Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра прикладної математики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Методи ідентифікації інцидентів інформаційної безпеки з використанням машинного навчання
(тема)

Виконав:
студент _2_ курсу, групи _ПМм-22-1
Сергєєв М. О.
(прізвище, ініціали)
Спеціальність
113 Прикладна математика
(код і повна назва спеціальності)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма
Прикладна математика
(повна назва освітньої програми)
Керівник проф. Кіріченко Л. О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ПМ _____ Сидоров М.В.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет _інформаційно-аналітичних технологій та менеджменту
Кафедра _прикладної математики
Рівень вищої освіти _другий (магістерський)
Спеціальність _113 Прикладна математика
(код і повна назва)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма _Прикладна математика
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри ПМ _____
(підпис)
"_06_" _листопада_ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _Сергєєву Микиті Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи _Методи ідентифікації інцидентів інформаційної безпеки з використанням машинного навчання
затверджена наказом по університету від 2 листопада 2023 р. № 1276 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10 січня 2024 р.

3. Вихідні дані до роботи _модельні та реальні реалізації мережного трафику

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області
2. Вибір і обґрунтування методу розв'язання
3. Програмна реалізація
4. Результати обчислювального експерименту
5. Аналіз можливих застосувань

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

1. Актуальність теми роботи
2. Постановка задачі
3. Аналіз предметної області
4. Метод чисельного аналізу
5. Результати обчислювального експерименту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Підбір та вивчення технічної літератури за темою роботи	6 – 12 листопада 2023 р.	виконано
2	Вибір та обґрунтування методу	13 – 26 листопада 2023 р.	виконано
3	Розробка алгоритму і програми	27 листопада – 10 грудня 2023 р.	виконано
4	Проведення аналітичних досліджень та розрахунків	11 грудня – 24 грудня 2023 р.	виконано
5	Робота над текстом пояснювальної записки	25 грудня 2023 р. – 9 січня 2024 р.	виконано
6	Представлення роботи на рецензію в ЕК	10 січня 2024 р.	виконано

Дата видачі завдання _6_ листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Кіріченко Л. О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 60 с., 3 табл., 13 рис., 1 дод., 18 джерел.

МАШИННЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, ОЦІНКА НАВАНТАЖЕННОСТІ РОЗРАХУНКОВОГО ВУЗЛА, РОЗПОДІЛЕНІ СИСТЕМИ, ДИЗБАЛАНС СИСТЕМИ.

Об’єкт дослідження – проблема виявлення дисбалансу завантаженості серверів та її вирішення методами машинного навчання.

Мета роботи – розробка механізму, що дозволяє класифікувати дані сервера з метою аналізу їх на предмет виявлення дисбалансу навантаження на сервери.

Методи дослідження – метод штучних нейронних мереж та метод аналізу дисбалансу.

У магістерській роботі розглянуто завдання розпізнавання дисбалансу системи серверів з використанням штучних нейронних мереж для розв’язання задачі класифікації. Розроблена програмна імплементація штучної нейронної мережі, включаючи модель та реалізацію для аналізу мережевого трафіку. Нейронна мережа була навчена на спеціально згенерованих даних, що виникли з даної моделі. Точність класифікації була досліджена в контексті різних параметрів моделі та типів дисбалансу.

ABSTRACT

Introductory note: 60 pages, 3 tables, 13 figures, 1 appendix, 18 sources.

MASHING EDUCATION, NEURAL NETWORKS, LOAD MEASURE, DISTRIBUTED SYSTEMS, SYSTEM IMBALANCE.

Object of research – the problem of detecting the imbalance of server load and solving it using machine learning methods.

Purpose of work – development of a mechanism that allows classifying server data for the purpose of analyzing them to identify imbalances in the load on servers.

Methods of research – the method of artificial neural networks and the method of imbalance analysis.

The master's thesis deals with the task of recognizing the imbalance of the server system using artificial neural networks to solve the classification problem. Developed software implementation of artificial neural network, including model and implementation for network traffic analysis. The neural network was trained on specially generated data generated from this model. Classification accuracy was investigated in the context of different model parameters and imbalance types.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

MP – Multiprocessing;

HTTP –Hyper Text Transfrer Protocol;

URL – Unified Resource List;

TCP – Transmission Control Protocol;

CAB – Compare And Balance;

ШНМ – штучна нейронна мережа;

RNN – Recurrent Neural Network;

ICMP – Internet Control Message Protocol.

ВСТУП

Актуальність теми. Стрімкий розвиток обчислювальної техніки привів до того, що комп’ютерна мережа стала використовуватися як повно-функціональний розподілений обчислювальний пристрій для обробки і передачі даних.

Зараз проводиться велика кількість досліджень, що мають на меті виявлення характерних властивостей і параметрів технології передачі даних, а також пошук оптимально гарантованих способів виявлення аномалій в роботі комп'ютерної мережі, які можуть будь-яким образом вплинути на процеси що протікають в ній.

При проектуванні, запуску і експлуатації інформаційних комунікаційних мереж однією з основних проблем є завдання забезпечення якості обслуговування (заданих рівнів затримок, втрат і ін.) при обробці потоку даних – трафіку, що є наслідком інформаційного обміну між системами.

На тлі стрімкого розвитку і модифікації шкідливого програмного забезпечення (ПЗ) и підвищення рівня хакерства, антивірусне ПЗ не завжди може повною мірою захистити користувачів комп'ютерних мереж від дій зловмисників. Все частіше поряд зі статистичними методами, які аналізують відповідність конкретної дії в мережі певним шаблоном і записів журналів (брандмауери), використовується аналіз поведінки трафіку під час роботи мережі.

З постійним ростом кількості онлайн-атак та кіберзлочинів, необхідність розробки ефективних методів виявлення та аналізу інцидентів інформаційної безпеки стає вельми критичною. Використання машинного навчання у цьому контексті є обіцяючим напрямком, оскільки воно дозволяє автоматизувати процес виявлення аномалій та надзвичайних подій, які можуть бути зв'язані з потенційними загрозами для інформаційної безпеки. Крім того, швидкість та точність аналізу, які можуть надати алгоритми машинного навчання, дозволяють швидко реагувати на потенційні загрози та запобігати їх поширенню.

Інтернет стрімко перетворюється в ринок послуг з дуже великим оборотом грошових коштів. Збій або недоступність інформаційного сервера тягне величезні фінансові втрати. Останнім часом стала особливо актуальною проблема кібератак і інцидентів інформаційної безпеки. Для забезпечення інформаційної безпеки в організаціях необхідно вміти відображати або запобігати кібератаки, але на сьогоднішній день немає ефективного вирішення.

Універсальних рецептів протидії кібератакам немає. Але вже зараз можна зробити ряд заходів щодо зниження ймовірності реалізації таких атак, грунтуючись на їх класифікації. Саме вона допомагає швидко виявити початок атаки і використовувати відомі методи боротьби для її запобігання. Завдання методів виявлення кібератак полягає в тому, щоб аналізуючи дані, отримані з серверу, можна зробити найбільш точний прогноз щодо того, атакують сервер на даному проміжку часу, чи ні [1].

Актуальність роботи зумовлена тим, що дослідження в цій області сприяє розвитку новітніх технологій і може стати основою для подальших інновацій та розробок у сфері кібербезпеки.

Мета і завдання кваліфікаційної роботи. Метою кваліфікаційної роботи є розробка моделі та методу ідентифікації інцидентів інформаційної безпеки на основі машинного навчання. Для досягнення поставленої мети під час проходження професійної практики необхідно виконати наступні завдання:

– вивчення характеристик і властивостей реалізацій фрактального трафіку;

– вивчення основних понять та властивостей балансування розподілених обчислювальних систем;

– моделювання роботи серверу розподілених систем і розрахунок балансування системи;

– розробка методу виявлення інцидентів інформаційної безпеки на основі машинного навчання.

Об’єктом дослідження є процес детектування інцидентів інформаційної безпеки.

Предметом дослідження є модельні та реальні реалізації мережного трафіку.

Методи дослідження. У роботі використовуються метод штучних нейронних мереж та методи фрактального аналізу.

Публікації. Результати, отримані у роботі, було представлено на 27-му Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» (м. Харків, 10-12 травня 2023 р.) [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Інциденти інформаційної безпеки при неправильному балансуванні навантаження у комп’ютерній мережі

Балансування навантаження є на сьогоднішній день одним з найбільш використовуваних прийомів підвищення продуктивності розподілених обчислювальних систем за рахунок оптимального розподілу завдань між вузлами обчислювальної системи. Під терміном «система балансування навантаження» будемо розуміти інструментальний засіб, призначений для перенаправлення запитів користувачів на найменш завантажений вузол розподіленої обчислювальної системи. Для користувача присутність такої системи має бути прозорим. Завдання системи балансування навантаження полягає в розподілі запитів користувачів між найменш завантаженими обчислювальними вузлами розподіленої системи з метою найбільш повного використання всіх наявних ресурсів. Для виконання цього завдання можуть використовуватися різні методи оцінки завантаження обчислювального вузла.

Система балансування навантаження повинна оперувати даними. які містять інформацію про стан обчислювальних вузлів розподіленої системи, а також правилами заданими адміністратором. На підставі цих даних або правил приймається рішення про перенаправлення запиту на найменш завантажений вузол.

Неправильне балансування навантаження в комп'ютерній мережі може призвести до різних інцидентів інформаційної безпеки, включаючи:

а) перевантаження деяких вузлів: якщо навантаження нерівномірно розподілене, деякі вузли можуть бути перевантажені, що може призвести до відмови обладнання або низької продуктивності, а також виявлення слабких місць у мережевій інфраструктурі;

б) затримки в передачі даних: нерівномірне розподілення навантаження може призвести до затримок у передачі даних через перевантажені вузли (це

може вплинути на швидкість реакції системи та надійність передачі інформації);

в) підвищена вразливість до атак : надмірне навантаження деяких вузлів може зробити їх більш вразливими до атак, оскільки атакуючі можуть спрямувати свої зусилля на ці точки для отримання несанкціонованого доступу або завдання шкоди;

г) недостатня пропускна здатність: якщо деякі вузли перевантажені, це може призвести до недостатньої пропускної здатності для обробки всіх запитів, що може призвести до втрати даних або низької швидкості передачі;

ґ) втрата конфіденційності: недостатнє балансування навантаження може призвести до незаконного доступу до конфіденційної інформації через перевантажені вузли;

д) сприяння DDoS-атакам: нерівномірне розподілення навантаження може полегшити проведення DDoS-атак, оскільки атакуючі можуть сконцентрувати свої зусилля на перевантажених вузлах.

Загалом, неправильне балансування навантаження може призвести до порушення нормального функціонування мережі та загрози інформаційної безпеки.

Система балансування навантаження повинна виконувати наступні завдання: контроль за навантаженням і станом обчислювальних вузлів, вибір, відповідно до заданого алгоритму, вузла для передачі на нього запиту користувача.

Всі алгоритми балансування навантаження можна розділити на дві великі групи: пасивні та активні. До групи пасивних алгоритмів балансування навантаження відносять ті, які працюють з таблицею, яка містить відомості про обслуговуваних вузлах. Такі як тактова частота процесора, кількість оперативної пам'яті, пропускна здатність внутрішньої шини. Для розподілу запитів користувачів пропорційно продуктивності обчислювального вузла в таблицю можуть бути додані вагові коефіцієнти. Таблиця формується до початку роботи системи балансування і майже не змінюється протягом її

роботи. Ці алгоритми, в свою чергу, діляться на дві групи, статичні і динамічні. Статичні алгоритми засновані на розподілі запитів користувача між обчислювальними вузлами за певним правилом, яке встановлюється заздалегідь і не змінюється в процесі роботи виконання алгоритму. Найпростішим прикладом такого алгоритму є алгоритм з циклічної вибіркою.

До групи динамічних алгоритмів відносяться ті, які змінюють правила обходу обчислювальних вузлів під час своєї роботи. В процесі свого функціонування система балансування навантаження працює по динамічному алгоритму розподілу запитів, змінює порядок обходу обчислювальних вузлів в залежності від їх стану системи балансування навантаження, які працюють за динамічним алгоритмам розподілу навантаження самостійно формують таблицю. Описує стан обчислювальних вузлів, не отримуючи даних від самих обчислювальних вузлів. Прикладом може стати алгоритм, заснований на таку ознаку завантаження вузла, як кількість встановлених з'єднань При встановленні чергового з'єднання між користувачем і обчислювальним вузлом система балансування знаходить в своїй таблиці вузол з найменшим на поточний момент кількістю з'єднань і відправляє завдання на виконання в цей вузол, поповнюючи при цьому відповідний запис в таблицю. Коли завдання виконано і з'єднання закривається, таблиця знову коригується. Основним недоліком усіх розглянутих вище алгоритмів є відсутність можливості автокорегування їх роботи в процесі функціонування системи балансування в разі виникнення нерівномірного розподілу навантаження між обчислювальними вузлами. Правила розподілу запитів встановлюються заздалегідь і або не змінюються взагалі, або змінюються незначно. Такі алгоритми доцільно використовувати тільки в комплексі з ще якимось алгоритмом, який реалізує більш гнучкий підхід до розподілу запитів користувачів.

Активні алгоритми розподілу навантаження характеризуються тим, що система балансування навантаження постійно відстежує рівень навантаження і доступність обчислювальних вузлів, які вона контролює, з тим, щоб на підставі наявної інформації в будь-який момент передати завдання користувача найменш

завантаженому вузлу. Тут можна виділити два методи отримання інформації про поточний стан обчислювального вузла. Перший метод - це так званий зовнішній моніторинг стану обчислювальних вузлів розподіленої системи, другий метод – внутрішній моніторинг. Системи балансування навантаження, засновані на зовнішньому моніторингу, як і всі системи балансування, засновані на алгоритмах, які були розглянуті вище, мають централізовану структуру і розташовуються на одному вузлі розподіленої обчислювальної системи.

1.2 Методи запобігання інцидентам, пов'язаним з неправильним балансуванням навантаження в комп'ютерній мережі

Щоб запобігти інцидентам, пов'язаним з неправильним балансуванням навантаження в комп'ютерній мережі, доцільно вживати наступні заходи.

Захід 1. Моніторинг навантаження: постійне відстеження навантаження на вузлах мережі. Вчасно виявлення перевантажених вузлів дозволить вжити відповідних заходів.

Захід 2. Резервні канали та вузли: забезпечення резервних каналів та вузлів, що можна використовувати у разі перевантаження основних.

Захід 3. Використання балансувальників навантаження: встановлення спеціальних пристроїв або програмного забезпечення, яке розподіляє навантаження між вузлами для запобігання перевантаженню.

Захід 4. Оптимізація мережевої інфраструктури: Періодичний аналіз та оптимізація конфігурації мережевого обладнання, щоб уникнути нерівномірного розподілення навантаження.

Захід 5. Використання технологій кешування: для зменшення навантаження на сервери та забезпечення швидшого доступу до даних.

Захід 6. Масштабування системи: у випадку постійного перевантаження важливо розглянути можливість масштабування мережі, включаючи додавання нових вузлів або підвищення потужності існуючих.

Захід 7. Безпека мережі: забезпечте надійний захист мережі від атак та несанкціонованого доступу, щоб уникнути непередбачуваних навантажень.

Система балансування централізовано збирає дані про стан всіх обчислювальних вузлів розподіленої системи. При проведенні зовнішнього моніторингу система балансування навантаження розраховує час відгуку обчислювального вузла, для чого направляє на нього запит на послугу і заміряє час відповіді. Найпростіший варіант такого методу – це використання ping-тестів по протоколу управління повідомленнями Internet Control Message Protocol (ICMP).

Використання прохання про надання послуг протоколу ICMP дозволяє системі переконатися в готовності обчислювального вузла і визначити, скільки часу необхідно для передачі даних на обчислювальний вузол від системи балансування навантаження і назад.

Якщо система балансування не отримує відгуку від обчислювального вузла після декількох послідовних запитів, то даний вузол вважається недоступним і виключається зі списку доступних, якщо час відповіді вузла значно збільшується, то це означає, що відповідний вузол перевантажений і система балансування залишають поза передачею йому запити користувача. Однак затримка в отриманні відповіді може надаватися незалежно від завантаження обчислювального вузла. Тут затримки може вносити середовище передачі даних. Якщо навантаження на мережу в розподіленої системі нерівномірна, то дані про завантаження вузлів, засновані на часі відповіді вузла, можуть бути сильно спотворені.

Наступний метод реалізації зовнішнього моніторингу заснований на відправку службових пакетів отримання більш детальної інформації про завантаження обчислювального вузла. Метод, заснований на ping-тестах по протоколу ICMP діагностує тільки стек протоколів IP. Протоколи верхніх рівнів не будуть зачіпатися простим ping-тестом і для отримання даних про функціонування протоколів верхніх рівнів необхідно використовувати більш складні запити. Прикладом реалізації цього методу зовнішнього моніторингу є

метод, при якому система балансування навантаження встановлює з'єднання з обчислювальним вузлом по протоколу TCP, для чого потрібно здійснити обмін службовими повідомленнями, що складається з трьох етапів.

Після встановлення TCP з'єднання система балансування негайно розриває його, щоб не займати додаткові ресурси обчислювального вузла. Навантаження на стек TCP обчислювального вузла оцінюється за загальним часом, необхідного на встановлення з'єднання з вузлом. При цьому серйозне навантаження відчуває середовище передачі даних, так як кількість службової інформації, переданої по мережі, істотно зростає.

Наступним методом оцінки завантаження обчислювального вузла з використанням зовнішнього моніторингу є підхід, дозволяє забезпечувати моніторинг часу відгуку і готовності як самого обчислювального вузла, так і додатків, що працюють на ньому. В даному випадку час відгуку програми визначається як інтервал часу між відправленням запиту на надання даних і до моменту оголошення про готовність до передачі.

У всіх цих методах можна враховувати продуктивність обчислювального вузла і вводити вагові коефіцієнти. Крім того, на відміну від пасивних алгоритмів балансування, вагові коефіцієнти можна динамічно змінювати в процесі роботи системи балансування, що дозволяє змінювати навантаження на обчислювальний вузол більш гнучко.

Висновок, який можна зробити після аналізу методів оцінки завантаження обчислювальних вузлів з використанням зовнішнього моніторингу, наступний. Ці методи дозволяють отримати дані про завантаження вузла, виражені тільки в часі відповіді на запит системи балансування. При таких істотних характеристиках обчислювального вузла, як стан процесора або процесорів, стан пам'яті або підсистеми введення / виводу, за допомогою зовнішнього моніторингу можна отримати будь-які відомості. Крім того, серйозним недоліком всіх методів зовнішнього моніторингу є те, що завантаження обчислювального вузла визначається за часом, який необхідний для передачі і отримання службового пакета. Великий час відповіді

обчислювального вузла може і не залежати від завантаження самого вузла, а бути таким через перевантаження в мережі. Для отримання більш детальної інформації про стан обчислювального вузла необхідно використовувати методи другої групи.

Внутрішній моніторинг стану вузла відкриває нові можливості для точної і ефективної оцінки завантаження обчислювальних вузлів у розподіленій системі. Кожен вузол взаємодіє із спеціалізованою програмою-агентом, яка активно здійснює збір та обробку даних про його стан. Ця програма-агент періодично надсилає інформацію про стан окремих компонентів вузла на центральну машину системи балансування. Однією з ключових переваг цього методу є те, що всі обчислювальні вузли беруть активну участь в процесі збору та передачі даних. Самостійно ініціюючи збір інформації про своє завантаження, вони стають більш автономними в управлінні власними ресурсами. Дані від програм-агентів можуть передаватися відповідно до запитів центральної частини системи балансування або автоматично, безпосередньо на центральний вузол через певний часовий інтервал. Цей підхід не тільки дозволяє ефективно визначати стан обчислювальних вузлів, але й зменшує навантаження на комунікаційну мережу. Передача даних відбувається меншими порціями, порівняно з методами зовнішнього моніторингу, що дозволяє забезпечити більш швидке та ефективне зібрання інформації про завантаження вузлів. Такий підхід виявляється необхідним у сучасних розподілених обчислювальних системах, де важлива не лише точність вимірювань, але і мінімізація впливу на загальну мережеву інфраструктуру.

Перевагою методу є істотне зменшення навантаження на комунікаційне середовище, так як службові дані передаються меншими порціями в порівнянні зі службовою інформацією, що передається при використанні методів балансування навантаження з використанням зовнішнього моніторингу стану обчислювальних вузлів.

1.3 Формальна та змістовна постановка задачі

Розрахунок завантаження кожного сервера шляхом розрахунку середнього коефіцієнта використання процесора, пам'яті, пропускної здатності мережі -го сервера. Отримана інформація про завантаження використовується для процесу балансування, по-перше, для визначення виникнення дисбалансу, по-друге, для визначення нового розподілу завдань шляхом обчислення обсягу робіт, необхідного для переміщення завдань. Звідси, якість роботи балансування завантаження безпосередньо залежить від точності і повноти інформації.

Дисбаланс завантаження може визначатися синхронно і асинхронно. При синхронному визначенні дисбалансу все процесори (сервери мережі) переривають роботу в певні моменти синхронізації і визначають дисбаланс завантаження шляхом порівняння завантаження окремого процесора із загальною середньою завантаженням. При асинхронному визначенні дисбалансу кожен сервер зберігає історію свого завантаження. У цьому випадку момент синхронізації для визначення ступеня дисбалансу відсутня. Обчисленням обсягу дисбалансу займається фоновий процес, що працює паралельно з завданнями.

Одним з найважливіших властивостей трафіку, як випадкового процесу, є наявність «важких хвостів» його функцій розподілів. Самоподібний трафік має особливу структуру, що зберігається на багатьох масштабах – в реалізації завжди присутня деяка кількість дуже великих викидів при відносно невеликому середньому рівні трафіку. Ці викиди викликають значні затримки і втрати пакетів, навіть коли сумарна потреба всіх потоків далека від максимально допустимих значень.

Наявність у переданих клієнтами інформаційних потоків властивостей самоподібності дуже впливає на ефективність роботи хмарних сервісів. Особливо важливу роль це відіграє для роботи сервісів, що забезпечують передачу мультимедійного трафіку і трафіку реального часу.

Сучасні інформаційні мережі побудовані на основі мультиплексування потоків даних. Відповідно до класичної теорії масового обслуговування, безліч потоків даних з випадковими варіаціями розподілів ймовірностей дадуть в результаті якийсь усереднений згладжений трафік. Однак цей підхід не застосовний до потокам даними, що володіє властивостями самоподібності. Особливе значення це набуває при побудові хмарних інфраструктур, де важливим завданням є оптимальний розподіл навантаження між компонентами.

Основним інструментом дослідження і прогнозування поведінки самоподібних потоків даних є імітаційне моделювання, для якого необхідна модель вхідний самоподобной навантаження. Існує безліч моделей самоподібного трафіку. У роботі запропонована модель агрегованого самоподібного потоку, що враховує ступінь самоподібності і «тяжкість хвоста» функції розподілу. Параметрами моделі є інтенсивність трафіку, показник Херста і коефіцієнт варіації, великі значення якого відповідають великих викидів в реалізації. Дана модель була використана для проведення чисельного моделювання.

У кожний момент часу на балансувальник LB надходить трафік інтенсивністю $\lambda = [\lambda_1, \lambda_2, .., \lambda_\sigma]$, що відноситься до qs-му класу обслуговування, який необхідно доставити на i-й сервер $Serv_i$ для обробки, не перевищуючи заданих максимально допустимих значень затримки τ_{qs} і максимально допустимого відсотка втрат l_{qs} в залежності від поточної завантаження серверів і реальної пропускної здатності в конкретний момент часу.

Трафік має безліч характеристик $V = \{\lambda, h, \mu\}$, де $\lambda = [\lambda_1, \lambda_2, .., \lambda_\sigma]$ – потоки заявок (пакетів) різної інтенсивності; $h = [H, h(q), \Delta h]$, де $h(q)$ – вибіркове значення функції узагальненого показника Херста, $H = h(2)$ – значення параметру Херста, $\Delta h = h(q_{\min}) - h(q_{\max})$ – діапазон значень узагальненого показника Херста для ділянки трафіку, μ – трудомісткість запиту. Трудомісткість запиту визначається як вектор необхідних ресурсів

$\mu = (CPU, Net, RAM)$ для виконання запиту. Кожному qs-му класу обслуговування відповідає набір векторів необхідних ресурсів $\mu_{qs} = (CPU, Net, RAM)$.

Балансувальник навантаження і сервера з'єднані між собою двосторонніми мережевими зв'язками $Link_{lk} = \{L_{lk}\}$, $lk = 1,2,..,$ які діляться на k каналів з пропускною спроможністю $Net_{lk}^{k}(t) = \{Net_{lk}^{k}\}$ в момент часу t. Кожен i-й сервер характеризується наступними параметрами: $CPU_i^{n_i}$ середня загрузка ЦПУ i-го серверу за період часу T, де n_i – кількість ЦПУ (ядер) на i-м сервері (то, як ядра розподілені по процесорам вважатимемо несуттєвим: два чотириядерних процесора відповідають чотирьом двоядерним і відповідають восьми одноядерним процесорам, має значення лише загальне число ядер); середній коефіцієнт використання пам'яті i-го серверу $RAM_i^{m_i}$ за період часу T, де m_i – кількість пам’яті на i-м сервері; середній коефіцієнт використання пропускної здатності каналу $Net_i^{k_i}$ за період часу T, де k_i – кількість каналів в лінії зв'язку між балансувальник і на i-м сервері.

На вхід балансувальника завантаження LB надходять кілька незалежних мультифрактальних потоків пакетів с різною інтенсивністю $\lambda_1, \lambda_2, .., \lambda_\sigma$ кожен з яких відправляється в чергу Q_w обмеженої місткості. Час обслуговування заявок залежить від класу обслуговування qs, тобто враховується пріоритетність заявок (найвищий пріоритет – першим). Поки все пріоритетні запити на обслуговування не будуть оброблені, пакети інших типів залишаються в черзі до закінчення їх часу життя. Знову надійшли пріоритетні запити переривають обробку непріоритетних і з ймовірністю дорівнює одиниці витісняють їх в накопичувач (якщо є вільні місця очікування), або за межі системи (якщо вільних місць немає). Витіснення з обслуговування пакети приєднуються до черги непріоритетних вимог і можуть бут обслуговування після всіх пріоритетних. Накопичувачі є роздільними для кожного вхідного

потоку, вільні місця очікування доступний повністю для будь-якого знову надходження запиту.

На відміну від типових пріоритетних СМО розглянута система забезпечена імовірнісним виштовхує механізмом. Пріоритетний пакет, що застав все місця очікування зайнятими в момент обробки іншого пріоритетного пакета, із заданою вірогідністю витісняє з накопичувача один з менш пріоритетних пакетів і займає його місце. Витіснений пакет втрачається або відправляється назад в чергу. Підсистема балансування завантаження, Відповідно до закладеного в неї алгоритмом здійснює вилучення завдань з черг Q_w і призначення їх на вільні обчислювальні ядра відповідних серверів.

Для опису механізму вивільнення зайнятих трафіком мережевих ресурсів при закінченні передачі трафіку qs-го класу обслуговування, (це відбувається на основі даних, що надходять від протоколу маршрутизації, що підтримує повідомлення про доступну смузі пропускання і доступних ресурсах на сервері, наприклад, CSPF, SNMP), введемо змінну LB $\varepsilon_{C_{L_i}}^{qs,t_0}(t)=\{0,1\}$ вказує, що в момент t на сервер перестав надходити трафік класу $qs(\varepsilon=1)$, який був прийнятий на обслуговування в момент t_0 і повинен був передаватися по шляху $Net_i^{k_i}(t)$ на сервер $Serv_i$. Дана змінна містить всі необхідні дані для визначення мережевих ресурсів, що підлягають вивільненню.

Балансувальник LB у t-й момент характеризується коефіцієнтом втрат та середнім часом очікування пакета в черзі.

Змінна дорівнює відсотку втрат на балансувальник трафіку до класу обслуговування qs, переданого по шляху $Net_{lk}^{k}(t)$ на сервер $Serv_i$ у момент t. Передбачається, що ймовірністю спотворення пакету в тракті можна знехтувати і втрати відбуваються виключно в балансувальник через переповнення буфера.

1.4 Постановка задач дослідження

Метою роботи є розробка моделі та методу ідентифікації інцидентів інформаційної безпеки на основі машинного навчання. Для досягнення поставленої мети необхідно виконати наступні завдання:

– вивчення характеристик і властивостей реалізацій трафіку;

– вивчення основних понять та властивостей балансування розподілених обчислювальних систем;

– моделювання роботи серверу розподілених систем і розрахунок балансування системи;

– розробка методу виявлення інцидентів інформаційної безпеки на основі машинного навчання.

2 ВИБІР ТА ОБҐРУНТУВАННЯ МЕТОДУ РОЗВ’ЯЗАННЯ

2.1 Метод комплексного вимірювання загального рівня дисбалансу системи

Основним недоліком методів балансування навантаження. заснованих на внутрішньому моніторингу стану обчислювального вузла. є присутність резидентної програми на кожному вузлі обчислювальної системи. Службові програми-агенти поглинають певну кількість ресурсів підконтрольного обчислювального вузла. Обсяг необхідних ресурсів залежить від того. по скільком параметрам йде оцінка завантаженості обчислювального вузла, чи йде обробка цих даних на самому вузлі або вони передаються на центральну машину системи балансування навантаження.

Для зменшення кількості необхідних програмою-агентом ресурсів необхідно виявити, які функції вона повинна виконувати. Основна функція програми-агента – надання даних про завантаження підконтрольного їй обчислювального вузла. Тому, при розробці системи балансування навантаження важливим моментом є визначення критеріїв завантаження обчислювального вузла. Від вибору критерію завантаження залежить те, які ресурси будуть затребувані програмою-агентом. які накладні витрати ляжуть на мережу, що зв'язує окремі вузли розподіленої обчислювальної системи.

Якщо програма-агент буде збирати дані про завантаження різних компонентів обчислювального вузла. то для обробки цих даних буде потрібно час і ресурси самого вузла. З іншого боку, оцінка завантаження тільки одного компонента може не дати об'єктивних даних про стан всього вузла в цілому. Найбільш вдалим показником завантаження обчислювального вузла є кількість завдань, які перебувають в черзі на виконання. Число вже очікують завдань безпосередньо впливає на час очікування новоприбулих. Так як розподілена система зазвичай створюється під конкретну задачу або область завдань, то і запити користувачів будуть однотипні.

Звідси можна зробити висновок. що критерій завантаження, заснований на визначенні кількості завдань, які перебувають в черзі на виконання, є досить об'єктивним.

Програма-агент, яка відстежує тільки стан черги завдань на виконання, не вимагає великої кількості обчислювальних ресурсів, що дозволить зменшити і розмір самої програми, що знаходиться в пам'яті обчислювального вузла.

Інше питання, який також впливає на ресурсомісткість програми-агента – це передача обробленої інформації на центральну машину системи балансування. Існує два підходи до вирішення цього завдання: програма-агент сама ініціює передачу даних через певний інтервал часу або передає дані за запитом центрального вузла системи балансування.

Перший метод дозволяє постійно мати свіжі дані про завантаження обчислювальних вузлів, але, з іншого боку, підвищується навантаження на мережу, так як постійно йде передача службових даних. Другий метод дозволяє отримувати дані від всіх вузлів за запитом центрального вузла в той момент. коли необхідно поставити новий запит клієнта в чергу на виконання.

Вибір одного з цих методів залежить від декількох умов. Як зазначалося вище, розподілена система створюється для вирішення певного кола завдань, тому заздалегідь відомо приблизний час на обробку одного завдання. Якщо завдання ресурсномістке, вимагає тривалої обробки і інтенсивного обміну між обчислювальними вузлами, то має сенс використовувати метод, де дані про завантаження обчислювальних вузлів надходять за запитом центральної машини системи балансування навантаження. Якщо ж завдання користувачів виконуються за короткий проміжок часу, наприклад, запити користувачів до Web-вузла, то тоді краще використовувати метод внутрішнього моніторингу з «активними» програмами-агентами, які самі періодично передають дані про завантаження своїх обчислювальних вузлів на центральну машину-диспетчер.

Системи балансування навантаження, побудовані на активних алгоритмах оцінки завантаження обчислювального вузла, розглянутих вище, отримують односторонню інформацію про поточний стан обчислювального вузла.

Внутрішній моніторинг дозволяє отримати досить точну інформацію про завантаження самого обчислювального вузла і його окремих компонентів, зовнішній моніторинг надає дані про завантаження обчислювального вузла за часом його відгуку, таким чином, тестується сегмент мережі, в якому знаходиться даний обчислювальний вузол.

Для отримання найбільш об'єктивної картини завантаження як обчислювального вузла, так і мережевих з'єднань найбільш доцільно використовувати комбінований похід. На кожному вузлі обчислювальної системи повинна знаходитися програма-агент, яка збирає дані про завантаження підконтрольного вузла. Крім цього, центральна машина системи балансування навантаження повинна періодично тестувати мережу для визначення найбільш завантажених сегментів. Це особливо важливо для систем, в яких йде активна взаємодія вузлів один з одним при обробці завдання. Такий підхід дозволить регулювати навантаження не тільки на обчислювальних вузлах, але і в мережі.

З усього вищесказаного можна зробити висновок, що на сьогоднішній момент для ефективної балансування навантаження в розподіленої обчислювальної системи необхідно використовувати комплексний підхід. Класичні алгоритми балансування навантаження не дозволяють досягти найкращих результатів у збільшенні продуктивності розподіленої обчислювальної системи. Пропонований метод балансування навантаження заснований на використанні декількох з перерахованих вище алгоритмів в комплексі. В основу системи балансування навантаження повинен бути покладений метод внутрішнього моніторингу стану обчислювального вузла. Тільки він дозволяє отримати об'єктивну картину завантаження вузла і дані про завантаження окремих компонентів вузла. Критерієм завантаження обчислювального вузла пропонується вважати число завдань, що знаходяться в черзі на виконання. Крім того необхідний моніторинг стану мережі, що зв'язує окремі вузли розподіленої обчислювальної системи. Третій компонент повинен враховувати ту обставину, що сучасні розподілені обчислювальні системи складаються з різноманітних, з точки зору їх обчислювальної потужності,

компонентів. Для обліку цього факту необхідно використовувати вагові коефіцієнти, які будуть враховувати потенційну обчислювальну потужність кожного вузла.

Саме використання комплексного підходу до балансуванню навантаження дозволить домогтися найкращих показників в збільшенні продуктивності розподіленої обчислювальної системи.

З огляду на переваги і недоліки існуючих метрик для планування ресурсів, була розроблена методика комплексного вимірювання загального рівня дисбалансу системи, а також середнього рівня дисбалансу кожного сервера. Розглянуто наступні параметри.

Параметр 1. Середнє завантаження кожного u-го процесору $CPU_i^u(T)$ i-го серверу визначається як середнє завантаження процесора протягом спостережуваного періоду. Наприклад, якщо період спостережень становить 1 хв., А завантаження процесора записується через кожні 10 секунд, тобто CPU_i^u це середнє значення з шести записаних значень i-го серверу.

Аналогічно визначається середнє завантаження кожної r-й пам’яті $RAM_i^r(T)$ i-го сервера і середнє завантаження k-го каналу $Net_i^k(T)$ i-го сервера протягом спостережуваного періоду.

Параметр 2. Оскільки вимірювана завантаження процесора, використання пам'яті і каналу потоком класу qs відрізняється від середнього завантаження процесора CPU_i^{qs}, пам’яті RAM_i^{qs} та каналу Net_i^{qs} потоком класу qs відсутністю часу роботи операційної системи і перемиканням між завданнями, то можна ввести величини CPU_i^{qsv}, RAM_i^{qsv} и каналу Net_i^{qsv} які позначають завантаження процесора, пам'яті і каналу потоками класу qs, виміряну системою обліку або монітором операційної системи.

Завантаження процесора потоком класу qs обчислюється за формулою:

$$CPU_i^{qs} = CPU_i^u \times f_{CPU}^{qs}, \qquad (2.1)$$

де f_{CPU}^{qs} – частка сумарного використання u-го процесора, яку можна віднести на клас qs.

Параметр f_{CPU}^{qs} обчислюється таким чином:

$$f_{CPU}^{qs} = CPU_i^{qsv} / \sum CPU_i^{qsv} \ . \tag{2.2}$$

Точно так же розраховуються значення завантаження пам'яті і пропускної здатності мережі з урахуванням класів потоків. Завантаження пам'яті потоком класу qs обчислюється за формулою:

$$RAM_i^{qs} = RAM_i^r \times f_{RAM}^{qs} , \tag{2.3}$$

де f_{RAM}^{qs} – частка сумарного використання r-й пам'яті, яку можна віднести на клас qs.

Параметр f_{RAM}^{qs} обчислюється таким чином:

$$f_{RAM}^{qs} = RAM_i^{qsv} / \sum RAM_i^{qsv} \ . \tag{2.4}$$

Загрузка каналу потоком класу qs обчислюється за формулою:

$$Net_i^{qs} = Net_i^k \times f_{Net}^{qs} , \tag{2.5}$$

де f_{Net}^{qs} – доля сумарного використання k-го каналу, яку можна віднести на клас qs.

Параметр f_{Net}^{qs} обчислюється таким чином:

$$f_{Net}^{qs} = Net_i^{qsv} / \sum Net_i^{qsv} \ . \tag{2.6}$$

Параметр 3. Введемо середній коефіцієнт використання всіх процесорів в системі. Нехай $CPU_i^{n_i}$ середнє завантаження ЦПУ i-го серверу,

$$CPU_u^{All} = \frac{\sum CPU_i^u CPU_i^{n_i}}{\sum CPU_i^{n_i}}, \quad (2.7)$$

де N – загальне число фізичних серверів в системі;

n_i – кількість ЦПУ на i-м сервері.

Аналогічним чином, середній коефіцієнт використання пам'яті $RAM_i^{m_i}$, пропускної здатності лінії зв'язку $Net_i^{k_i}$ i-го серверу, вся пам'ять RAM_r^{All}, та вся пропускна здатність мережі в системі Net_k^{All} може бути визначена.

$$RAM_r^{All} = \frac{\sum RAM_i^r RAM_i^{m_i}}{\sum RAM_i^{m_i}}, \quad (2.8)$$

$$Net_k^{All} = \frac{\sum Net_i^k Net_i^{k_i}}{\sum Net_i^{k_i}}. \quad (2.9)$$

Параметр 4. Значення дисбалансу всіх процесорів. Використовуючи формулу дисперсії, значення дисбалансу всіх процесорів в системі визначається як

$$ISL_{CPU} = \sum (CPU_i^u - CPU_u^{All})^2. \quad (2.10)$$

Точно так само можуть бути розраховані значення дисбалансу пам'яті і пропускної здатності мережі.

$$ISL_{RAM} = \sum (RAM_i^r - RAM_r^{All})^2, \quad (2.11)$$

$$ISL_{Net} = \sum (Net_i^k - Net_k^{All})^2. \quad (2.12)$$

Параметр 5. Введемо комплексне значення дисбалансу навантаження SIL_i i-го серверу, враховує всі три ресурсу сервера. Використовуючи формулу розрахунку дисперсії як міри нерівномірності, інтегроване значення дисбалансу навантаження i-го серверу можемо визначити як:

$$SIL_i = a(CPU_i^u - CPU_u^{All})^2 + b(RAM_i^r - RAM_r^{All})^2 + c(Net_i^k - Net_k^{All})^2 . \quad (2.13)$$

Параметри a,b,c позначають вагові коефіцієнти для процесора, пам'яті і пропускної здатності мережі, відповідно, які вибираються експериментальним шляхом таким чином, що $a+b+c=1$, в залежності від розв'язуваних завдань і структури системи.

SIL_i застосовується для позначення рівня дисбалансу навантаження шляхом порівняння коефіцієнтів використання процесора, пам'яті і пропускної здатності мережі.

Тоді значення дисбалансу всіх серверів в системі записується наступною сумою:

$$ISL_{tot} = \frac{1}{N}\sum SIL_i . \quad (2.14)$$

Параметр 6. Середня тривалість роботи при однаковій кількості завдань дозволяє порівнювати різні алгоритми планування.

Параметр 7. Період обробки на i-м сервері визначається як максимальне навантаження на i-м сервері. Період обробки в системі визначається як середнє завантаження на всіх серверах.

Параметр 8. Ефективність використання визначається як середнє навантаження на будь-якому сервері.

2.2 Застосування нейронних мереж для виявлення інцидентів і неправильного балансування в системі

Рекурентні нейронні мережі (RNNs) є потужним інструментом для аналізу послідовних даних, таких як часові ряди, текстова інформація чи сигнали. Вони мають здатність враховувати контекст та взаємозв'язки між різними частинами даних, що дозволяє їм ефективно виявляти аномалії та неправильне балансування.

Рекурентні нейронні мережі можуть бути застосовані для виявлення інцидентів та неправильного балансування в комп'ютерних мережах:

– аналіз часових трафіків: RNNs можуть аналізувати часові ряди навантаження в мережі, враховуючи залежності в часі. Це може допомогти виявити аномальні коливання, які можуть свідчити про неправильне балансування;

– виявлення аномалій: RNNs можуть навчитися розпізнавати типові зразки та виявляти аномальні відхилення від них. Наприклад, якщо навантаження виявляється значно вищим або нижчим за звичайне, це може свідчити про інцидент;

– класифікація стану мережі: за допомогою навчання на типових станах мережі, RNNs можуть класифікувати поточний стан мережі та виявляти незвичайні або небажані стани;

– прогнозування навантаження: на основі попередніх даних, RNNs можуть намагатися прогнозувати майбутнє навантаження. Якщо прогноз сильно відрізняється від фактичного значення, це може вказувати на проблему;

– моніторинг поведінки користувачів: у випадку виявлення незвичайної активності від конкретного користувача або пристрою, RNNs можуть виявити це як потенційний інцидент.

Для виявлення інцидентів, пов'язаних з неправильним балансуванням навантаження в комп'ютерній мережі, можна використовувати різні типи нейронних мереж, але найбільш ефективним може бути використання

рекурентних нейронних мереж (RNNs) та глибоких нейронних мереж (Deep Neural Networks - DNNs).

Основні кроки в застосуванні нейронних мереж для виявлення неправильного балансування навантаження.

Крок 1. Збір та підготовка даних: зібрати дані про навантаження в мережі, такі як вхідні/вихідні швидкості передачі даних, обсяги пакетів, часові мітки тощо; розділіть дані на навчальну та тестову вибірки для тренування та оцінки моделі.

Крок 2. Обробка та нормалізація даних: попередній аналіз та обробка даних може включати в себе видалення аномалій, виокремлення важливих ознак, нормалізацію даних тощо.

Крок 3. Вибір моделі: вибір типу нейронної мережі залежить від конкретного випадку та обсягу даних. Використання RNNs може бути корисним для аналізу часових рядів.

Крок 4. Тренування моделі: використовуйте навчальну вибірку для тренування обраної нейронної мережі. Під час тренування модель намагається вивчити внутрішні зв'язки між вхідними та вихідними даними.

Крок 5. Валідація та тестування моделі: використовуйте тестову вибірку для оцінки ефективності моделі. Оцініть її точність, чутливість та специфічність виявлення неправильного балансування навантаження.

Крок 6. Налаштування та оптимізація моделі: в залежності від результатів тестування, внесіть необхідні корективи у модель та її параметри.

Висновки за розділом 2

У висновку можна зазначити, що основні тези стосуються недоліків методів балансування навантаження на основі внутрішнього моніторингу стану обчислювального вузла. Зокрема, обговорено проблеми, пов'язані з присутністю резидентних програм-агентів на вузлах, що може призводити до

значного споживання ресурсів. Також акцентовано увагу на важливості визначення ефективних критеріїв завантаження, зокрема кількості завдань у черзі, як одного з ключових показників.

Запропоновано вирішення цих проблем шляхом використання рекурентних нейронних мереж для аналізу послідовних даних у комп'ютерних мережах. Автор також рекомендує комбінування декількох алгоритмів балансування навантаження для досягнення оптимальних результатів. Узагальнюючи, текст вказує на важливість створення ефективних систем балансування, які забезпечують мінімізацію витрат ресурсів та ефективно впорядковують навантаження в розподіленій обчислювальній системі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Огляд мови програмування Python

Python – це перш за все об'єктно-орієнтована мова програмування з невеликою кількістю ключових слів, що робить її простою та гнучкою. Вона вищорівнева порівняно з Pascal, C++, чи C завдяки вбудованим високорівневим алгоритмам та структурам даних.

Однією з ключових переваг Python є його широка реалізація інтерпретатора на різних платформах, що відрізняє його від мови C. Це робить його більш адаптованим та гнучким, оскільки алгоритми та структури даних мають стабільну роботу на різних машинах, уникнувши проблем пам'яті.

Розширюваність – ще один важливий аспект. Python приділяє велику увагу цьому, розглядаючи мову як розширювану. Його інтерпретатор написаний на мові C та відкритий для маніпуляцій. Можна вставити його як модуль у програму або розробляти власні методи та алгоритми на мові C для подальшого використання в Python.

Однією з найважливіших переваг Python є наявність багатьох модулів, які можна інтегрувати для розширення можливостей програми. Ці додатки, як правило, програмуються на мові C або Python і розробляються досвідченими розробниками. Наприклад, бібліотека NumPy, що позиціонує себе як вдосконалений еквівалент Mathlab для математичних розрахунків.

Python володіє рядом ключових переваг, до яких можна віднести:

– чистий синтаксис, де для відзначення блоків використовуються відступи;

– переносність програм, що є характерним для більшості інтерпретованих мов;

– обширний набір корисних модулів у стандартному дистрибутиві, включаючи модуль для розробки графічного інтерфейсу;

– можливість використання Python у діалоговому режимі, що є дуже зручним для експериментів та розв'язання простих задач;

– просте, але потужне середовище розробки, відоме як IDLE, яке входить у стандартний дистрибутив і написане на мові Python;

– зручність для розв'язання математичних проблем, включаючи засоби роботи з комплексними числами та операції з цілими числами довільної величини;

– відкритий код, що надає можливість редагування іншими користувачами.

Python також вражає ефективними структурами даних високого рівня та простим, але ефективним підходом до об'єктно-орієнтованого програмування. Його елегантний синтаксис, динамічна обробка типів та інтерпретована природа роблять його ідеальним для написання скриптів та швидкої розробки прикладних програм у різних галузях на більшості платформ.

Оскільки Python простий та гнучкий, його рекомендується не лише програмістам, але й математикам, статистикам, економістам, фізикам та іншим фахівцям, які використовують програмування та обчислювальні додатки в своїй діяльності.

3.2 Багатопоточніть у Python

Існують дві основні причини використання потоків: перше, для підвищення ефективності використання багатоядерної архітектури сучасних процесорів та, отже, продуктивності програми. Друге – якщо необхідно розділити логіку роботи програми на паралельні повністю або частково асинхронні секції (наприклад, для одночасного нотифікування кількох серверів). Однак використання багатопоточності супроводжується двома значущими недоліками:

– кожному потоку потрібно місце для роботи з даними, тому кожен потік забирає пам'ять, навіть якщо вона не використовується, і потік спить;

– якщо два потоки використовують один і той же ресурс, може виникнути змагання за цей ресурс.

Для подолання цих проблем може бути використаний інший підхід. У Python існує модуль multiprocessing, який дозволяє створювати процеси звичайними функціями. Методи роботи з процесами схожі на методи роботи з потоками з модуля threading. Однак для синхронізації процесів і обміну даними рекомендується використовувати інші інструменти, такі як черги і канали.

Крім того, в модулі multiprocessing реалізований механізм роботи із загальною пам'яттю, що дозволяє використовувати класи змінної і масиву між процесами.

3.3 Опис програми

Програма розроблена в середовищі PyCharm за використання високорівневої мови програмування Python. Основною метою програми є класифікація отриманих запитів щодо можливості атаки за кількома параметрами. Головна задача полягає в отриманні класифікатора, який може із певною ймовірністю вказати, чи є запит від бота чи від людини.

Програма отримує запити, які надходять на сервер, та розбиває їх на окремі компоненти для подальшого аналізу. Цей етап включає в себе використання стандартних модулів Python для обробки HTTP-запитів та розбиття їх на окремі параметри, такі як тип запиту, URL, протокол, інформація відносно хоста та інші.

На основі отриманих параметрів програма виконує аналіз та виділення ознак, що можуть свідчити про можливу атаку. Для цього використовуються стандартні бібліотеки Python, такі як re для обробки рядків та виражень, або urllib для аналізу URL-адрес.

Програма використовує нейронну мережу для класифікації отриманих ознак та визначення ймовірності атаки. Для цього можуть бути використані стандартні бібліотеки для глибокого навчання, такі як TensorFlow або PyTorch. Нейронна мережа підтримується навчанням на попередньо підготовленому наборі даних, який включає в себе приклади атак та нормальних запитів.

Для оцінки завантаження системи може бути використана нейронна мережа, що навчалася на даних про обсяг та характер запитів. Під час моделювання роботи серверів та введення в нейронну мережу відомостей про навантаження, можна прогнозувати ймовірність перевантаження та визначити оптимальні стратегії балансування навантаження.

Таким чином, розроблена програма на Python дозволяє ефективно моделювати та класифікувати запити на сервері, використовуючи для цього стандартні модулі та бібліотеки мови програмування, а також нейронні мережі для прогнозування та оцінки завантаження системи.

Висновки за розділом 3

У даному пункті відображено ключові особливості мови програмування Python, висловлено переваги та сильні сторони цієї мови. Python вирізняється своєю простотою та гнучкістю, що робить його популярним в різних галузях, від програмування та аналізу даних до розробки веб-застосунків та наукових досліджень.

Також надано огляд багатопоточності у Python, де визначено дві головні причини використання потоків та розглянуті недоліки, які можуть виникнути при їхньому використанні. Зазначено, що для подолання цих недоліків можна використовувати модуль multiprocessing, який дозволяє створювати процеси та вирішує питання синхронізації процесів та обміну даними.

Нарешті, в описі програми подано інформацію про розробку класифікатора на мові Python з використанням нейронної мережі. Описано кроки підготовки даних, вибір маркерів та побудову мережі. Також вказано на використання модулю multiprocessing для прискорення роботи програми.

4 РЕЗУЛЬТАТИ ОБЧИСЛЮВАЛЬНОГО ЕКСПЕРИМЕНТУ ТА ЇХ АНАЛІЗ

4.1 Розробка системи балансування навантаження

Балансування навантаження є стратегічним підходом у розподілених обчислювальних системах, мета якого полягає в ефективному розподілі завдань між різними обчислювальними вузлами. Під терміном "система балансування навантаження" мається на увазі інструмент, спрямований на перенаправлення запитів користувачів до найменш завантажених вузлів в системі. Головна мета цього підходу - забезпечити оптимальне використання ресурсів системи шляхом розподілу запитів між найменш завантаженими вузлами.

Система балансування навантаження опирається на дані про стан обчислювальних вузлів та задані правила, що визначають оптимальний вибір вузла для перенаправлення конкретного запиту користувача. Завдання включають в себе контроль за завантаженням та станом вузлів, а також вибір оптимального вузла для передачі конкретного запиту відповідно до заданого алгоритму.

Методи внутрішнього моніторингу, що використовують активні алгоритми, дозволяють кожному вузлові самостійно збирати дані про свій стан. Це дозволяє зменшити навантаження на мережу та отримати більш детальну і об'єктивну інформацію про стан вузла. Однак ці методи мають свої недоліки, такі як присутність резидентної програми на кожному вузлі, що може вимагати додаткових ресурсів та впливати на мережеве середовище.

Враховуючи ці аспекти, можна вважати, що критерій завантаження, заснований на визначенні кількості завдань, що знаходяться в черзі на виконання, є досить об'єктивним. Такий показник завантаження може надати достатньо об'єктивних даних про стан обчислювального вузла загалом, оскільки кількість завдань, що очікують виконання, безпосередньо впливає на час очікування нових завдань. Однак слід брати до уваги специфіку розподіленої системи та типу завдань, що обробляються.

Програма-агент, яка відстежує лише стан черги завдань на виконання, може ефективно працювати із значно меншою кількістю обчислювальних ресурсів. Це дозволяє зменшити розмір самої програми в пам'яті обчислювального вузла.

Системи балансування навантаження, побудовані на активних алгоритмах оцінки завантаження обчислювального вузла, отримують односторонню інформацію про його стан. Внутрішній моніторинг дозволяє отримати точну інформацію про завантаження обчислювального вузла та його компонентів, тоді як зовнішній моніторинг надає дані про завантаження за часом відгуку, тестуючи сегмент мережі, в якому розташований обчислювальний вузол.

Для отримання найбільш об'єктивної картини завантаження обчислювального вузла та мережевих з'єднань рекомендується використовувати комбінований підхід. Кожен обчислювальний вузол мережі повинен мати програму-агента, яка збирає дані про завантаження вузла. Крім того, центральна машина системи балансування навантаження повинна періодично тестувати мережу для визначення найбільш завантажених сегментів, особливо для систем з активною взаємодією між вузлами при обробці завдань. Це дозволяє регулювати навантаження не тільки на обчислювальних вузлах, але й у мережі в цілому.

Аналізуючи вищезазначене, можна зробити висновок, що для ефективного балансування навантаження в розподіленій обчислювальній системі необхідно використовувати комплексний підхід. Класичні алгоритми балансування навантаження, окремо взяті, не забезпечують найкращих результатів у плані підвищення продуктивності розподіленої обчислювальної системи.

Пропонований метод балансування навантаження базується на використанні кількох з перерахованих вище алгоритмів у комплексі. В основу системи балансування навантаження слід покласти метод внутрішнього моніторингу стану обчислювального вузла, оскільки тільки він дозволяє отримати об'єктивну картину завантаження вузла та дані про завантаження його окремих компонентів. Критерієм завантаження обчислювального вузла пропонується

вважати кількість завдань, що знаходяться в черзі на виконання. Додатково важливий є моніторинг стану мережі, яка з'єднує окремі вузли розподіленої обчислювальної системи. Третій компонент системи повинен враховувати різнобарвність обчислювальної потужності компонентів, що складають розподілену обчислювальну систему. Для врахування цього факту використання вагових коефіцієнтів, які враховують потенційну обчислювальну потужність кожного вузла, є необхідним.

4.2 Моделювання балансування завантаження системи з декількома серверами

Трафік, що надходить на вхід системи, має певні фрактальні властивості, в залежності від яких балансувальник, певним методом розподіляє його по системі серверів (рисунок 4.1).

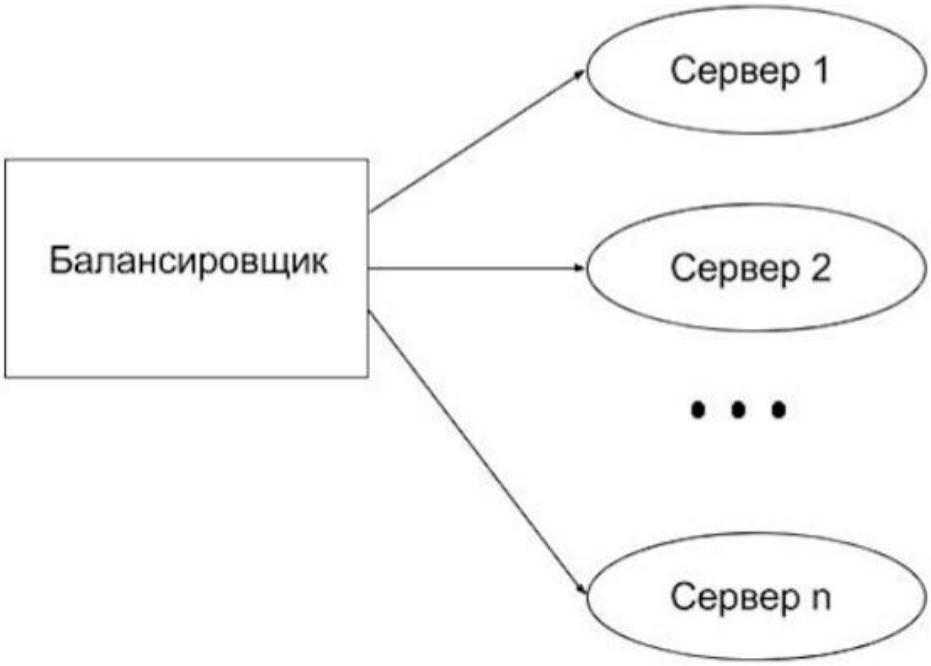

Рисунок 4.1 - Діаграма взаємодії компонентів системи

Розглянемо завантаження кожного сервера за параметрами трафіку $h = 0{,}6$; $k = 3$ на рисунках 4.2 – 4.3 зображені ресурси «1» и «2» серверів. Початково на «1» сервері маємо 450 RAM, 400 CPU и 300 BANDWITH а на «2» сервері 350 RAM, 300 CPU и 250 BANDWITH. На балансувальник надходить згенерований трафік з певними параметрами з частотою 0,25 секунди. Графіки завантаженості ресурсів виходять відніманням від ресурсів сервера, що залишилися, ресурсів які приходять з балансувальника. Ми проводили експеримент із методом балансування «Compare And Balance» в якому враховується нинішній стан завантаженості сервера в системі і найбільш вільний сервер бере на себе ресурс, що прийшов. Відповідно на графіку сплески трафіку відбиваються у вигляді западин.

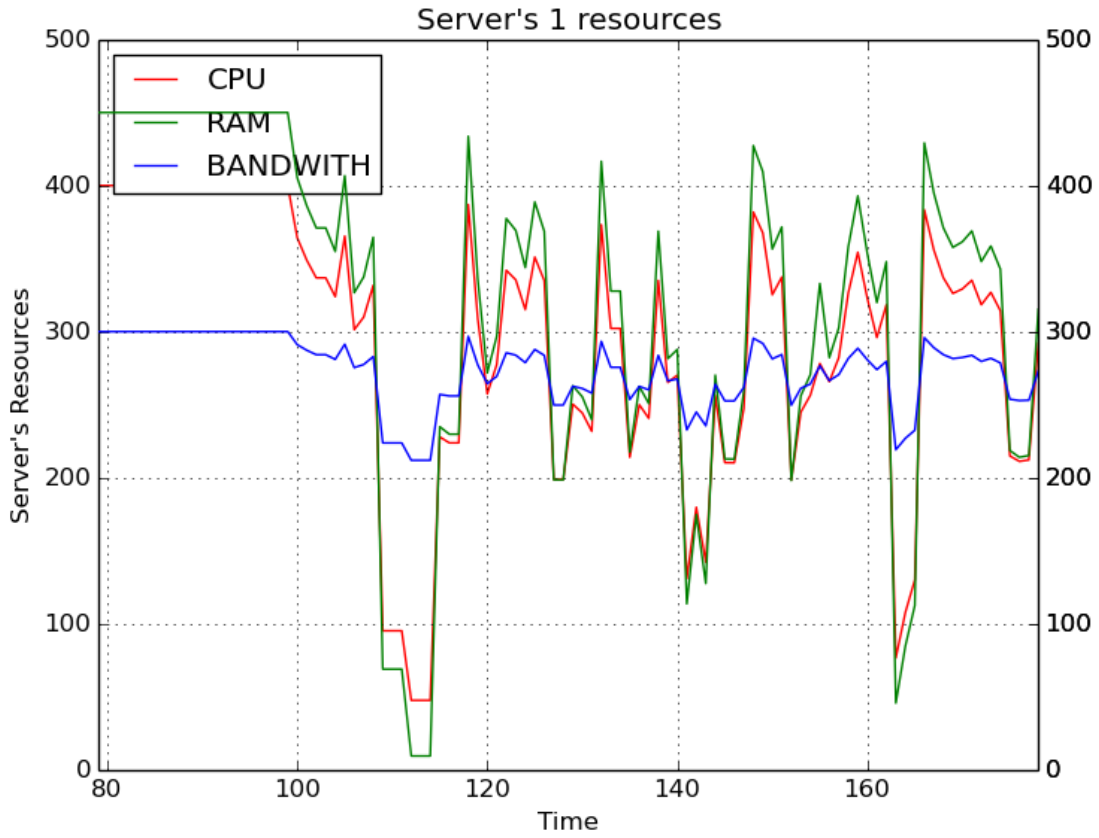

Рисунок 4.2 – Графік завантаженості ресурсів сервера «1»

при параметрах трафіка $h = 0{,}6$; $k = 3$

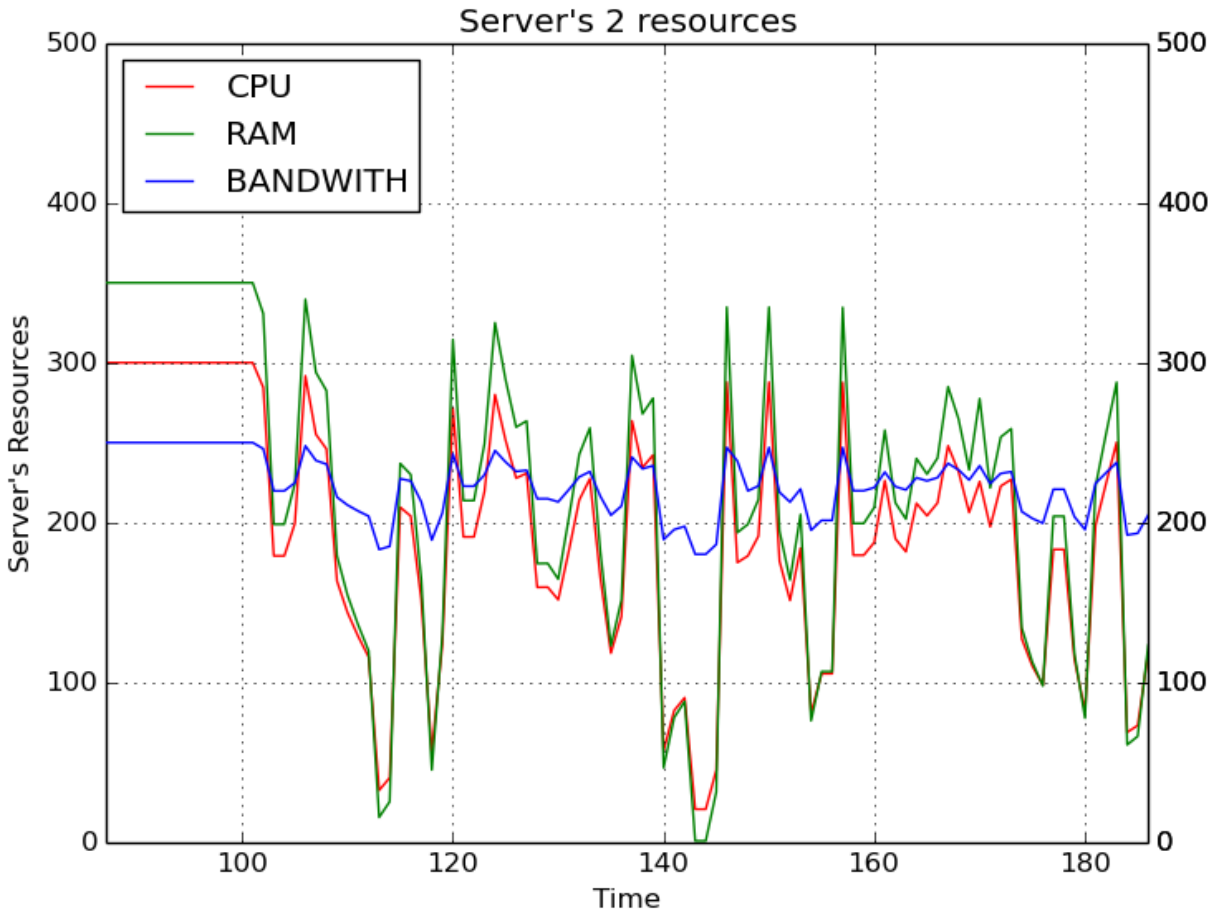

Рисунок 4.3 – Графік завантаженості ресурсів сервера «2»

при параметрах трафіка $h = 0{,}6$; $k = 3$

Виходячи з рисунків 4.2 – 4.4 можна сказати, що при низькому значенні показника Херста ми отримуємо сплеск дисбалансу на підході, що було природно та теоретично обґрунтовано, а після значення дисбалансу практично сходять на 0, бо при такому показнику Херста ми спостерігаємо малу наявність сплесків та різких перепадів графіка. Далі розглянемо показання роботи програми за ідентичних умов балансувальника і серверах але за іншому показнику Херста.

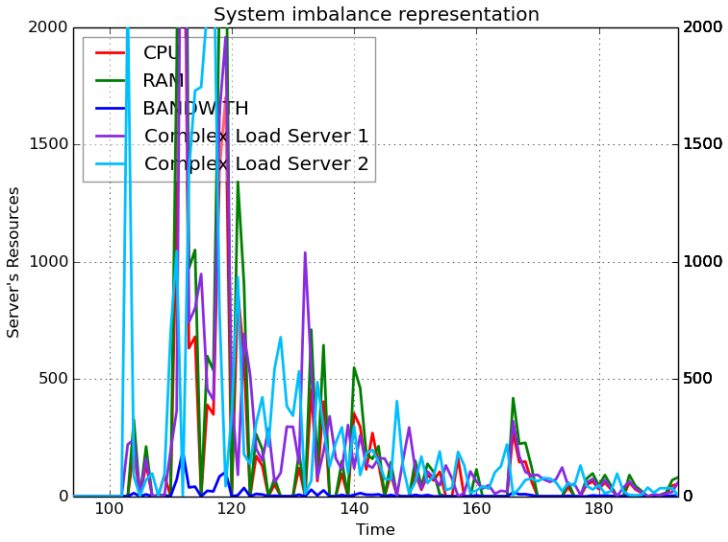

Рисунок 4.4 – Графік дисбалансу для CPU, RAM, BANDWITH, комплексне значення дисбалансу навантаження на кожен з серверів при параметрах трафіка $h = 0{,}6$; $k = 3$

Розглянемо завантаження кожного сервера за параметрами трафіку $h = 0{,}9$; $k = 3$ які зображені на рисунках 4.5 – 4.7.

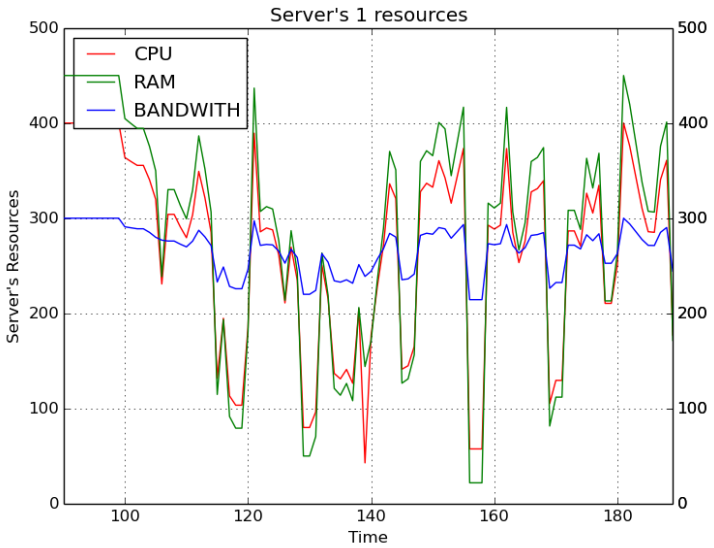

Рисунок 4.5 – Графік завантаженості ресурсів сервера «1» при параметрах трафіку $h = 0{,}9$; $k = 3$

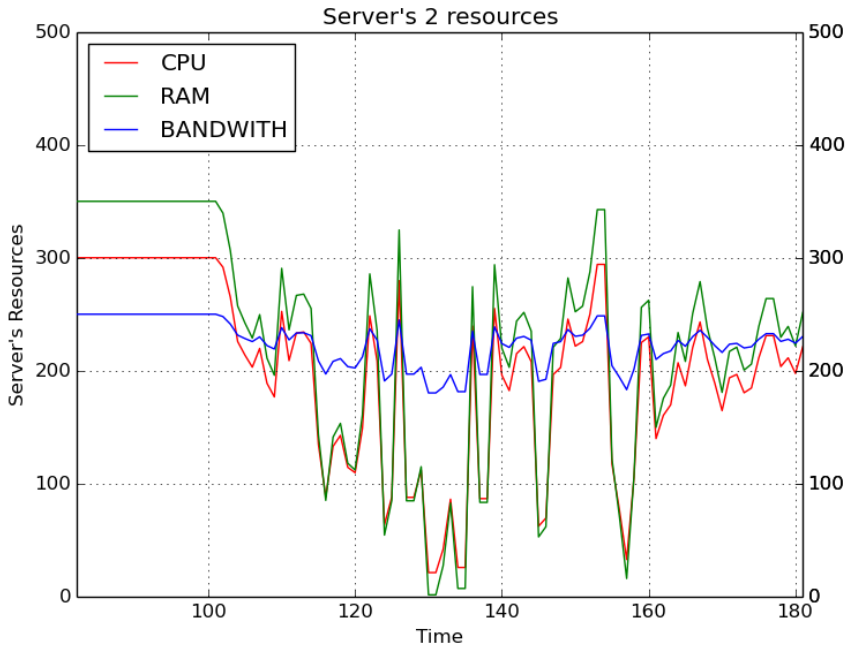

Рисунок 4.6 – Графік завантаженості ресурсів сервера «2»

при параметрах трафіку $h = 0{,}9$; $k = 3$

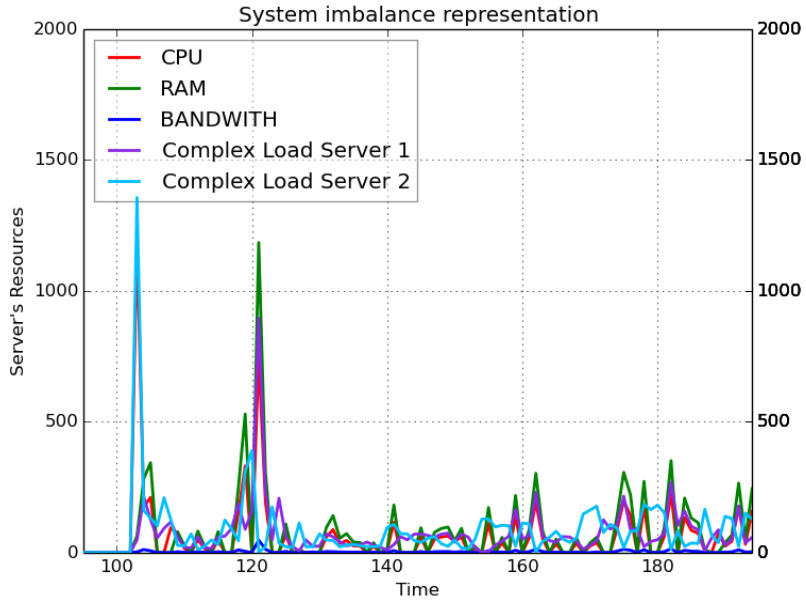

Рисунок 4.7 – Графік дисбалансу для CPU, RAM, BANDWITH, комплексне значення дисбалансу навантаження на кожен з серверів

при параметрах трафіку $h = 0{,}9$; $k = 3$

Як бачимо на рисунках 4.5 – 4.7, візуально складно відрізнити результати завантаженості серверів від першого експерименту. За такого показника Херста теоретично ми маємо спостерігати різкі, великі перепади трафіку, і дивлячись на графік дисбалансу видно, що з часом дисбаланс не зменшується. Для наступного експерименту нам довелося сильно збільшити характеристики наших серверів (у 10 разів збільшено пам'ять, пропускну здатність та потужність процесора) та у 250 разів зменшити масштаб графіка дисбалансу, щоб оцінити результати візуально. На рисунках 4.8 – 4.10 ми побачили , що з зміні параметра k, що відповідає величину і частоту сплесків ми маємо величезні сплески використання ресурсів які супроводжуються різкими сплесками дисбалансу у системі.

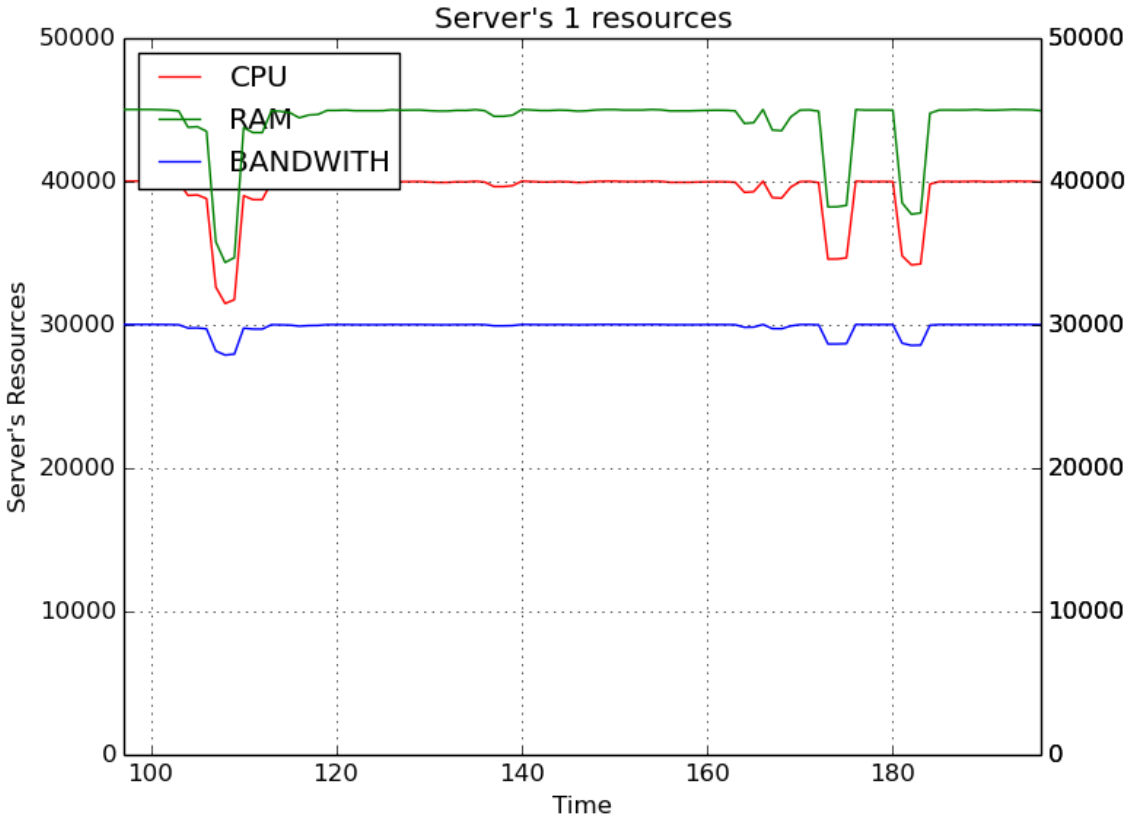

Рисунок 4.8 – Графік завантаженості ресурсів серверу «1»

при параметрах трафіку $h = 0{,}9\,;\ k = 4$

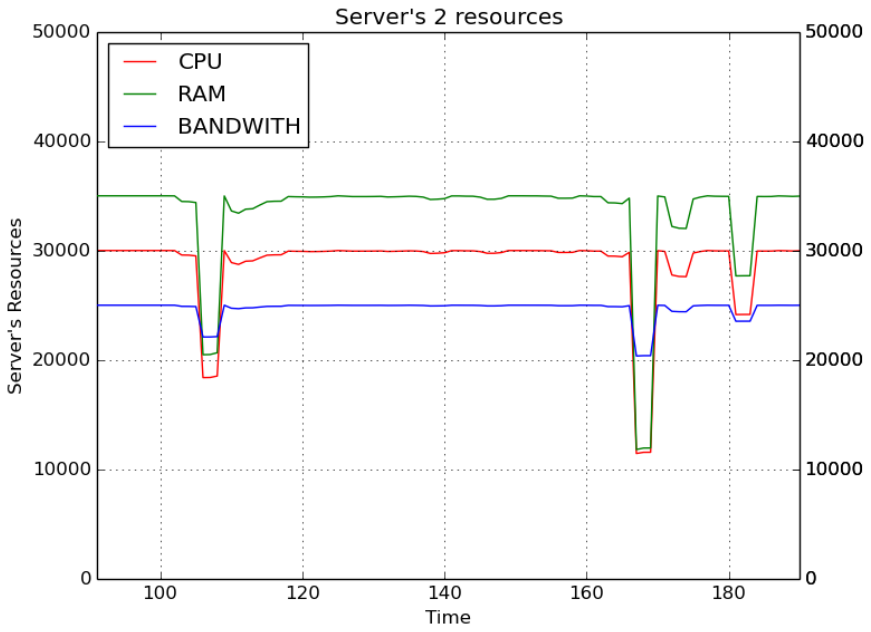

Рисунок 4.9 – Графік завантаженості ресурсів серверу «2»

при параметрах трафіку $h = 0{,}9\,;\ k = 4$

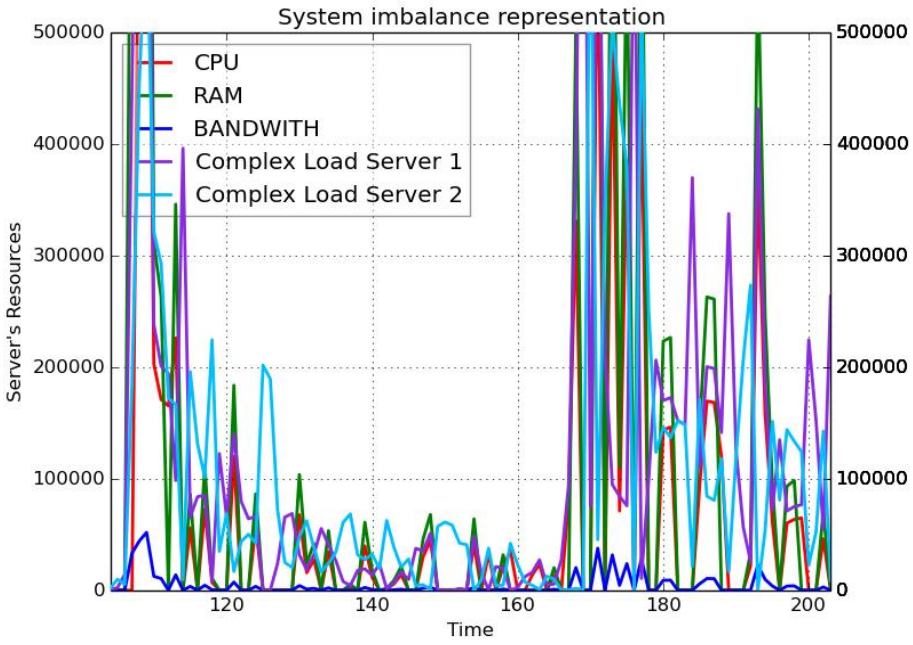

Рисунок 4.10 – Графік дисбалансу для CPU, RAM, BANDWITH,

комплексне значення дисбалансу навантаження на кожен

з серверів при параметрах трафіку $h = 0{,}9\,;\ k = 4$

4.3 Застосування машинного навчання для визначення дисбалансу системи

Для розв'язання задачі визначення дисбалансу системи та навчання комп'ютерної мережі було використано такий підхід.

Етап 1. Було зібрано дані про роботу балансувальника та розподіл ресурсів системи між серверами. Параметри, такі як завантаження CPU, об’єм оперативної пам’яті, характеристики трафіку, кількість серверів, тощо були включені в аналіз.

Етап 2. Зібрані дані було розбито на два класи: нормальний стан системи (без дисбалансу) та стан системи з дисбалансом. Кожен клас було позначено відповідною міткою.

Етап 3. Введено штучний дисбаланс в навчальний датасет для урахування різних сценаріїв навчання.

Етап 4. Вибрано модель нейронної мережі для навчання, яка оптимально вирішує завдання класифікації.

Етап 5. Дані було розділено на навчальний та тестовий набори для ефективної оцінки результатів моделі.

Етап 6. Модель нейронної мережі було навчено на навчальному наборі, використовуючи алгоритми машинного навчання. Впевнено, що модель отримала інформацію про стани системи з дисбалансом та без нього.

Етап 7. Проведено оцінку точності моделі на тестовому наборі, порівнюючи прогнозовані та фактичні класи.

4.4 Проведення експерименту класифікації для визначення дисбалансу за допомогою нейронної мережі

Таким чином було вирішено завдання класифікації даних на два класи: нормальний стан системи та стан системи з дисбалансом. Для отримання даних

для навчання нейронної мережі використовувалася модель фрактального трафіку на основі броунівського рух. Створено кілька навчальних наборів з різними початковими даними.

Кожен з наборів містить 5000 різних масивів даних, половина з яких містить нормальний стан системи, а інша половина – ні. Архітектуру побудованої нейронної мережі можемо побачити на рисунку 4.11. Для кожного з наборів була проведена навчання нейронної мережі та оцінена точність класифікації.

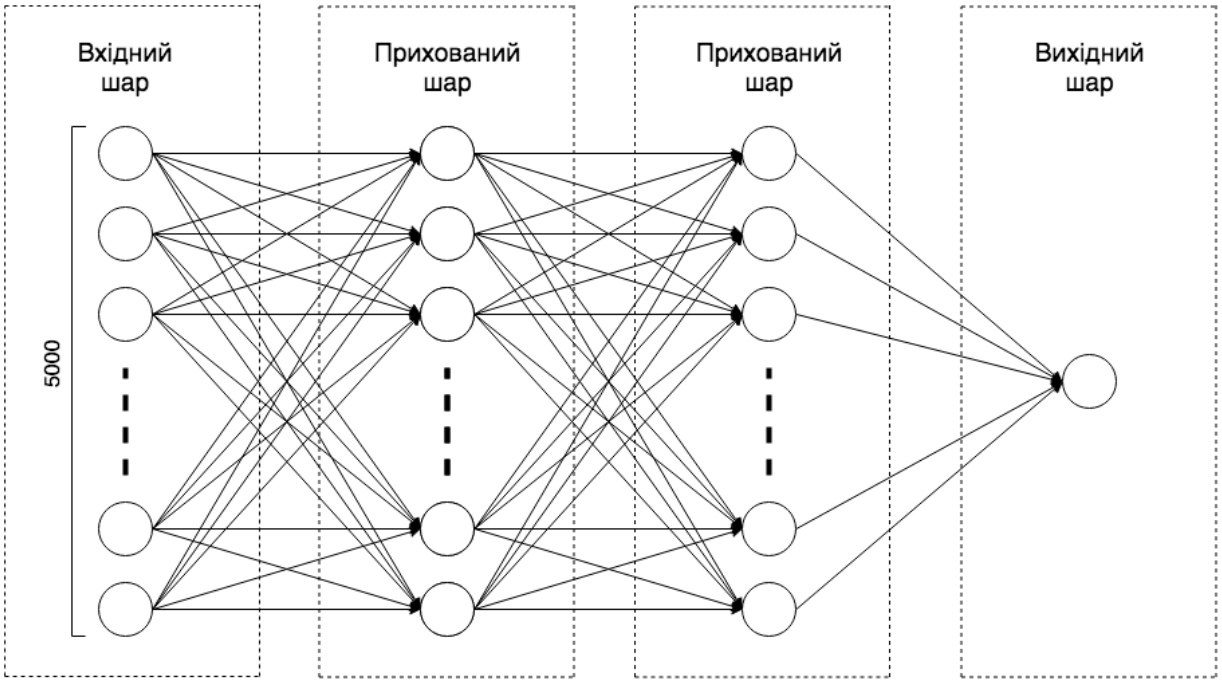

Рисунок 4.11 – Архітектура штучної нейронної мережі

Тренувальні дані, які ми отримали, були розділені у співвідношенні 70/30 на дві частини:

– тренувальний набір, який використовується для навчання нашої нейронної мережі;

– тестовий набір, що використовується для оцінки якості навчання мережі.

Цей розподіл обумовлений тим, що нейронна мережа, яка має найменшу помилку на тренувальному наборі, може виявити велику помилку на нових даних через явище «перенавчання», коли мережа занадто сильно адаптується до тренувальних даних. У наступному пункті ми розглянемо проблему перенавчання нейронної мережі та підхід який можна використовувати задля вирішення цієї проблеми.

4.5 Проблема перенавчання

Якщо навчена нейронна мережа успішно розпізнає приклади з тренувальної множини, але не здатна адекватно розпізнавати інші приклади, крім тих, що входять до тренувального набору, ми можемо стверджувати, що мережа перенавчилася. Перенавчання виникає через занадто глибоке адаптування мережі до тренувальних прикладів. Один із способів боротьби з цим явищем – розділення даних на тренувальну і тестову частини. Навчання проводиться на тренувальній множині, а ефективність моделі оцінюється на тестовій. Важливо, щоб ці дві множини не перетиналися.

Параметри моделі змінюються під час кожної ітерації, і хоча значення цільової функції зменшується на тренувальній вибірці, виникає необхідність вивчення впливу на тестовій множині. З початку обидві помилки зменшуються паралельно. Проте після певного етапу помилка на тестовій множині може збільшуватися, хоча на тренувальній вона продовжує зменшуватися. Цю точку вважається кінцем оптимального навчання, і після цього мережа може почати перенавчання.

Для розв'язання цього проблемного випадку використовується концепція "спроб". Розділяючи вибірку на тренувальну та тестову частини, ми можемо спостерігати за процесом навчання нейронної мережі (рисунок 4.12). Завдяки спробам, тобто.

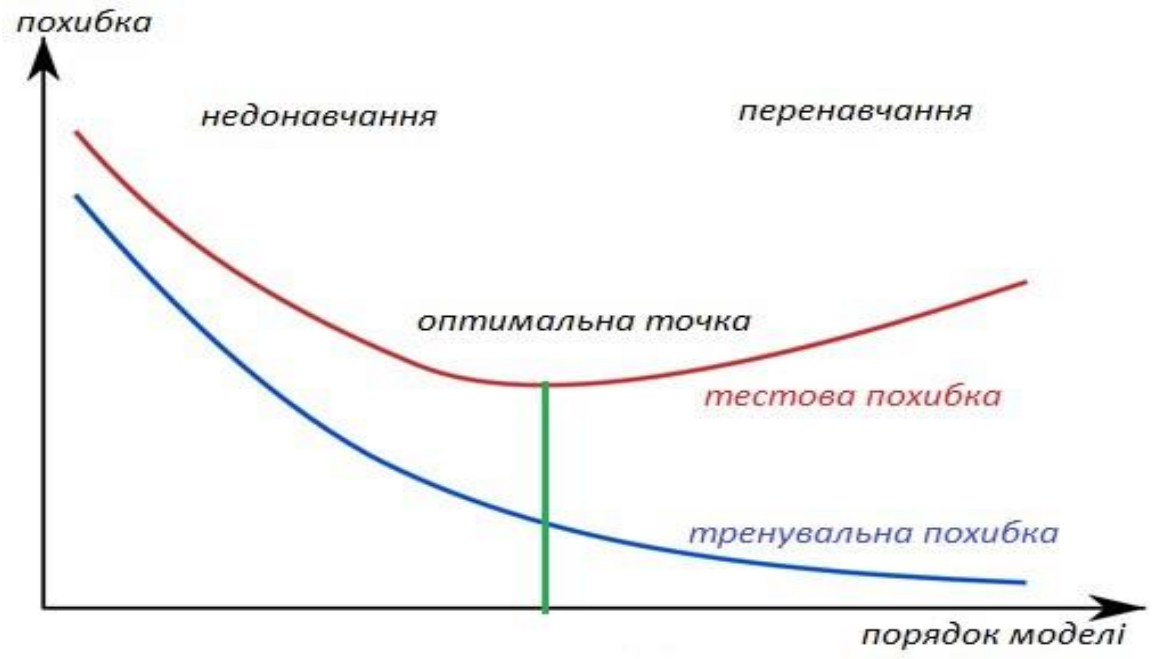

Рисунок 4.12 – Графік тренувальної та тестової похибок у процесі навчання мережі

Для обрання оптимальної кількості спроб прогонів нейронної мережі було виміряні тестові похибки та час виконання роботи алгоритму для різної кількості спроб. Результати можна побачити у таблиці 4.1.

Таблиця 4.1 – Залежність похибки та часу роботи від кількості спроб

К-сть спроб (шт.)	5	10	25	50
Тестова похибка (%)	17,21	10,64	9,12	7,73
Час роботи алгоритму (с)	1,72	2,52	7,83	13,02

Після проведення 10 спроб виявлено, що значне збільшення часу виконання алгоритму не призводить до бажаного покращення точності. Тому для нашої нейронної мережі було встановлено ліміт в 10 спроб.

З метою прискорення роботи алгоритмів було вирішено використовувати модуль multiprocessing та розділяти алгоритм на процеси. Паралелізація виконувалася шляхом розподілу даних на окремі файли, що дозволяло кожному процесу працювати з власним файлом. В таблиці 4.2 можна побачити

залежність часу відпрацювання алгоритму та тестової похибки від кількості процесів (загальна довжина вибірки – 1000).

Таблиця 4.2 – Залежність похибки та часу роботи від кількості процесів

К-сть процесів (шт.)	1	2	3	5
Тестова похибка (%)	1,98	3,12	4,86	6,01
Час роботи алгоритму (с)	59,96	38,45	23,30	15,97

4.6 Результати досліджень

Після проведення досліджень та аналізу була створена нейронна мережа для класифікації даних на два класи: нормальний стан системи та стан системи з дисбалансом. Складається вона з одного прихованого шару, розмір якого подвоєний порівняно з вхідним шаром. Функція активації сигмоїда була обрана для прихованого шару, а Softmax – для вихідного. Зафіксовано кількість епох на рівні десяти для досягнення бажаної точності та кількість спроб також у десяти для запобігання перенавчанню. Навчальна вибірка була розділена у співвідношенні 70/30 для вирішення проблеми перенавчання. Отримана мережа зображена на рисунку 4.13.

На вхід нейронна мережа отримує вектор приналежності ознак, який формується зі значень, таких як завантаження CPU, об’єм оперативної пам’яті, характеристики трафіку, кількість серверів, тощо.

У отриманому класифікаторі була проаналізована залежність помилок першого роду (хибне прогнозування класифікатором дисбалансу), другого роду (хибне прогнозування класифікатором нормального стану) та часу відпрацювання алгоритму в залежності від довжини тренувальної вибірки. Результати досліджень відображені у табл. 4.3.

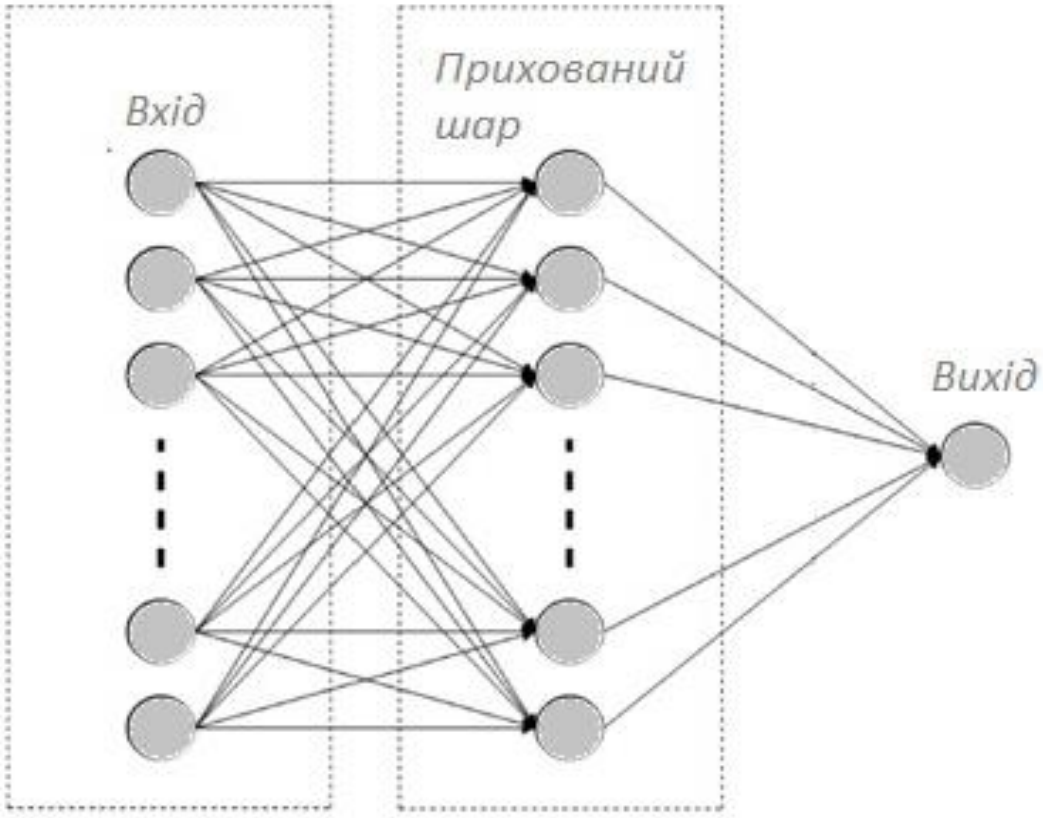

Рисунок 4.13 – Архітектура побудованої нейронної мережі

Таблиця 4.3 – Залежність помилок першого, другого роду та часу відпрацювання алгоритму від довжини тренувальної вибірки

Довжина вибірки	Помилка першого роду (%)	Помилка другого роду (%)	Час відпрацювання алгоритму (с)
10	10,89	11,47	1,32
25	10,02	10,42	2,68
50	9,65	10,12	4,96
100	8,11	9,25	8,78
200	6,34	8,81	17,01
500	4,77	5,33	33,43
1000	2,76	2, 98	59,96

Висновки за розділом 4

Для досягнення ефективного балансування навантаження в розподіленій обчислювальній системі важливо використовувати комплексний підхід. Традиційні алгоритми балансування навантаження в ізоляції не гарантують оптимальних результатів для підвищення продуктивності системи. Розроблений метод балансування поєднує кілька алгоритмів, акцентуючи увагу на внутрішньому моніторингу стану обчислювального вузла та мережі, яка об'єднує вузли системи. Врахування різнобарвності обчислювальної потужності компонентів через вагові коефіцієнти виявляється необхідним кроком для повноцінного балансування.

Аналіз трафіку, який надходить на вхід системи, вказує на наявність фрактальних властивостей, які визначають спосіб розподілу цього трафіку по серверах системи. Здійснено експеримент з методом балансування "Compare And Balance", де акцент робиться на стані завантаженості сервера, і результати відображаються на графіках у вигляді западин. Однак, важкість відрізнення результатів завантаженості серверів підкреслюється візуально.

Під час створення нейронної мережі для класифікації стану системи виявлено, що використання одного прихованого шару та певного типу функцій активації дозволяє досягти бажаних результатів. Врахування ознак, таких як завантаження CPU, об'єм оперативної пам'яті та характеристики трафіку, важливо для точної класифікації. Аналіз залежності помилок класифікації та часу відпрацювання алгоритму від довжини тренувальної вибірки вказує на потребу уважного підбору параметрів для досягнення оптимальних результатів.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи вирішувалась задача виявлення потенційно шкідливих запитів на сервер, використовуючи методи машинного навчання та нейронних мереж. Досліджено ряд алгоритмів та побудовано програмний класифікатор, спрямований на аналіз компонентів запитів та їх класифікацію за допомогою нейронної мережі. Отримані результати свідчать про високу точність класифікації, яка виявилася корисною для виявлення атак та впровадження у системи фільтрації.

Для оптимізації використання ресурсів та забезпечення відмово стійкості в розподіленій системі використовується метод балансування навантаження. Пропонована методика оцінки завантаження обчислювальних вузлів та комплексного дисбалансу в системі дозволяє ефективно розподіляти завдання та забезпечувати стійкість системи. Здійснено імітаційне моделювання алгоритмів балансування та моніторингу завантаженості серверів за різних вилах навантаження, що підкреслює важливість розуміння та ефективного вирішення проблем динамічної роботи розподіленої системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sheluhin O. I., Smolskiy S. M., Osin A. V. Similar processes in telecommunications. London : John Wiley & Sons Ltd, 2007. 306 p.

2. Сергєєв М. О. Рекомендаційні системи та математичні методи їх побудування. *27-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті»* : зб. матеріалів форуму. Т. 7. Харків : ХНУРЕ, 2023. С. 165-167.

3. Idhammad M., Afdel K., Belouch M. Detection System of HTTP DDoS Attacks in a Cloud Environment Based on Information Theoretic Entropy and Random Forest. *Security and Communication Networks*. 2018. V. 55, № 123. P. 1–13.

4. Deka R., Bhattacharyya D. Self-similarity based DDoS attack detection using Hurst parameter. *Security and Communication Networks*. 2016. V. 9, №. 17. P. 4468–4481.

5. Feldmann A., Gilbert A., Willinger W. Data networks as cascades: Investigating the multifractal nature of Internet WAN trac. *ACM Computer Communication Review*. 1998. № 28. P. 42–55.

6. Dadkhah M., Lyashenko V. V., Jazi M. D. Methodology of wavelet analysis in research of dynamics of phishing attacks. *International Journal of Advanced Intelligence Paradigms*. 2019. V. 12, № 4. P. 220–238.

7. Calvet L., Fisher A., Mandelbrot B. Large Deviationsand the Distribution of Price Changes. London : Cowles Foundation Discussion Paper, 1997. 30 p.

8. Al-Kasassbeh M., Al-Naymat G., Al-Hawari E. Towards Generating Realistic SNMP-MIB Dataset for Network Anomaly Detection. *International Journal Computer Science. Information Security*. 2016. V. 14, № 1. P. 1162–1185.

9. Doukhan P., Oppenheim G. Long Range Dependence: Theory and Applications. Bern : Birkhuser, 2002. 715 p.

10. Rao U. H., Nayak U. Intrusion Detection and Prevention Systems. *The InfoSec Handbook*. 2014. V. 3, № 104. P. 225–243.

11. Veneziano D., Moglen G., Bras R. Multifractal analysis: pitfalls of standard procedures and alternatives. *Phys. Rev. E*. 1995. V. 52, № 10. P. 1387–1398.

12. Suda H., Natsui M., Hanyu T. Systematic Intrusion Detection Technique for an In-vehicle Network Based on Time-Series Feature Extraction. *International Symposium on Multiple-Valued Logic*. 2018, V. 1, № 18. P. 56–61.

13. Kirichenko L., Radivilova T., Ryzhanov V. Applying Visibility Graphs to Classify Time Series. *Lecture Notes on Data Engineering and Communications Technologies*. 2022. № 77. P. 397–409.

14. Zhou W., Wen J., Li P. Abnormal Profiles Detection Based on Time Series and Target Item Analysis for Recommender Systems. *Mathematical Problems in Engineerin*. 2015. V.204, № 261. P. 1–9.

15. Kirichenko L., Radivilova T., Sydorenko B. Detection of Shoplifting on Video Using a Hybrid Network. *Computation.* 2022. V. 10, № 199. P. 28–31.

16. Pichugina O., Kirichenko L., Radivilova T. Binary classification: Ensemble Methods Utilizing Decision Theory Tools. *CEUR Workshop Proceedings*. 2022, № 3348. P. 19–33.

17. Kang J., Yang M., Zhang J. Accurately Identifying New QoS Violation Driven by High-Distributed Low-Rate Denial of Service Attacks Based on Multiple Observed Features" *Journal of Sensors*. 2015. V. 103, № 402. P. 1–11.

18. Kirichenko L., Radivilova T., Bulakh V. Machine Learning in Classification Time Series with Fractal Properties. *Journal of Electronics and Telecommunications*. 2019. V. 9, № 4. P. 1–13.

19. Kirichenko L., Radivilova T., Zinkevich I. Comparative Analysis of Conversion Series Forecasting in E-commerce Tasks. *Systems and Computing*. 2002. V. 689, № 13. P. 230–242.