила повинні бути реалізовані, нейронна мережа повинна бути побудована та натренована	Docker, tenserflow, keras знаходяться в відкритому доступі.
Обрана технологія реалізації ідеї проекту: так як для реалізації ідеї проекту, не всі технології є наявними та доступними, тому обираються всі вище описані технології, а недоступні технології повинні бути реалізовані самостійно.				

### 4.3 Аналіз ринкових можливостей запуску стартап-проекту

В межах данного підрозділу було визначено ринкові можливості які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів (таблиця 4.4, таблиця 4.5).

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

<i>n/ n</i>	<i>Показники стану ринку (найменування)</i>	<i>Характеристика</i>
1	Кількість головних гравців, од	більше 2
2	Загальний обсяг продаж, грн/ум.од	500 млн ум. од.
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Потреба у серверах
5	Специфічні вимоги до стандартизації та сертифікації	Не потребує стандартизації та сертифікації

6	Середня норма рентабельності в галузі (або по ринку), %	Не менше 500
---	--	--------------

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

<i>Потреба, що формує ринок</i>	<i>Цільова аудиторія (цільові сегменти ринку)</i>	<i>Відмінності у поведінці різних потенційних цільових груп клієнтів</i>	<i>Вимоги споживачів до товару</i>
Виявлення неправильної роботи та відновлення компонентів програмного забезпечення, що можуть призвести до відмови в обслуговуванні	1. ІТ компанії, що займаються розробкою ПЗ 2. ІТ компанії, що займаються обслуговуванням великої кількості клієнтів	Для кожної з категорій існують окремі цінності пропозицій. Пропозиція відрізняється для кожної цільової групи.	<ul style="list-style-type: none"> <li>• швидкість виявлення та відновлення компонентів системи</li> <li>• точність детектування неправильної роботи КС</li> <li>• відновлення системи без втрати функціоналу</li> <li>• мінімальне використання продуктом ресурсів системи</li> </ul>

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (таблиця 4.6-4.7)

Таблиця 4.6 – Фактори загроз

<i>№ n/n</i>	<i>Фактор</i>	<i>Зміст загрози</i>	<i>Можлива реакція компанії</i>
1	Конкуренція	Деякі продукти стрімко покращують свої показники використовуючи нові технології	Покращення точності виявлення неправильної поведінки КС та методів відновлення
2	Непередбачені помилки в програмному забезпеченні	Можлива наявність помилок при розробці ПЗ для моніторингу та відновлення КС на стороні клієнта та сервері обслуговування, які не були виявлені при тестуванні	Покращення ПЗ шляхом виправлення знайдених помилок
3	Використання програмного забезпечення інших розробників	Можлива наявність прихованих вразливостей в сервісах від сторонніх розробників	Розробка власного ПЗ
4	Проблеми з'єднання	Проблеми зв'язку між клієнтом на сервером, а також	Реалізація автономного відновлення без втрати функціоналу в разі проблем



		між серверами унеможлиблюють відновлення КС	із з'єднанням
--	--	---	---------------

Таблиця 4.7 – Фактори можливостей

<i>№ n/n</i>	<i>Фактор</i>	<i>Зміст можливості</i>	<i>Можлива реакція компанії</i>
1	Поява нових топологій нейронних мереж та методів моніторингу КС	Підвищення результатів точності виявлення та часу роботи для класифікації	Впровадження нової топології нейронної мережі для класифікації, розширення функціоналу, проведення маркетингової компанії
2	Поява нових методів відновлення КС	Підвищення результатів відновлення КС	Впровадження нової технології, розширення функціоналу, проведення маркетингової компанії
3	Реалізація запропонованої архітектури не тільки для телевізорів на ОС Tizen але і для інших ОС (кросплатформеність)	Збільшення кількості потенційних клієнтів	Реалізація кросплатформеності шляхом створення спеціального API
4	Залучення нових відомих компаній у якості клієнтів чи постачальників	Підвищення фінансування	Підвищення якості програмного коду, оптимізації, шляхом найму додаткових

			спеціалістів
--	--	--	--------------

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (таблиця 4.8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

<i>Особливості конкурентного середовища</i>	<i>В чому проявляється дана характеристика</i>	<i>Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)</i>
Олігополістична конкуренція	Розробка розподіленої системи з властивістю самовідновлення є дуже дорогим і складним, потребує великого фінансування, а також кількості розробників, тому існуючі рішення є тільки в великих ІТ компаніях	Вдосконалення рішення для підвищення якості обслуговування клієнтів, розширювати продукт на інші платформи та гаджети
За рівнем конкурентної боротьби: національний	Надання послуг для клієнтів в Україні та за її межами	Застосування новітніх технологій для підвищення конкурентоспроможності
За галузевою ознакою: внутрішньогалузева	Рішення можуть використовуватися тільки компаніям, що займаються розробкою ПЗ або надають послуги клієнтам і повинні гарантувати доступність	Застосування маркетингових методів, які дозволять користуватися продуктом компаніям

Конкуренція за видами товарів: товарно-видова	Рішення що допомагає обслуговувати клієнтів, відрізняється від інших продуктів використаними технологіями	Надання послуг з моніторингу та відновлення компонентів системи
---	---	---

Кінець таблиці 4.8

<i>Особливості конкурентного середовища</i>	<i>В чому проявляється дана характеристика</i>	<i>Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)</i>
За характером конкурентних переваг: цінова	Аналогічні продукти використовуються тільки великими компаніями і не розповсюджуються іншим особам	Перенесення системи на інші платформи та створення можливості приєднання до системи по підписці
За інтенсивністю: марочна	Сукупність характеристик та властивостей рішення	Підвищення надійності програмного забезпечення клієнтів

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (за моделлю 5 сил М. Портера) (таблиця 4.9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

<i>Складові аналізу</i>	<i>Прямі конкуренти в галузі</i>	<i>Потенційні конкуренти</i>	<i>Постачальники</i>	<i>Клієнти</i>	<i>Товари - замітники</i>
	<i>Немає</i>	<i>Google Facebook</i>	<i>Великі компанії, які піклуються о доступності своїх послуг</i>	<i>ІТ компанії, які займаються розробкою ПЗ, яка орієнтована на велику кількість клієнтів</i>	<i>Товари замітники відсутні</i>

Кінець таблиці 4.9

<i>Складові аналізу</i>	<i>Прямі конкуренти в галузі</i>	<i>Постачальники</i>	<i>Клієнти</i>	<i>Товари-замінники</i>
Висновки:	На сьогоднішній час технології самовідновлення конкуренти використовують тільки для підтримки власних сервісів та систем	Оскільки системи конкурентів є складними і поки що не реалізована можливість розгортання їх на інші системи	Основними клієнтами можуть бути ІТ компанії, які займаються підтримкою своїх продуктів на стороні клієнта	Обмежень на ринку через товари замітники немає

На основі проведеної роботи проводиться обґрунтування факторів конкурентоспроможності (таблиця 4.10)

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

<i>№ n/n</i>	<i>Фактор конкурентоспроможності</i>	<i>Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)</i>
	Якість продукту	Продукт був протестований і показує відмінні результати по моніторингу та відновленню системи

Кінець таблиці 4.10

<i>№ n/n</i>	<i>Фактор конкурентоспроможності</i>	<i>Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)</i>
	Рівень ціни	Оскільки систему самовідновлення для розподіленої системи на сьогоднішній день придбати неможливо, а існуючі системи використовуються компаніями у власних цілях, ціна на продукт може бути середньою
	Наявність конкурентів	Аналогічні системи розроблені та використовуються конкурентами у власних цілях. На сьогоднішній день використовувати або придбати систему конкурентів немає можливості
	Масштабованість продукту	Продукт потребує масштабованості

За визначеними факторами конкурентоспроможності (таблиця 4.10) проводиться аналіз сильних та слабких сторін стартап-проекту (таблиця 4.11).

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін

<i>n/ n</i>	<i>Фактор конкурентоспроможності</i>	<i>Бали 1-20</i>	<i>Рейтинг товарів-конкурентів у порівнянні з Google, Facebook</i>						
			<i>-3</i>	<i>-2</i>	<i>-1</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>
	Якість продукту	20		+					
	Рівень ціни	10				+			
	Масштабованість продукту	10			+				

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 4.12)

Таблиця 4.12 – SWOT- аналіз стартап-проекту

Сильні сторони: реалізація багаторівневої системи моніторингу та відновлення КС, використовуючи сучасні алгоритми машинного навчання, а також власні алгоритми на базі детермінованих правил	Слабкі сторони: Підтримка тільки Tizen OS, складність розгортання продукту на інших системах
Можливості: залучення великих ІТ компаній в якості постачальників чи клієнтів	Загрози: реалізація можливості перенесення продуктів конкурентів на інші системи

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації (таблиця 4.13)

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

<i>№ п/п</i>	<i>Альтернатива (орієнтовний комплекс заходів) ринкової поведінки</i>	<i>Ймовірність отримання ресурсів</i>	<i>Строки реалізації</i>
	Створення угоди з певною ІТ компанією	Висока ймовірність отримання ресурсів	до 11 місяців
	Впровадження рекламних засобів	Середня ймовірність отримання ресурсів	до 4 місяців
	Участь в ІТ виставках, публікація в журналу	Середня ймовірність отримання ресурсів	до 3 місяців
	Організування патенту	Висока ймовірність отримання ресурсів	до 6 місяців
	Додання до	Мала ймовірність	до 7 місяців

	функціоналу специфічних функцій	отримання ресурсів	
--	------------------------------------	--------------------	--

#### 4.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (таблиця 4.14)

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

<i>№ п/ п</i>	<i>Опис профілю цільової групи потенційних клієнтів</i>	<i>Готовність споживачів сприйняти продукт</i>	<i>Орієнтовний попит в межах цільової групи (сегменту)</i>	<i>Інтенсивність конкуренції в сегменті</i>	<i>Простота входу у сегмент</i>
	1. ІТ компанії, що займаються розробкою ПЗ	Клієнти потребують продукт такого типу та готові ним користувати сь	Високий рівень попиту	Немає конкуренції	Особої складності входу немає але деякі великі компанії мають власнорозробл ені продукти
	2. Компанії, що займаються обслуговуванн	Клієнти потребують продукт	Високий рівень попиту		Немає складності входу



	ям клієнтів	такого типу та готові ним користувати сь			
Які цільові групи обрано: вибрані групи з високим рівнем попиту, а саме ІТ компанії, що займаються розробкою ПЗ та компанії, що займаються обслуговуванням клієнтів					

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку (таблиця 4.15).

Таблиця 4.15 – Визначення базової стратегії розвитку

<i>n/ n</i>	<i>Обрана альтернатива розвитку проекту</i>	<i>Стратегія охоплення ринку</i>	<i>Ключові конкурентоспромо жні позиції відповідно до обраної альтернативи</i>	<i>Базова стратегія розвитку*</i>
	Стратегія лідера	Стратегія концентрован ого маркетингу	Розповсюдження технології на інші платформи, підвищення мобільності системи	Стратегія лідерства по витратах

Наступним кроком є вибір стратегії конкурентної поведінки (таблиця 4.16)

Таблиця 4.16 – Наступним кроком є вибір стратегії конкурентної поведінки

<i>№ п/п</i>	<i>Чи є проект «першопрохідцем» на ринку?</i>	<i>Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?</i>	<i>Чи буде компанія копіювати основні характеристики товару конкурента, і які?</i>	<i>Стратегія конкурентної поведінки*</i>
	Так	Так	Так	Стратегія лідера

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (див. таблицю 4.5), а також в залежності від обраної базової стратегії розвитку (таблиця 4.15) та стратегії конкурентної поведінки (таблиця 4.16) розробляється стратегія позиціонування (таблиця 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 – Визначення стратегії позиціонування

<i>Вимоги до товару цільової аудиторії</i>	<i>Базова стратегія розвитку</i>	<i>Ключові конкурентоспро- можні позиції власного стартап-проекту</i>	<i>Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)</i>
Легка інтеграція з іншими системами, можливість розгортання системи на різних ОС, низьке використання ресурсів продуктом, висока точність виявлення	Стратегія лідерства по витратах	Розповсюдження технології на інші платформи, підвищення мобільності системи	Точність, швидкість, мобільність, якість

неправильної роботи модулів та відновлення системи без втрати функціоналу			
--	--	--	--

#### 4.5 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у таблиці 4.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

<i>№ п/п</i>	<i>Потреба</i>	<i>Вигода, яку пропонує товар</i>	<i>Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)</i>
1	Виявлення неправильної роботи системних модулів	Багаторівнева розподілена система моніторингу з високою точністю виявлення неправильної	Багаторівневність системи, можливість виявляти помилки на платформі Tizen, використання алгоритмів машинного навчання

		роботи модулів	
--	--	----------------	--

Кінець таблиці 4.18

<i>№ n/n</i>	<i>Потреба</i>	<i>Вигода, яку пропонує товар</i>	<i>Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)</i>
2	Відновлення модулів в разі збою	Відновлення компонентів розподіленої системи без втрати функціоналу в реальному часі на стороні серверу та з невеликою затримкою на стороні клієнта	Розподіл клієнтів на групи для більш ефективного моніторингу та відновлення, використання спеціально розроблених механізмів відновлення без втрати функціоналу
3	Час між детектуванням неправильної роботи модуля	Відновлення виконується з мінімальною затримкою	Використавши розроблених алгоритм Healthgoritm, відновлення модулів на стороні клієнта виконується дуже швидко. На

	та відновленням		стороні серверу відновлення відбувається в реальному часі при використання технологій машинного навчання та docker
--	--------------------	--	---

Надалі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (таблиця 4.19)

Таблиця 4.19 – Опис трьох рівнів моделі товару

<i>Рівні товару</i>	<i>Сутність та складові</i>		
I. Товар за задумом	Розподілена система з властивостями самовідновлення для обслуговування клієнтів		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Використовує поєднання підходів TTL та перевірку доступності на стороні клієнта	М	Тх
		М	Тх
	2. Використовує розроблений алгоритм для відновлення модулів на стороні клієнта	М	Тх
		М	Тх
	3. Політика розбивання клієнтів на групи для більш ефективної роботи системи		
	4. Використання Docker для моніторингу та відновлення компонентів системи на стороні серверу		

	5. Використання алгоритмів машинного навчання для моніторингу та відновлення системи на стороні серверу	М	ТХ
	Якість: забезпечення критерія доступності		
	Пакування: Послуга постачається разом з технікою		
III. Товар із підкріпленням	До продажу: початок рекламної компанії		
	Після продажу: продовження рекламної компанії		
Товар буде розповсюджуватись разом з приладами, які дана система підтримує			

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субституту, а також аналіз рівня доходів цільової групи споживачів (таблиця 4.20). Аналіз проводиться експертним методом.

Таблиця 4.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари- замінники	Рівень цін на товари- аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
	-	Немає даних	12000	Послуга (Безкоштовна)  Товар 5000-500000 залежно від класу товару

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (таблиця 4.21)

Таблиця 4.21 – Формування системи збуту

<i>№ п/п</i>	<i>Специфіка закупівельної поведінки цільових клієнтів</i>	<i>Функції збуту, які має виконувати постачальник товару</i>	<i>Глибина каналу збуту</i>	<i>Оптимальна система збуту</i>
	мінімальна кількість посередників	організувати широку мережу збуту товару	1	не пряма

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 4.22).

Таблиця 4.22 – Концепція маркетингових комунікацій

<i>№ п/п</i>	<i>Специфіка поведінки цільових клієнтів</i>	<i>Канали комунікацій, якими користуютьс я цільові клієнти</i>	<i>Ключові позиції, обрані для позиціонування</i>	<i>Завдання рекламного повідомленн я</i>	<i>Концепція рекламног о звернення</i>
	Моніторинг ефективності запропонован ої системи, впровадження новітніх технологій	Інформаційні канали	Розподілена система з властивостями самовідновленн я для обслуговування клієнтів	Донести переваги та доступність до можливих клієнтів	



для покращення показників продукту. Розширення продукту на інші платформи					
--	--	--	--	--	--

#### Висновки до розділу 4

Було виконано аналіз стартап проекту для визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження. Проаналізовані сильні та слабкі сторони проекту в порівнянні з конкурентами. Стартап проект має великий потенціал, оскільки має кращі результати ніж конкуренти на ринку, а також підтримує платформи, які не підтримують інші компанії. Для проекту наявні обмеження для входу на ринок, оскільки система потребує підключення серверів обслуговування, а також використовує інструменти, яких немає у відкритому доступі, які не мають своїх безкоштовних аналогів.

Розроблена ринкова стратегія, яка передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів. Основними споживачами продукту виявились компанії, які займаються розробкою програмного забезпечення, а також великі ІТ компанії, які займаються обслуговуванням клієнтів. Були проаналізовані базові стратегії розвитку і обрані стратегія лідерства, як альтернатива розвитку проекту і стратегія лідерства по витратах, оскільки на ринку відсутні прямі конкуренти. Були

сформовані маркетингові концепції товару, який отримає споживач, а також інші маркетингові програми стартап-проекту.

Базуючись на отриманих результатах, стартап-проект має потенціал виходу на ринок.

## ВИСНОВКИ

Складність існуючих систем зростає з кожним днем, концепції багатомодульності та мікросервісів призводять до того, що виникає необхідність впроваджувати механізми моніторингу, автоматичного визначення помилок та відновлення нормального режиму роботи системи. Саме тому розробка нових методів моніторингу та відновлення є актуальною.

У даній роботі було розглянуто функції систем самовідновлення для підтримки кібербезпеки і забезпечення цими системами нормативних вимог безпеки для розподілених і нерозподілених систем. Проаналізовано існуючі підходи для моніторингу та відновлення компонентів системи, їх переваги та недоліки. Експериментальним шляхом доведено недосконалість сучасних систем самовідновлення, на прикладі операційних систем Android та Linux. А також розглянуто приклад розподіленої системи з властивістю самовідновлення MongoDB з використанням AWS сервісів.

Виконано аналіз рекомендацій по реалізації системи з властивістю самовідновлення, а також методи детектування неправильної роботи модулів з використанням штучного інтелекту. Було проаналізовано вимоги до системи обслуговування клієнтів з властивістю самовідновлення і на базі цього аналізу було створено її архітектуру.

Дана робота містить реалізацію запропонованої системи. Як результат, вдалося побудувати багаторівневу систему самовідновлення, для більш надійного і ефективного відновлення у разі збою одного із компонентів системи, або групи КС. На стороні клієнта реалізовано моніторинг КС шляхом поєднання методів TTL та пінгування і розроблено алгоритм на базі детермінованих правил для швидкого і надійного відновлення КС. Всі команди, які передаються між клієнтом на сервером, який контролює стан об'єктів на стороні клієнта, було прийнято стискати алгоритмом RLE. Моніторинг на стороні серверу було реалізовано шляхом використання технології Docker

health check у поєднанні з детермінованими правилами як другий рівень моніторингу та відновлення. Як третій рівень моніторингу та відновлення системи, було обрано Decision Tree у поєднанні з власним алгоритмом обробки даних використовуючи часові вікна. Як результат, вдалося виявляти неправильну роботу серверів обслуговування використовуючи Decision Tree з точністю приблизно 97%.

Розроблена система була проаналізована як стартап проект для визначення принципової можливості його ринкового впровадження. Базуючись на отриманих результатах, стартап-проект має потенціал виходу на ринок.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1     João Paulo Ferreira de Magalhães. Self-healing Techniques for Web-based Applications [Електронний ресурс]. — 2013. — Режим доступу: <https://pdfs.semanticscholar.org/f1b3/57e9654622ec35c8d6a0cdfd4a7c39f096cd.pdf>
- 2     Sinan Kordemir. Self Healing Systems [Електронний ресурс]. — 2013 — Режим доступу: <https://www.slideshare.net/comengsin/self-healingsystems>
- 3     Victor Farcik. The DevOps 2.0 Automating the Continuous Deployment Pipeline with Containerized Microservices [Електронний ресурс]. — Режим доступу: <https://leanpub.com/the-devops-2-toolkit>
- 4     Kramer J, Magee J. Self-managed systems: an architectural challenge. Future of Software Engineering [Електронний ресурс]. — 2007. — Режим доступу: [https://www.researchgate.net/publication/4250873\\_Self\\_Managed\\_Systems\\_an\\_Architectural\\_Challenge](https://www.researchgate.net/publication/4250873_Self_Managed_Systems_an_Architectural_Challenge)
- 5     Horn P. Autonomic computing: IBM's perspective on the state of information technology [Електронний ресурс]. — 2001. — Режим доступу: <https://www.semanticscholar.org/paper/Autonomic-Computing%3A-IBM's-Perspective-on-the-State-Horn/1ad1c619a9b3ba5a3ac597f51c8d15011a83423b>
- 6     Pernici B. Self-healing systems and web services: The ws-diamond approach [Текст] / Business Process Management Workshops / Lecture Notes in Business Information Processing / том 17 // Springer Berlin Heidelberg. — 2009. —; С. 440 –442.
- 7     Debanjan Ghosh. Self-healing systems — survey and synthesis [Електронний ресурс]. — 2013 — Режим доступу:

- [https://www.researchgate.net/publication/222434671\\_Self-healing\\_systems\\_-\\_survey\\_and\\_synthesis](https://www.researchgate.net/publication/222434671_Self-healing_systems_-_survey_and_synthesis)
- 8 Microsoft. Design for self-healing [Електронний ресурс]. — 2018 — Режим доступу: <https://docs.microsoft.com/en-us/azure/architecture/guide/design-principles/self-healing>
  - 9 Департамент спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України. Критерії оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу [Електронний ресурс]. — 1999 — Режим доступу: <http://www.dsszzi.gov.ua/dsszzi/doccatalog/document?id=106342>
  - 10 Ubuntu [Електронний ресурс]. — 2019 — Режим доступу: <https://ubuntu.com/>
  - 11 Google. Android Open Source Project [Електронний ресурс]. — 2019 — Режим доступу: <https://source.android.com/setup/build/downloading>
  - 12 Mongo. How to create a MongoDB ReplicaSet with Self Healing using AWS services [Електронний ресурс]. — 2019 — Режим доступу: <https://medium.com/proud2becloud/how-to-create-a-mongodb-replicaset-with-self-healing-using-aws-services-c6da84f4fce3>
  - 13 Abhishek Bhavsar, Ameya More. A Holistic Approach to Autonomic Self-Healing Distributed Computing System [Електронний ресурс]. — 2013 — Режим доступу: <https://106.125.46.186/pages/viewpage.action?pageId=32137224>
  - 14 R. K. Sahoo. Critical event prediction for proactive management in large-scale computer cluster [Електронний ресурс]. — 2003 — Режим доступу: <http://doi.acm.org/10.1145/956750.956799>
  - 15 S. B. Aher. Comparative study of association rule algorithms for course recommender system in e-learning [Текст] / S. B. Aher, L. L.M.R.J // Inter-national Journal of Computer Applications, vol. 39, no. 1 — 2012 — С. 48-52.

- 16 R. Agrawal. Fast algorithms for mining association rules [Текст] / R. Agrawal, R. Srikan. // In Proc. of 20th Intl. Conf. on VLDB — 1994 — С. 487-499.
- 17 Daniel Ramotsoela. A Survey of Anomaly Detection in Industrial Wireless Sensor Networks with Critical Water System Infrastructure as a Case Study [Электронный ресурс]. — 2018 — Режим доступа: <https://www.mdpi.com/1424-8220/18/8/2491>
- 18 Tolga Ergen. Unsupervised and Semi-supervised Anomaly Detection with LSTM Neural Networks [Электронный ресурс]. — 2017 — Режим доступа: <https://arxiv.org/abs/1710.09207>
- 19 Khaled Alrawashdeh. Toward an online anomaly intrusion detection system based on deep learning [Текст] / Khaled Alrawashdeh, Carla Purdy // In Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference. — 2016. — С. 195-200.
- 20 Raghavendra Chalapathy. Deep learning for anomaly detection: a survey [Электронный ресурс]. — 2017 — Режим доступа: <https://arxiv.org/abs/1901.03407>
- 21 Riadh Ben Halima. Providing Predictive Self-Healing for Web Services: AQoS Monitoring and Analysis-based Approach [Электронный ресурс]. — 2009 — Режим доступа: <https://hal.archives-ouvertes.fr/halshs-00367395>
- 22 Prashant Gupta. Decision Trees in Machine Learning Approach [Электронный ресурс]. — 2017 — Режим доступа: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>
- 23 Decision tree [Электронный ресурс]. — 2019 — Режим доступа: <https://learnmachinelearning.wikia.org/ru/wiki/DecisionTree>
- 24 Jonathan Long. Fully convolutional networks for semantic segmentation [Текст] / Jonathan Long, Evan Shelhamer, Trevor Darrell // In Proceedings

- of the IEEE conference on computer vision and pattern recognition – 2015. -C. 3431-3440.
- 25 Yi Zheng. Time series classification using multi-channels deep convolutional neural networks [Текст] / Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, J Leon Zhao. // In International Conference on Web-Age Information Management – 2014. -C. 298-310.
  - 26 Wei Yao. Pixel-wise regression using u-net and its application on pansharpening [Текст] / Wei Yao, Zhigang Zeng, Cheng Lian, Huiming Tang. // Neurocomputing – 2018. -C. 312-371.
  - 27 Tailai Wen, Roy Keyes. Time Series Anomaly Detection Using Convolutional Neural Networks and Transfer Learning [Электронный ресурс]. — 2019 — Режим доступа: [https://www.zurich.ibm.com/AI4IoT/AI4IoT-19\\_Wen.pdf](https://www.zurich.ibm.com/AI4IoT/AI4IoT-19_Wen.pdf)
  - 28 Onel Harrison. Machine Learning Basics with the K-Nearest Neighbors Algorithm [Электронный ресурс]. — 2019 — Режим доступа: <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
  - 29 Rohith Gandhi. Support Vector Machine — Introduction to Machine Learning Algorithms [Электронный ресурс]. — 2018 — Режим доступа: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
  - 30 Oscar Contreras Carrasco. Support Vector Machines for Classification [Электронный ресурс]. — 2019 — Режим доступа: <https://towardsdatascience.com/support-vector-machines-for-classification-fc7c1565e3>
  - 31 Scott Robinson. Run-Length Encoding [Электронный ресурс]. — 2019 — Режим доступа: <https://stackabuse.com/run-length-encoding/>
  - 32 Elasticsearch. Kibana, FileBeat [Электронный ресурс]. — 2019 — Режим доступа: <https://www.elastic.co/start>



- 33 Riemann. Riemann monitors distributed systems [Электронный ресурс]. — 2019 — Режим доступа: <http://riemann.io/>
- 34 Collectd. The system statistics collection daemon [Электронный ресурс]. — 2019 — Режим доступа: <https://collectd.org/>
- 35 syslog-ng [Электронный ресурс]. — 2019 — Режим доступа: <https://www.syslog-ng.com/products/open-source-log-management/3rd-party-binaries.aspx>
- 36 Docker Inc [Электронный ресурс]. — 2019 — Режим доступа: <https://www.docker.com/>