Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-24М
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Маламуж В.С.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Коваленко А.С.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Маламуж В.С. Дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи управління у розподілених Cloud-системах GENI.

Метою розробки є дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI.

Об'єктом дослідження є процес управління у розподілених Cloud-системах GENI.

Предметом дослідження є методи управління у розподілених Cloud-системах GENI.

Методи дослідження базуються на методах хмарних обчислень, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи управління у розподілених Cloud-системах GENI.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерні науки, Cloud-системи, GENI

ABSTRACT

Malamuzh V.S. Research and software implementation of the management system in distributed Cloud-systems GENI. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed, which is intended for the management system in distributed Cloud-systems GENI.

The purpose of the development is the research and software implementation of the management system in distributed Cloud-systems GENI.

The object of the research is the management process in distributed Cloud-systems GENI.

The subject of the research is management methods in distributed Cloud-systems GENI.

The research methods are based on cloud computing methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of the management system in distributed Cloud-systems GENI.

In the process of working on the software model, an analysis of existing hardware and software tools was performed. All components of the developed software are fully described.

A user-friendly user interface has been developed. Instructions for working with the software are provided.

The program can be used on a PC with Windows 10/11.

The program is developed in the Python environment.

Keywords: computer science, Cloud systems, GENI

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

EOM	–	електронно-обчислювальна машина
ATM	–	Asynchronous Transfer Mode – асинхронний спосіб передачі даних
BER	–	Bit Error Rate – імовірність ушкодження одного біта
CBWFQ	–	class based weighted fair queueing
DiffServ	–	Differentiated Services – механізм який залежно від вимог до якості обслуговування записує в IP заголовки пакетів спеціальні мітки
DSCP	–	DiffServ CoSde Point
ICMP	–	Internet CoSntrol Message Protocol – міжмережний протокол управляючих повідомлень
ISP	–	Internet Service Provider – провайдер
LLQ	–	low latency queueing
MPLS-TE	–	Multiprotocol Label Switching Traffic Engineering
PQ	–	priority queueing
RIO	–	RED with Input Output
RTT	–	Round Trip Time – час між відправкою запиту та отриманням відповіді
STM	–	Synchronous Transfer Mode – синхронний спосіб передачі даних
QoS	–	Quality of Service – якість обслуговування
WFQ	–	Weighted Fair Queuing – механізм планування пакетних потоків даних
WRED	–	Weighted random early detection – взвішане значення довжини черги, у якості фактора, визначаючого ймовірність відкидання пакета

ВСТУП

Актуальність теми. GENI використовується для досліджень у сфері майбутніх інтернет-архітектур, програмно-визначених мереж, нових протокольних наборів, які можуть базуватися на IP, а можуть і не базуватися, бездротових мереж 4G та хмарних обчислень. GENI добре підходить для експериментів, які потребують:

– Масштабна експериментальна інфраструктура. GENI надає вам доступ до сотень широко розподілених ресурсів, включаючи обчислювальні ресурси, такі як віртуальні машини та «голі машини», а також мережеві ресурси, такі як з'єднання, програмовані комутатори та бездротові базові станції 4G.

– Не-IP-з'єднання між ресурсами. GENI дозволяє налаштовувати з'єднання 2-го рівня між комп'ютерними ресурсами та запускати власні протоколи 3-го рівня та вище.

– Глибокі можливості програмування. За допомогою GENI ви можете програмувати не лише кінцеві хости вашої експериментальної мережі, але й комутатори в ядрі вашої мережі. Це дозволяє експериментувати з новими протоколами мережевого рівня або з новими алгоритмами IP-маршрутизації.

– Інструменти та засоби вимірювання. Системи приладів та вимірювань GENI дозволяють вам інструментально аналізувати ваші експерименти, візуалізувати вимірювання та архівувати їх.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем управління у розподілених Cloud-системах GENI.

– Дослідження системи управління у розподілених Cloud-системах GENI.

– Програмна реалізація системи управління у розподілених Cloud-системах GENI.

Об’єктом дослідження є процес управління у розподілених Cloud-системах GENI.

Предметом дослідження є методи управління у розподілених Cloud-системах GENI.

Методи дослідження базуються на методах хмарних обчислень, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод управління у розподілених Cloud-системах GENI.

– Розроблено вітчизняний продукт управління у розподілених Cloud-системах GENI, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі управління у розподілених Cloud-системах GENI.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2025 р.), основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Стійки GENI мають кілька робочих вузлів, з'єднаних комутатором OpenFlow. Експериментатори можуть налаштовувати топології, використовуючи обчислювальні та мережеві ресурси, які повністю знаходяться в стійці. Вони можуть використовувати OpenFlow для керування трафіком на каналах, що проходять через комутатор стійки.

Це найпоширеніші обчислювальні ресурси GENI. Кожна стійка – це невелика хмара, яка надає експериментаторам доступ до віртуальних та базових машин. Усі стійки GENI мають IP-з'єднання через звичайний Інтернет для контролю та управління. Вони також підключаються на другому рівні до магістральної мережі GENI (національні та регіональні дослідницькі та освітні мережі), яка забезпечує програмовану площину даних.

1.2 Область застосування

Областю застосування є розподілене управління системами. Розподілені системи є основою незліченних програм, пропонуючи масштабованість та стійкість. Однак управління цими системами створює унікальні труднощі. Ефективне управління розподіленими системами є важливим для забезпечення надійності , продуктивності та безпеки. У цій роботі ми розглянемо основи, труднощі та управління розподіленими системами, що дозволяє організаціям використовувати їхній повний потенціал.

Управління розподіленою системою стосується процесу нагляду та контролю за роботою, конфігурацією та продуктивністю розподілених систем. Воно включає управління різними компонентами, вузлами та ресурсами, що

складають розподілену систему, для забезпечення її надійної, ефективної та безпечної роботи.

Важливість ефективного управління в розподіленій системі

Ефективне управління завжди має значення та забезпечує безперебійну роботу системи та першокласну продуктивність мереж. Це включає:

– Розподіл ресурсів:

- Розподілені системи часто складаються з численних взаємопов'язаних вузлів з різними обчислювальними ресурсами.
- Ефективне управління гарантує, що такі ресурси, як процесор, пам'ять і сховище, розподіляються ефективно для задоволення потреб робочого навантаження та уникнення вузьких місць.

– Балансування навантаження:

- Рівномірний розподіл вхідних запитів або завдань між вузлами допомагає запобігти перевантаженню певних вузлів і забезпечує оптимальне використання ресурсів.
- Ефективні механізми балансування навантаження динамічно регулюють розподіл ресурсів на основі поточного робочого навантаження та системних умов.

– Відмовостійкість:

- Розподілені системи схильні до збоїв через апаратні несправності, проблеми з мережею або програмні помилки.
- Ефективне управління включає впровадження відмовостійких механізмів, таких як резервування, реплікація та перехід на збій, для забезпечення стійкості системи та мінімізації часу простою.

– Масштабованість:

- Зі зростанням вимог до робочого навантаження розподілені системи повинні масштабуватися, щоб враховувати зростаючий трафік і обсяги даних.
- Ефективне управління включає проектування масштабованих архітектур та використання методів масштабування, таких як горизонтальне

масштабування (додавання більшої кількості вузлів) та вертикальне масштабування (оновлення ресурсів вузлів) для підтримки зростання без шкоди для продуктивності.

– Послідовність та координація:

- Підтримка узгодженості даних між розподіленими вузлами є складною через затримку мережі та одночасні оновлення.
- Ефективне управління включає впровадження моделей узгодженості, розподілених транзакцій та протоколів координації для забезпечення цілісності даних та координації між вузлами.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Ми живемо в епоху спроб знайти більш екологічні, безпечні рішення, які можуть зменшити шкоду, якої зазнає наша планета. Окрім того, що ми стаємо

Тенденція 2025 року щодо зберігання та передачі даних полягає в тому, що розподілене хмарне сховище є набагато екологічнішою альтернативою звичайному хмарному сховищу.

Але що відрізняє розподілене хмарне сховище від звичайного хмарного сховища?

У випадку розподіленого хмарного сховища швидкість передачі даних вища, ризик втрати даних зменшується, а мережа набагато менш перевантажена. Причина цього полягає в географічно розподіленій інфраструктурі, яка розміщує дані ближче до кінцевого користувача.

Окрім цих переваг, виникає ще одна важлива перевага. Розподілене хмарне сховище менш енергоємне, оскільки воно не споживає енергію для охолодження, на відміну від централізованого хмарного сховища. Використання розподіленого хмарного сховища допомагає значно зменшити споживання енергії в усьому світі.

Визначивши основну різницю між цими двома типами хмарних сховищ, ми пропонуємо 10 найкращих рекомендацій щодо постачальників розподілених хмарних сховищ, яких варто спробувати у 2025 році.

1. Swarm

Головна мета Swarm – децентралізація інтернету та підтримка інфраструктури для економіки даних майбутнього. Розподілена мережа Swarm забезпечує захист від витоків повідомлень завдяки своїй стійкій до цензури

конструкції.

Спосіб придбання сховища у Swarm досить унікальний. Swarm має власний токен під назвою BZZ, який можна обміняти на фіатну валюту або криптовалюту. Поточна ціна за 1 ГБ сховища на місяць становить 0,3271 BZZ.

Ключові характеристики Swarm:

– Криптографічна підтримка.

– Розподілений захист від відмови в обслуговуванні (DDoS).

– Відкритий вихідний код.

– Достовірний нейтралітет.

2. **Каббіт**

Cubbit – це георозподілений хмарний сервіс зберігання даних, надзвичайно стійкий та незалежний. Cubbit плавно інтегрується з вашою поточною конфігурацією S3, не турбуючись про порушення вашого робочого процесу.

Щоб випробувати Cubbit, ви можете скористатися 15-денною безкоштовною пробною версією, яку він пропонує. Для індивідуальних планів вам слід звернутися до експертів Cubbit. Доступні два варіанти: DS3 та DS3 Composer. З DS3 ви отримуєте індивідуальний план, що відповідає вашим потребам, тоді як DS3 Custom дозволяє вам створити власну георозподілену хмару.

Основні характеристики Cubbit:

– Захист від програм-вимагачів.

– Сумісність S3.

– Без обмежень щодо сховища.

– Цифровий суверенітет.

3. **Хмара Google**

Google Cloud стверджує, що здійснив революцію у функціонуванні хмарних сховищ. Пропонуючи безпеку рівня Google, Google Cloud побудований на основі генеративного штучного інтелекту та розроблений для ШІ та

розробників.

Що стосується ціноутворення, Google Cloud має прозорий підхід, тому ви не платите за те, чим не користувалися. Він має калькулятор цін і пропонує автоматичне збереження на основі вашого щомісячного використання.

Основні характеристики Google Cloud:

– Гнучкість інфраструктури.

– Захист від DDoS-атак.

– Інструменти моніторингу та управління.

– Гібридна платформа.

4. **Microsoft Azure**

Одним з головних напрямків Microsoft Azure є надання хмарного сховища даних, що базується на екологічно чистих технологіях. Таким чином, разом з Azure ваша компанія може зробити свій внесок у майбутнє чистої енергії. Більше того, Azure адаптується до потреб вашого бізнесу – залежно від того, чи потрібно вам більше чи менше сховища чи пропускної здатності, Azure вам підійде.

Що стосується ціноутворення, Microsoft Azure надає калькулятор цін для оцінки вартості сховища. Він також має політику оплати за використанням, тому ви платите лише за те, що використали – без жодних попередніх зобов'язань.

Основні характеристики Microsoft Azure:

– Масштабованість.

– Безпечне сховище.

– Розширена аналітика.

– Різноманітне оброблення.

5. **Амазон S3**

За допомогою Amazon S3 ви можете легко переміщувати, зберігати та аналізувати дані, пропонуючи водночас провідну в галузі масштабованість, продуктивність та безпеку. Це чудовий варіант для великого бізнесу, оскільки він надає такі функції, як S3 buckets, S3 inventory або S3 packet operations. Ці функції дозволяють ефективно керувати даними у більшому масштабі.

Amazon S3 стягує плату за зберігання даних у buckets. Ціна розраховується на основі розміру, періоду зберігання та класу зберігання. Вони пропонують вісім класів зберігання на вибір, і ціни за ГБ знижуються, якщо ви обираєте більшу ємність сховища.

Основні характеристики Amazon S3:

– Високий рівень безпеки.

– Можливості аудиту.

– Надійний контроль доступу.

– Видимість на організаційному рівні.

6. **Інфраструктура хмари Oracle (OCI)**

OCI пропонує понад 100 послуг, для яких ви отримуєте однакову платформу, досвід і, тим не менш, ціни. Розподілена хмара OCI є гнучкою, тому ви можете вибрати, як і де ви хочете, щоб їхні послуги надавалися. Існує чотири типи хмарних послуг, які OCI може надати своїм клієнтам: мультихмара, публічна хмара, гібридна хмара та виділена хмара.

Ціни OCI конкурентні та розроблені саме для вас. Як і Google, Microsoft та Amazon, OCI надає калькулятор цін. Ви також можете отримати винагороду за використання OCI, якщо ваше підприємство перейде на Oracle.

Ключові характеристики OCI:

– Гнучкі обчислення.

– Автоматичне налаштування пам'яті.

– Моніторинг послуг.

– Високий рівень безпеки.

7. Cloudian

Ще одним чудовим постачальником хмарних послуг для великих підприємств є Cloudian. Він здатний збирати та керувати величезними обсягами даних, пропонуючи аналіз даних у режимі реального часу. Таким чином, Cloudian допомагає підтримувати зростання вашого бізнесу. Максимізація продуктивності підтримується георозподіленою архітектурою Cloudian та штучним інтелектом.

Cloudian пропонує два способи доступу до інформації про ціни. Ви можете або отримати ціни самостійного обслуговування, які надсилаються на вашу електронну пошту після заповнення форми, або запросити цінову пропозицію, після чого вам зателефонує команда з продажу Cloudian.

Основні характеристики Cloudian:

– Суверенітет даних.

– Спостережуваність даних.

– Управління даними зі штучним інтелектом.

– Безпека хакерського шифрування.

8. **F5**

Безпека, захист від шахрайства та ризиків, багатохмарні мережі, продуктивність та надійність – саме так F5 описує свої розподілені хмарні сервіси. Їхні сервіси призначені для сучасних підприємств, щоб вони могли скористатися перевагами публічного або приватного хмарного сховища.

Щоб скористатися F5, вам потрібно подати запит на пробний період, і з вами зв’яжуться. Ви також можете запланувати демонстрацію з експертом.

Основні характеристики F5:

– Запобігання шахрайству та зловживанням.

– Продуктивність застосунків та мережі.

– Запобігання несанкціонованому доступу.

– Захист інтерфейсу прикладного програмування (API).

9. **Nutanix**

Nutanix – чудовий постачальник, якого варто розглянути, якщо ви шукаєте хмару, яка проста у використанні. Пропозиція Nutanix складається з єдиної платформи, з якої ви можете запускати та керувати своїми даними. З Nutanix хмара є одночасно спрощеною та економічно ефективною.

Щоб побачити, як працює платформа Nutanix, ви можете спробувати її. Вам просто потрібно заповнити форму з вашою інформацією та інформацією вашого підприємства, і з вами зв’яжуться телефоном або електронною поштою.

Основні характеристики Nutanix:

– Уніфіковане управління платформою.

– Автоматизація на основі штучного інтелекту.

– Операції одним клацанням миші.

– Реархітектура та переобладнання.

10. Elastic Cloud Storage (ECS)

ECS має простішу архітектуру хмарного сховища порівняно з іншими постачальниками послуг. Однак, вона може керувати мільярдами і навіть трильйонами файлів за низькою ціною та без ризику. ECS має багатосайтову архітектуру, яка дозволяє доступ з будь-якого пристрою незалежно від вашого місцезнаходження.

Щоб дізнатися ціни та отримати додаткову інформацію про продукт, вам потрібно запросити цінову пропозицію, заповнивши форму. Вам потрібно чекати максимум 24 години, щоб отримати відповідь, або ви можете зателефонувати безпосередньо до ECS.

Основні характеристики ECS:

– Універсальна доступність.

– Масштабованість.

– Аналіз даних.

– Сховище, доступне через API.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – динамічна інтерпретована об’єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Офіційний сайт мови програмування Python https://www.python.org/. Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові. Python – багатоплатформова мова

програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо. Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Python абсолютно безкоштовний.

Швидкість виконання коду Python

Один з можливих недоліків Python – швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C.

Втім, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості застосунків швидкість розробки важливіша швидкості виконання, а програми на Python зазвичай пишуться набагато швидше.

Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор.

Використання Python

Python використовується для різних цілей: для створення ігор і веб-застосунків, розробки внутрішніх інструментів для різноманітних проектів. Мова також широко застосовується в науковій області для досліджень і розв'язування прикладних завдань.

Застосування мови програмування Python:

1. BitTorrent – протокол для обміну даними.

2. Ubuntu Software Center – вільне програмне забезпечення для пошуку, установки і видалення пакунків в системі Ubuntu Linux.

3. Blender – програма для створення тривимірної комп’ютерної графіки, що включає засоби моделювання, анімації, вимальовування, пост-обробки відео, а також створення відеоігор.

4. GIMP – растровий графічний редактор, із підтримкою векторної графіки.

5. World of Tanks.

6. Вільна енциклопедія Вікіпедія.

7. Пошукова система Google.

8. DropBox – файловий хостинг, що включає персональне хмарне сховище, синхронізацію файлів і програму-клієнт.

9. YouTube – популярне відеосховище.

Версії Python

Мови програмування з часом змінюються – розробники додають в них нові можливості, а також виправляють помилки. Так з’являються різні версії мови. Наприклад, код написаний на Python 2 у більшості випадків не буде працювати у версії Python 3 без внесення додаткових змін.

Процесор є найважливішим компонентом в комп’ютері. Одна з основних функцій процесора – це обробка даних згідно комп’ютерної програми, яка є списком інструкцій, шляхом виконання арифметичних і логічних операцій над фрагментами даних.

Кожна інструкція в програмі – це команда, яка «повідомляє» процесору, яку операцію він повинен виконати. Процесор комп’ютера може розуміти лише ті інструкції, які написані на машинній мові. Машинна мова – це штучна мова, створена для передачі команд комп’ютеру. За допомогою машинної мови створюються ефективні програми, оскільки розробник отримує доступ до всіх можливостей процесора. Машинна мова – мова низького рівня.

Інструкція машинної мови існує для кожної операції, яку процесор здатний виконати – є інструкція для додавання чисел, є інструкція для віднімання чисел і т.д. Увесь набір інструкцій, який центральний процесор може виконати, відомий як набір інструкцій процесора.

Наприклад, у вас є певна програма, яка зберігається на диску вашого комп'ютера. Для виконання програми, ви здійснюєте подвійний клік на значку програми. Це змушує програму копіюватися з диска в оперативну пам'ять, після чого процесор комп'ютера виконує копію програми, яка знаходиться в оперативній пам'яті.

Коли процесор виконує інструкції програми, він бере участь у процесі, який є відомим як цикл fetch – decode – execute (отримати – декодувати – виконати). Цей цикл виконується для кожної інструкції у програмі і складається з трьох кроків:

Отримати

Програма – це послідовність інструкцій на машинній мові. Першим кроком циклу є завантаження (отримання) наступної інструкції з пам'яті в процесор.

Декодувати

Інструкція машинної мови – це двійкове число, яке представляє команду, що повідомляє процесору виконати певну операцію. На цьому кроці процесор декодує інструкцію, яку було «витягнуто» з пам'яті, для визначення того, яка операція повинна виконуватись.

Виконати

Останній крок циклу – виконати операцію.

Хоча процесор комп'ютера розуміє тільки машинну мову, людині непрактично писати програми на машинній мові. Така програма може мати тисячі або навіть мільйони бінарних інструкцій, і написання такої програми буде дуже обтяжливим процесом.

З цієї причини була створена мова асемблера як альтернатива машинній мові. Замість використання двійкових чисел для написання інструкцій, мова асемблера використовує короткі слова, відомі як мнемокоди.

Незважаючи на те, що мова асемблера не вимагає двійкових інструкцій, як у випадку машинної мови, проте вона вимагає високих знань про процесор.

Використовуючи мову асемблера, навіть для найпростішої програми, необхідно написати велику кількість інструкцій.

Мова програмування високого рівня дозволяє створювати складні програми, не знаючи, як працює процесор, і не записуючи великої кількості інструкцій низького рівня. Крім того, більшість мов програмування високого рівня використовують слова, які легко зрозуміти.

Python – одна із популярних сучасних мов програмування високого рівня. Python – інтерпретована мова програмування. Python – це високорівнева інтерпретована мова програмування, на відміну від C++, яка є прикладом компільованої мови програмування. Назва Python відноситься як до мови програмування, так і до інтерпретатора – комп'ютерної програми, яка зчитує початковий код (написаний на Python) і виконує інструкції (команди).

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

1. Компілятор.

2. Інтерпретатор.

Завантаження Python

Версії інтерпретатора Python для різних операційних систем доступні для безкоштовного завантаження за адресою https://www.python.org/downloads.

Середовище програмування для Python

Для написання програм використовують текстові редактори або інтегровані середовища розробки, які включають в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо.

Текстові редактори та інтегровані середовища програмування для Python:

– IDLE – стандартний редактор Python. Встановлюється разом з Python для користувачів Windows, окремим пакунком для користувачів Linux.

– Notepad++ – безкоштовний текстовий редактор початкового коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

– Visual Studio Code – це легкий, але потужний редактор початкового коду, який розповсюджується безкоштовно і доступний у версіях для платформ Linux, Windows і macOS.

– PyScripter – інтегроване середовище розробки для мови програмування Python. Для користувачів Windows. Поширюється безкоштовно.

– Wing IDE 101 – вільне інтегроване середовище для Python, розроблене для навчання програмістів-початківців. Для користувачів Linux, Windows і macOS. Поширюється безкоштовно.

– Geany – вільний текстовий редактор з базовими елементами інтегрованого середовища розробки, доступний для операційних систем Linux, Windows і macOS.

– PyCharm – інтегроване середовище розробки для мови програмування Python. PyCharm є власницьким програмним забезпеченням. Наявна безкоштовна версія Community з усіченим набором можливостей. Для користувачів Linux, Windows і macOS.

– Thonny – IDE для вивчення програмування мовою Python. Для користувачів Linux, Windows і macOS.

– Mu – редактор коду Python для програмістів-початківців. Для користувачів Linux, Windows і macOS.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи управління у розподілених Cloud-системах GENI.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Проблеми управління розподіленими системами

Керування розподіленими системами створює кілька проблем через їхню притаманну складність та розподілену природу. Керування розподіленими системами створює кілька проблем через їхню притаманну складність та розподілену природу.

– Складність:

- Впровадження розподілених систем вимагає великої кількості самодостатніх компонентів, які є взаємопов'язаними та мають власну конфігурацію, залежності та протоколи зв'язку.
- В цифрову епоху, коли правильне виконання цих частин та ефективне управління ними може бути болісним завданням,...

– Консистенція:

- З іншого боку, найскладнішим питанням є досягнення узгодженості даних між розподіленими вузлами, особливо для обробки одночасних оновлень та збоїв.
- Це вимагає вдосконалених методів координації та синхронізації.

– Безпека:

- Гарантія того, що розподілені системи не будуть зламані, а неавторизовані особи не отримають до них доступу за допомогою надійних інструментів безпеки, є важливим заходом і потребує шифрування, автентифікації та контролю доступу для всіх зацікавлених сторін.

Централізоване проти децентралізованого управління

1. Централізоване управління

У централізованому управлінні всі повноваження щодо контролю та

прийняття рішень зосереджені в межах однієї сутності або центрального органу. Ця сутність зазвичай має повноваження приймати рішення, розподіляти ресурси та забезпечувати дотримання політик у всій системі.

– Повноваження щодо прийняття рішень належать центральному органу влади або керівному органу.

– Політики, правила та процедури встановлюються та застосовуються централізовано.

– Розподіл ресурсів, координація та контроль здійснюються з однієї точки.

– Комунікація та потік інформації зазвичай спрямовуються через центральний орган влади.

2. Децентралізоване управління

У децентралізованому управлінні повноваження щодо прийняття рішень, контроль та координація розподілені між кількома об'єктами або вузлами в системі. Кожен об'єкт або вузол працює автономно та має право самостійно приймати рішення в межах своєї сфери відповідальності.

– Повноваження щодо прийняття рішень розподілені між кількома сутностями або вузлами.

– Сутності або вузли мають автономію приймати рішення та діяти самостійно.

– Координація та комунікація відбуваються безпосередньо між сутностями або вузлами без централізованого контролю.

Моніторинг розподілених систем

Спостереження за розподіленими системами є ключовою функцією; воно має забезпечувати відстеження відхилень, оптимізацію роботи та усунення помилок. Ключові показники для моніторингу включають:

– **Пропускна здатність:** Вимірює час відгуку окремо як показник загальних можливостей системи упаковувати та поміщати транзакції в чергу за раз.

– **Затримка:** З іншого боку, вона позначає період часу, необхідний для

виконання запиту, що може бути ознакою того, наскільки швидко працює система.

– **Використання ресурсів:** Відстежує використання процесора, а також пам'яті, сховища та пропускної здатності мережі. Таким чином, немає можливості затримки, а ресурси розподіляються ефективно.

– **Коефіцієнти помилок:** Він також досліджує, як часто виникають помилки та збої, пропонуючи вам уявлення про регулярність та надійність систем.

Реєстрація та трасування в розподілених системах

Ведення журналу та трасування є важливими компонентами моніторингу та налагодження розподілених систем, допомагаючи розробникам та адміністраторам розуміти поведінку системи, діагностувати проблеми та оптимізувати продуктивність.

1. Вхід у розподілених системах

Ведення журналу включає запис подій, повідомлень та інформації про системні операції, помилки та дії на постійне сховище (наприклад, файли, бази даних або системи керування журналами). Ведення журналу забезпечує історичний запис поведінки системи, допомагаючи розробникам та адміністраторам відстежувати потік виконання, виявляти помилки або аномалії та усувати неполадки.

– Повідомлення журналу зазвичай містять позначки часу, рівні серйозності (наприклад, INFO, DEBUG, WARN, ERROR), контекстну інформацію (наприклад, назву компонента, ідентифікатор запиту) та описові деталі (наприклад, трасування стека помилок, параметри запиту).

– Розробники використовують ведення журналу для моніторингу стану програм, відстеження потоку програм, налагодження проблем, аудиту дій користувачів та дотримання нормативних вимог.

2. Трасування в розподілених системах

Трасування включає фіксацію та кореляцію розподілених транзакцій або запитів, коли вони поширюються через різні компоненти чи служби в

розподіленій системі. Трасування допомагає розробникам та адміністраторам зрозуміти наскрізний потік запитів між розподіленими компонентами, виявити вузькі місця в продуктивності та проаналізувати затримку та залежності.

– Трасування складаються з серії корельованих інтервалів, кожен з яких представляє певну операцію або дію в межах компонента чи сервісу. Інтервали містять метадані, такі як назви операцій, позначки часу початку та завершення, а також контекстну інформацію (наприклад, назву сервісу, ідентифікатор запиту).

– Розробники використовують трасування для візуалізації потоків запитів, вимірювання показників продуктивності на рівні сервісів (наприклад, часу відгуку, пропускної здатності), аналізу залежностей між сервісами та оптимізації продуктивності розподіленої системи.

Управління конфігурацією в розподілених системах

Управління конфігурацією охоплює процеси, інструменти та методи для визначення, розгортання, оновлення та моніторингу параметрів конфігурації та ресурсів розподілених систем. Деякі з цілей управління конфігурацією включають:

– Забезпечення узгодженості: управління конфігурацією гарантує, що всі вузли або компоненти в розподіленій системі мають узгоджені налаштування конфігурації, запобігаючи дрейфу конфігурації та невідповідностям.

– Покращення масштабованості: автоматизуючи завдання управління конфігурацією, розподілені системи можуть масштабуватися ефективніше, що дозволяє швидко розгортати та надавати нові вузли або ресурси.

– Підвищення надійності: Підтримуючи стандартизовану конфігурацію та впроваджуючи найкращі практики, управління конфігурацією допомагає підвищити надійність і стабільність розподілених систем.

– Спрощення управління змінами: управління конфігурацією дозволяє систематично відстежувати, версіонувати та аудит змін конфігурації, що спрощує керування та відкат змін за потреби.

Розподіл ресурсів у розподілених системах

Розподіл ресурсів передбачає визначення того, як розподілити доступні ресурси між конкуруючими завданнями або роботами в розподіленій системі для оптимізації продуктивності, використання та справедливості.

– Оптимізація продуктивності: розподіл ресурсів таким чином, щоб максимізувати пропускну здатність системи, мінімізувати час відгуку та відповідати вимогам якості обслуговування (QoS).

– Забезпечення справедливості: розподіл ресурсів справедливо між конкуруючими завданнями або користувачами, щоб запобігти їх дефіциту та сприяти рівному доступу до ресурсів.

Деякі з підходів до розподілу ресурсів у розподілених системах включають:

– Статичне розподілення:

- Попередньо розподіляйте ресурси між завданнями або роботами на основі попередньо визначених політик, пріоритетів або квот.
- Цей підхід підходить для передбачуваних робочих навантажень з фіксованими вимогами до ресурсів.

– Динамічне розподілення:

- Динамічно налаштовуйте розподіл ресурсів на основі характеристик робочого навантаження, системних умов та показників продуктивності.
- Такі методи, як балансування навантаження, автоматичне масштабування та адаптивне виділення ресурсів, використовуються для коригування розподілу ресурсів у режимі реального часу.

– Розподіл кількох ресурсів:

- Розподіляючи ресурси між завданнями або роботами, враховуйте одночасно кілька ресурсів (наприклад, процесор, пам'ять і диск), враховуючи взаємозалежність та обмеження ресурсів.

Планування в розподілених системах

Планування включає визначення часу і місця виконання завдань на

розподілених вузлах для досягнення цільових показників продуктивності, обмежень ресурсів та системних вимог.

– Мінімізація затримки: Плануйте завдання або роботи, щоб мінімізувати час очікування, час відгуку та затримки обробки, покращуючи швидкість реагування системи та взаємодію з користувачем.

– Максимізація пропускної здатності: Планування завдань або проектів для максимізації пропускної здатності системи та обчислювальної потужності, забезпечення ефективного використання доступних ресурсів.

Деякі з підходів до розподілу ресурсів у розподілених системах включають:

– Планування на рівні завдань:

- Плануйте окремі завдання або роботи на основі пріоритету, термінів, залежностей та потреб у ресурсах.
- Такі методи, як планування на основі пріоритетів, планування за термінами та планування з урахуванням залежностей, використовуються для оптимізації виконання завдань.

– Пакетне планування:

- Плануйте групи пов'язаних завдань або проектів (наприклад, завдання пакетної обробки, завдання MapReduce), щоб оптимізувати використання ресурсів та мінімізувати час виконання завдань.

– Глобальне планування:

- Координуйте рішення щодо планування між кількома вузлами або кластерами для оптимізації продуктивності всієї системи та розподілу ресурсів.
- Такі методи, як глобальне балансування навантаження, розподілені алгоритми планування та централізовані планувальники, використовуються для координації рішень щодо планування в розподілених середовищах.

Виявлення та відновлення несправностей у розподілених системах

Інструмент виявлення дефектів – це механізм виявлення несправностей, який допомагає визначити стан розподілених компонентів і, таким чином, швидко

виявляти збої та реагувати на них. Вбудовані механізми відновлення, які можуть обробляти реплікацію, резервування та відновлення після відмови в системі, забезпечують безперервність та стійкість обслуговування з метою мінімізації простоїв або перебоїв у роботі.

1. Підходи до виявлення несправностей

– Моніторинг на основі серцебиття: Вузли періодично надсилають повідомлення про серцебиття, щоб вказати свою доступність та стан. Вузли моніторингу виявляють несправності, аналізуючи повідомлення про серцебиття та виявляючи відхилення від очікуваних шаблонів.

– Детектори відмов: Розподілені алгоритми та протоколи використовують детектори відмов для виявлення відмов вузлів або збоїв на основі спостережуваної поведінки, тайм-аутів повідомлень або збоїв зв'язку.

– Виявлення аномалій: Машинне навчання та статистичні методи використовуються для виявлення аномалій або незвичайних закономірностей у системних метриках, даних про продуктивність або поведінці зв'язку, що вказує на потенційні несправності або збої.

2. Підходи до відновлення після несправностей

– Резервування та реплікація: Використовуйте методи резервування та реплікації для реплікації критично важливих даних або послуг на кількох вузлах або в центрах обробки даних. У разі збою резервні копії можуть бути активовані для підтримки доступності послуг.

– Відкат та контрольні точки: Механізми відкату та методи контрольних точок дозволяють системі повернутися до попереднього відомого справного стану до виникнення помилки. Транзакції або процеси можна відкатити до узгодженого стану, а контрольні точки можна використовувати для відновлення обробки з відомої точки.

– Реконфігурація та самовідновлення: механізми самовідновлення автоматично переконфігурують систему, перерозподіляють робоче навантаження або замінюють компоненти, що вийшли з ладу, для відновлення функціональності

системи. Для автоматизації процесів відновлення використовуються такі методи, як автоматичне масштабування, динамічна реконфігурація та автоматичне перемикання на резервний ПК.

Безпека та контроль доступу в розподілених системах

Заходи безпеки, включаючи використання шифрування, автентифікації та авторизації, значною мірою допомагають гарантувати безпеку розподіленої системи від кібератак та несанкціонованого доступу. Заходи контролю доступу – це засоби забезпечення дотримання політики щодо заборони доступу до деяких конфіденційних ресурсів і даних. Це гарантує конфіденційність даних та дотримання чинних правил.

Масштабування та балансування навантаження в розподілених системах

Масштабування означає або динамічний приплив, або зменшення ресурсів, що збігається зі збільшенням або зменшенням використання цих ресурсів.

– Балансування навантаження служить для рівномірного розподілу вхідних запитів до кожного вузла розподіленої системи, що сприяє ефективному використанню ресурсів та зменшує перевантаження, тим самим забезпечуючи масштабованість, високу доступність та продуктивність у розподілених системах.

– Такі методи, як горизонтальне та вертикальне масштабування , а також автоматичне масштабування, є одними зі способів, за допомогою яких розподілені системи масштабуються. Ці методи використовуються для розподілу робочого навантаження на основі доступності ресурсів та моделей попиту.

3.2 Розробка структурної схеми

Ключові характеристики розподілених систем

1. Масштабованість: Розподілені системи можуть масштабуватися горизонтально (додаючи більше машин), а не вертикально (збільшуючи потужність однієї машини), що робить їх ідеальними для застосувань, які

потребують швидкого зростання.

2. Паралелізм: Багато розподілених систем обробляють тисячі або навіть мільйони запитів одночасно. Це призводить до потреби в ефективних протоколах зв'язку та обробці спільних ресурсів.

3. Відмовостійкість: оскільки розподілені системи працюють на кількох машинах, вони розроблені таким чином, щоб переносити збої окремих вузлів, не впливаючи на доступність усієї системи.

4. Гетерогенність: Розподілені системи можуть складатися з різних типів машин, операційних систем або мереж, що означає, що вони повинні безперешкодно обробляти цю різноманітність.

5. Прозорість: Розподілені системи намагаються приховати складність кількох машин від кінцевого користувача, забезпечуючи єдиний інтерфейс, ніби система працює на одному комп'ютері.

Типи розподілених систем

1. Клієнт-серверні системи

У цій моделі сервер надає послуги, а клієнти споживають ці послуги, надсилаючи запити до сервера. Прикладами є веб-сервери та сервери баз даних. Приклад: Браузер (клієнт) надсилає HTTP-запит на веб-сервер, а сервер відповідає запитуваними даними (веб-сторінкою, відповіддю API тощо).

2. Однорангові системи

На відміну від клієнт-серверних систем, однорангові системи не мають централізованого сервера. Натомість кожен вузол (або одноранговий вузол) може виступати як клієнтом, так і сервером. Приклад: Системи обміну файлами, такі як BitTorrent, де кожен вузол завантажує та вивантажує файли з інших вузлів.

3. Кластерні обчислення

Кластер складається з групи комп'ютерів (часто званих вузлами) **,** які працюють разом як єдина система для виконання великомасштабних обчислень. Вузли в кластері зазвичай розташовані в одному фізичному місці та з'єднані через високошвидкісну локальну мережу. Приклад: кластери Hadoop, що

використовуються для обробки великих даних.

4. Грід-обчислення

Грід-обчислення схожі на кластерні обчислення, але в більшому масштабі. Вузли в грід-обчисленнях часто географічно розподілені по різних місцях і з'єднані через Інтернет. Приклад: SETI@home, проект з мережевих обчислень, який здійснює пошук позаземного розуму шляхом аналізу радіосигналів з космосу.

5. Хмарні обчислення

У хмарних обчисленнях ресурси (обчислювальні, сховища, мережеві) надаються як послуга через Інтернет. Хмарні системи – це розподілені системи, що забезпечують гнучкість, масштабованість та моделі ціноутворення «оплата за використання». Приклад: Amazon Web Services (AWS), Microsoft Azure та Google Cloud.

Ключові проблеми розподілених систем

1. Затримка

Коли вузли в розподіленій системі розташовані в різних географічних місцях, мережеві затримки (латентність) стають серйозною проблемою. Латентність впливає на швидкість зв'язку між вузлами, що може вплинути на продуктивність.

2. Послідовність

Узгодженість означає, що всі вузли мають однаковий вигляд даних у будь-який момент часу. У розподілених системах підтримка узгодженості може бути складною, особливо коли вузли реплікуються в різних регіонах.

– Теорема CAP: Теорема CAP стверджує, що розподілена система не може одночасно забезпечити всі три наступні гарантії:

– Узгодженість: кожне зчитування отримує останню записану інформацію або помилку.

– Доступність: На кожен запит надходить відповідь без гарантії, що вона містить найновіший запис.

– Допуск розділу: Система продовжує працювати, незважаючи на випадкову втрату повідомлень або збій частини системи.

– Розподіленим системам часто доводиться йти на компроміси між цими властивостями.

3. Відмовостійкість

Вузли в розподіленій системі можуть вийти з ладу з різних причин (збій обладнання, проблеми з мережею тощо). Система повинна мати можливість виявляти такі збої та відновлюватися після них без втрати даних або послуг.

4. Безпека

Розподілені системи стикаються з проблемами безпеки, оскільки дані передаються через мережі, які можуть бути вразливими до таких атак, як перехоплення даних, атаки типу «людина посередині» або несанкціонований доступ.

5. Координація та синхронізація

У розподілених системах координація між вузлами є критично важливою. Вузлам часто потрібно узгоджувати стан системи, обробляти одночасні операції та синхронізувати спільні ресурси.

Проектування розподіленої системи

Під час проектування розподіленої системи слід дотримуватися певних принципів та найкращих практик:

1. Реплікація: Для покращення відмовостійкості та доступності дані та послуги слід реплікувати на кількох вузлах.

2. Балансування навантаження: балансувальник навантаження повинен розподіляти вхідні запити між кількома вузлами, щоб запобігти перевантаженню будь-якого вузла.

3. Шардінг: У великих системах дані можна розділити на менші частини (шарди), які розподіляються по різних вузлах. Це покращує продуктивність, дозволяючи паралельну обробку.

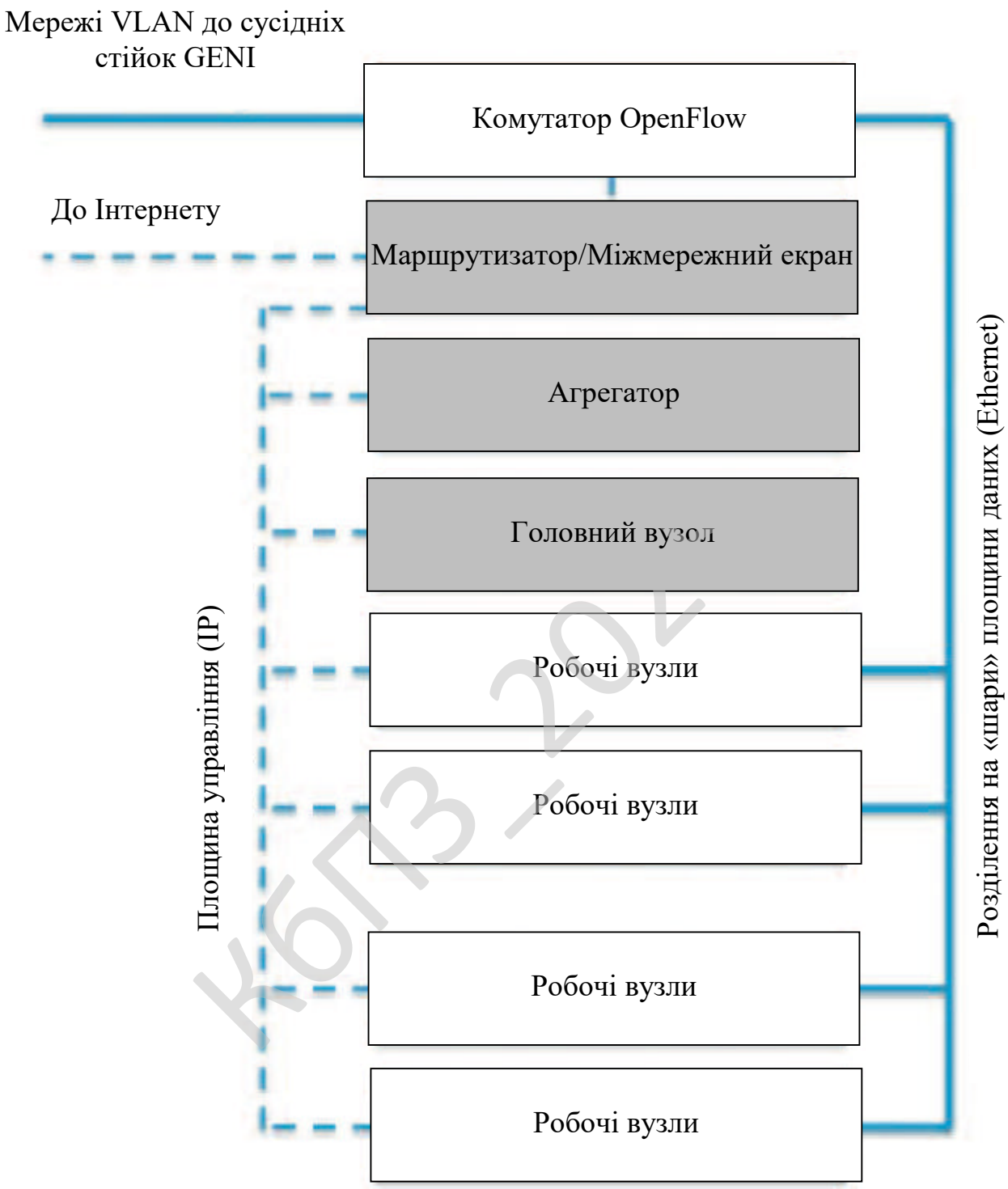

Рисунок 3.1 – Структурна схема системи

4. Кешування: Дані, до яких часто звертаються, слід кешувати, щоб зменшити затримку та покращити продуктивність системи.

5. Моніторинг та спостережуваність: Впровадження надійних інструментів моніторингу забезпечує справність системи та допомагає виявляти збої на ранній стадії.

Розподілені системи є основою сучасних застосунків, що дозволяє їм масштабуватися, обробляти збої та задовольняти потреби мільйонів користувачів. Вони стали важливими для підприємств, яким потрібна висока доступність, відмовостійкість та масштабованість. Однак проектування та управління розподіленими системами пов'язане з власним набором проблем, таких як затримка, узгодженість та відмовостійкість.

Як інженери-програмісти, розуміння принципів та проблем розподілених систем може значно покращити вашу здатність проектувати надійні, масштабовані та ефективні системи. Незалежно від того, чи створюєте ви мікросервіси, хмарні платформи чи великомасштабні конвеєри обробки даних, розподілені системи відіграватимуть вирішальну роль у вашій архітектурі.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2.

З рисунку видно, що розроблена система складається з наступних функціональних частин:

– Блок інтерфейсу користувача.

– Блок визначення параметрів QoS.

– Блок примусового завдання параметрів QoS.

– Блок моделювання завантаженості мережі системи управління у розподілених Cloud-системах GENI.

– Блок моніторингу мережі системи управління у розподілених Cloud-системах GENI.

– Блок дослідження можливостей механізмів WRED.

– Блок дослідження можливостей механізмів WFQ.

– Блок призначення пріоритетів.

– Блок організації та обслуговування черг.

– Блок управління навантаженням.

– Блок формування трафіка.

Розглянемо ці блоки більш детально.

Існує не занадто багато способів розрахунків показників QoS у системи управління у розподілених Cloud-системах GENI. Найпростіший з них – збільшення смуги пропущення мережі системи управління у розподілених Cloud-системах GENI за рахунок нарощування апаратних можливостей устаткування мережі системи управління у розподілених Cloud-системах GENI. Можна використовувати й такі прийоми, як завдання пріоритетів даних, організація черг, запобігання перевантажень і формування трафіку. Керування мережею за заданими правилами в перспективі повинне об'єднати всі ці способи в єдину автоматизовану систему, що буде гарантувати якість послуг абсолютно на всіх ділянках мережі.

Збільшення апаратної потужності, безсумнівно, є найбільш ефективним засобом реалізації QoS у мережі системи управління у розподілених Cloud-системах GENI. Тиск із боку конкурентів, необхідність підвищення ефективності виробництва, поява нових технологій, що дозволяють оснащувати спеціалізовані мікросхеми (ASIC) найрізноманітнішими функціями, – все це змушує постачальників комутаційного встаткування для локальних мереж викидати на ринок усе більше швидкодіючі пристрої за цінами, порівнянним з вартістю моделей колишнього покоління.

Малоймовірно, що в доступному для огляду майбутньому даний підхід до підтримки QoS у мережах системи управління у розподілених Cloud-системах GENI перестане бути пріоритетним. Оскільки в мережах системи управління у розподілених Cloud-системах GENI вдається забезпечити гарантовану якість послуг, не прибігаючи до дорогої модернізації усього встаткування й серйозних змін у системі керування мережею, мережні адміністратори будуть звертати увагу й на програмні засоби, що дозволяють реалізувати QoS.

Отже, найбільше поширення, швидше за все, одержить комбінований підхід. Деякі виробники висловлюються на його користь, затверджуючи, що найкраще збільшувати пропускну здатність мережі не прямо, а за рахунок інтелектуальних можливостей устаткування, що має засоби розрахунків показників QoS. Правда, виробники мережних пристроїв навряд чи можуть бути об'єктивними в цьому питанні, тому що вони зацікавлені в збуті тих самих продуктів, які підтримують гарантовану якість послуг.

У глобальних мережах нарощування апаратних потужностей використовується рідше. Звичайно, зниження вартості смуги пропущення зробило б передачу даних по глобальних мережах доступною для більше широкого кола користувачів (і навіть трохи знизило б актуальність впровадження гарантованої якості послуг). Але в найближчому майбутньому вартість смуги пропущення в глобальних мережах буде залишатися досить високою, тому й нарощування апаратної потужності не стане настільки популярним, як у мережах системи управління у розподілених Cloud-системах GENI.

Блок інтерфейсу користувача

Призначений для реалізації взаємодії користувача, або дослідника з системою.

Блок організації та обслуговування черг

Після того як переданим по мережі даним призначені відповідні пріоритети (за допомогою явних або неявних методів), потрібно визначити порядок передачі цих даних, задавши алгоритм обслуговування черг із необхідною якістю (рівнем QoS). По суті, черги являють собою області пам'яті комутатора або маршрутизатора, у яких групуються пакети з однаковими пріоритетами передачі. Алгоритм обслуговування черги визначає порядок, у якому відбувається передача пакетів, що зберігаються в ній. Зміст застосування всіх алгоритмів зводиться до того, щоб забезпечити найкраще обслуговування трафіку з більш високим пріоритетом за умови, що й пакету з низьким пріоритетом гарантується відповідна увага.

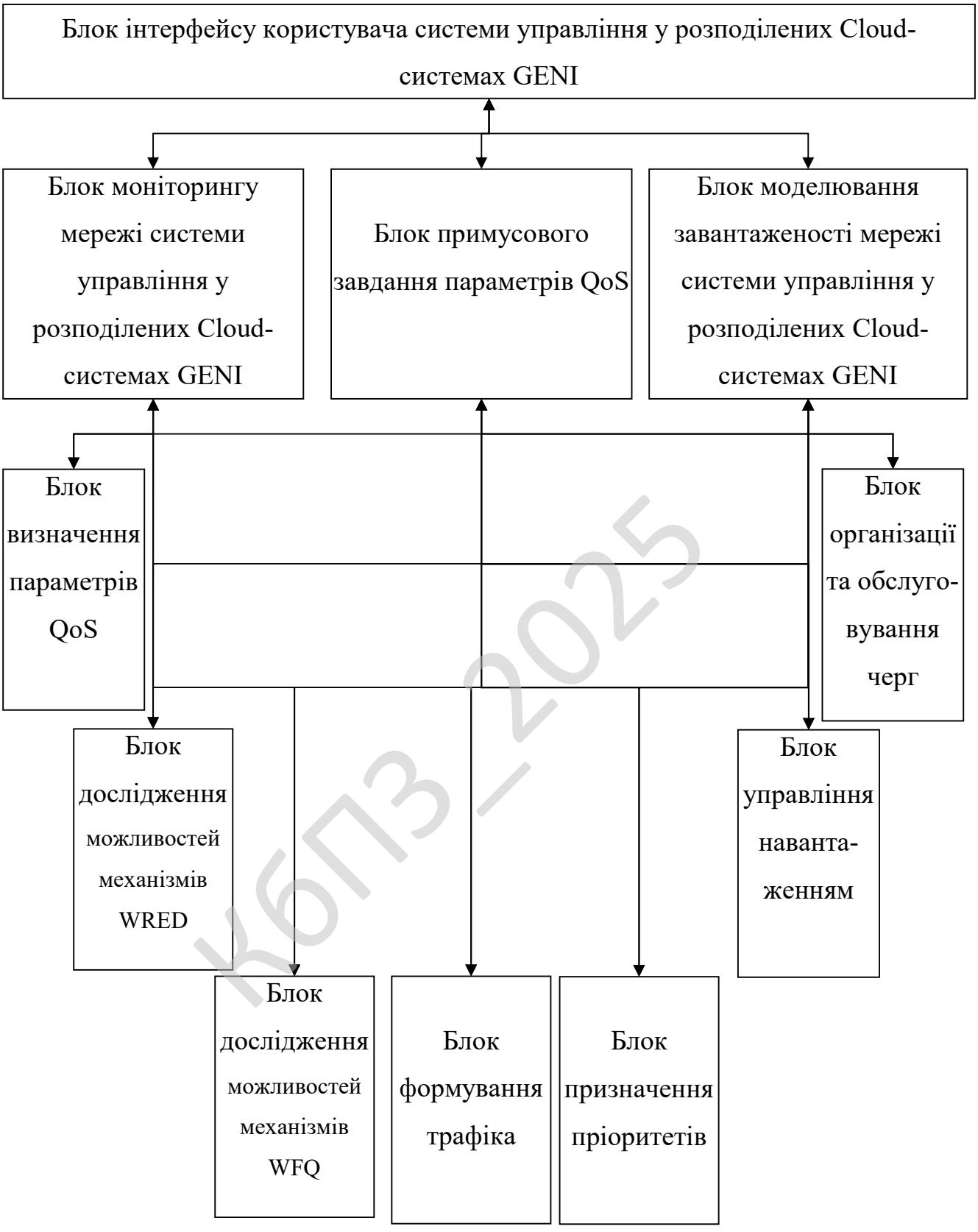

Рисунок 3.2 – Функціональна схема системи

При використанні способів завдання явних і неявних пріоритетів алгоритм обробки черг визначає порядок їхнього обслуговування. Відповідно до цього алгоритму на кожні два пакети, переданих у мережу із черги 1 (з високим пріоритетом) доводиться по одному пакету із черг 2 і 3. Пакети з однаковими пріоритетами передаються за принципом FIFO (”першим прийшов – першим вийшов”).

Якщо в мережі виникає перевантаження, служба черг не гарантує своєчасного досягнення пункту призначення найбільш важливими даними. Гарантується лише те, що ці пакети будуть передані раніше, ніж ті, що мають більш низький пріоритет.

Сучасні служби QoS вирішують таке завдання за рахунок резервування смуги пропущення. Кожній із черг (або їхніх груп) виділяється заздалегідь задана величина смуги пропущення, що гарантує певну смугу пропущення для черги з більш високим пріоритетом. Для критичних ситуацій, коли обсяг даних у черзі перевищує розміри смуги пропущення, в алгоритмах обслуговування звичайно передбачається передача трафіку з високим пріоритетом на смугу пропущення, “приналежну” чергам з низьким пріоритетом, і навпаки. Найпростіші алгоритми обслуговують кожну чергу за принципом FIFO. При цьому передача кадрів великого розміру, що мають високий пріоритет, може приводити до затримок трафіку іншого додатка з настільки ж високим пріоритетом, але меншим обсягом.

У більше складних алгоритмах уживає спроба “справедливої” обробки черг. Наприклад, алгоритм рівномірного пропорційного (або зваженого) обслуговування (WFQ – Weighted Fair Queuing), розроблений компанією Cisco, підрозділяє додатки на потребуючі великої й малої ширини смуги пропущення, а сама смуга пропущення розподіляється між всіма додатками нарівно. Слід зазначити, що основні виробники маршрутизаторів самі розробляють алгоритми обслуговування черг і використовують для їхнього опису власну термінологію.

Істотним недоліком сучасних маршрутизаторів і комутаторів є те, що вони підтримують мале число черг. Найчастіше виробники організують служби QoS,

що використовують чотири черги, хоча чим більше черг, тим більше різних пріоритетів можна привласнити переданим пакетам і тим “справедливіше” розподілити смугу пропущення між додатками. Наприклад, адміністратор у стані задати пріоритети таким чином, щоб перевага при передачі віддавалося пакетам, адресованим на більше віддалені вузли.

Блок управління навантаженням

Служба QoS дає можливість використовувати для керування мережею системи управління у розподілених Cloud-системах GENI два важливих механізми – керування в умовах перевантаження й запобігання перевантажень. Перший з них дозволяє кінцевій станції відразу знижувати швидкість передачі даних, коли в мережі починається втрата пакетів. У протоколах TCP/IP і SNA цей механізм підтримується вже протягом декількох років. І хоча сам по собі він не гарантує якості передачі, при його використанні разом з механізмом запобігання перевантажень результати виявляються набагато кращими. У мережах TCP/IP механізм запобігання перевантажень застосовується досить давно, але лише в останні роки він стає стандартом “де-факто” для маршрутизаторів телекомунікаційних мереж і Internet.

Стандартним способом запобігання перевантажень у мережі стало застосування механізму випадкового виділення пакетів (Random Early Detection, RED). При заповненні черг вище певної критичної оцінки цей механізм змушує маршрутизатор вибирати із черги за випадковим законом деякі пакети й “втрачати” їх. Швидкість передачі даних станціями-відправниками знижується, що й дозволяє уникнути переповнення черги.

Механізм пропорційного випадкового виділення пакетів – WRED (Weighted RED) – можна вважати наступною, більше зробленою “версією” RED. Він передбачає, що вибір пакетів, які повинні “втратитися”, буде відбуватися з обліком їх пріоритезації згідно IP TOS.

Блок формування трафіка

Формування трафіку – це загальний термін, яким прийнято позначати різні способи маніпулювання даними для підвищення якості їхньої передачі. Один

їхній таких способів – сегментація пакетів. У мережах системи управління у розподілених Cloud-системах GENI гарантовано високий рівень QoS досягається в тому числі й за рахунок малого розміру переданих пакетів (осередків – у термінології системи управління у розподілених Cloud-системах GENI). Максимальний час затримки при передачі будь-якого пакета мережі системи управління у розподілених Cloud-системах GENI – це час передачі одного осередку.

Запозичаючи корисні механізми технології системи управління у розподілених Cloud-системах GENI, виробники маршрутизаторів і комутаторів починають забезпечувати у своїх продуктах можливість сегментації пакетів. Деякі пристрої, призначені для мереж frame relay, сегментують пакети, передані по каналах глобальних мереж, щоб гарантувати конкретний час передачі й мінімізувати затримки.

Ще один спосіб формування трафіку – його "вирівнювання". Для таких протоколів, як наприклад, AppleTalk, характерна нерівномірна передача пакетів, що часом приводить до появи в мережі послідовностей або ланцюжків пакетів, а отже – до її перевантаження. Процедура вирівнювання трафіку дозволяє розчленувати ланцюжки шляхом розміщення пакетів у буфері перед їхньою передачею в мережу. Для забезпечення більше рівномірної передачі даних можна також вирівнювати трафік кінцевих вузлів мережі.

Блок визначення параметрів QoS

Призначений для визначення існуючих параметрів якості обслуговування (QoS). До них відносяться:

– Bandwidth (BW) – смуга пропущення, описує номінальну пропускну здатність середовища передачі інформації, визначає ширину каналу. Виміряється в bit/s (bps), kbit/s (kbps), mbit/s (mbps).

– Delay – затримка при передачі пакета.

– Jitter – коливання (варіація) затримки при передачі пакетів.

– Packet Loss – втрати пакетів. Визначає кількість пакетів, що відкидаються мережею під час передачі.

Блок примусового завдання параметрів QoS

Призначений для примусового завдання одного, або декількох параметрів якості обслуговування (QoS). До них відносяться:

– Bandwidth (BW) – смуга пропущення, описує номінальну пропускну здатність середовища передачі інформації, визначає ширину каналу. Виміряється в bit/s (bps), kbit/s (kbps), mbit/s (mbps).

– Delay – затримка при передачі пакета.

– Jitter – коливання (варіація) затримки при передачі пакетів.

– Packet Loss – втрати пакетів. Визначає кількість пакетів, що відкидаються мережею під час передачі.

Блок моделювання завантаженості мережі системи управління у розподілених Cloud-системах GENI

Призначений для моделювання завантаженості мережі системи управління у розподілених Cloud-системах GENI з визначеним трафіком та заданими параметрами якості обслуговування (QoS).

Блок моніторингу мережі системи управління у розподілених Cloud-системах GENI

Призначений для аналізу поточного стану мережі системи управління у розподілених Cloud-системах GENI.

Блок дослідження можливостей механізмів WRED

Одним з методів QoS, призначених для забезпечення необхідних вимог до різних потоків даних – запобігання перевантажень (congestion avoidance). Він заснований на обмеженні розмір черги, сигналізуючи джерелам даних про необхідність зменшити швидкість передачі інформації (WRED – Weighted random early detection).

Блок дослідження можливостей механізмів WFQ

Другим з методів QoS, призначених для забезпечення необхідних вимог до різних потоків даних – керування перевантаженням (congestion management). Він заснований на присвоєнні квот і пріоритетів потокам, і у випадку

перевантаження, потоки одержують якість, обмежену їхньою квотою й пріоритетом (WFQ – Weighted Fair Queuing).

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

Блок призначення пріоритетів

Нарівні з нарощуванням апаратного забезпечення мережі для реалізації QoS застосовуються й засобу типу завдання пріоритетів даних і організації черг. Маршрутизатори підтримують ці механізми протягом багатьох років, як і деякі з нових комутаторів для каналів Gigabit Ethernet. Однак ПЗ для керування мережею за заданими правилами, яких необхідно для практичного втілення цієї технології, поки не розроблено. Серед нових комутаторів такого класу можна назвати CoreBuilder 3500, CoreBuilder 9000 і SuperStack II компанії 3Com, пристрою серії Accelar фірми Bay Networks, SmartSwitch Router компанії Cabletron Systems, а також Catalyst 5000 і Catalyst 8000 виробництва Cisco.

Способи пріоритезації даних можна умовно підрозділити на явні й неявні.

При неявному призначенні пріоритетів маршрутизатор або комутатор автоматично привласнює послугам відповідні рівні, виходячи із заданих адміністратором мережі критеріїв (наприклад, типу додатка для застосовуваного протоколу передачі або адреси джерела). Кожний вхідний пакет аналізується (фільтрується) на відповідність цим критеріям. Механізм неявної пріоритезації підтримують практично всі маршрутизатори.

Деякі комутатори теж здатні задавати пріоритети, але мають обмежений набір функцій. Так, комутатори можуть забезпечувати пріоритезацію даних по типу віртуальної локальної мережі, адресі джерела або адресата, але не використовують інформацію більш високого рівня (протокол передачі або тип додатка). Розроблювальні в цей час системи керування мережею за заданими правилами дозволять реалізувати більше зроблені схеми пріоритезації даних при роботі з такими комутаторами.

При явній пріоритезації даних користувач або додаток запитує певний рівень служби, а комутатор або маршрутизатор намагається задовольнити запит. Імовірно, самим популярним механізмом явної пріоритезації стане протокол IP Precedence (протокол старшинства), що одержав другу назву IP TOS (IP Type Of Service), – один з розділів четвертої версії протоколу IP.

IP TOS резервує спеціальне поле в заголовку пакета, де можуть бути зазначені ознаки QoS, що визначають час затримки, швидкість пропущення й рівень надійності передачі пакета. Однак найдеться небагато популярних додатків – за винятком мультимедійного ПЗ, – у які реалізована підтримка протоколу IP TOS.

Зараз розробляється протокол резервування ресурсів RSVP, що передбачає більше складний, чим в IP TOS, механізм передачі від додатка до машрутизатору запиту на гарантовану якість послуг. Як і IP TOS, протокол RSVP поки не одержав широкої підтримки розроблювачів – він реалізований лише в окремих типах маршрутизаторів. Поширення RSVP стримується через те, що не вирішені деякі питання, пов'язані із сумісністю різних мереж. До того ж застосування RSVP значно збільшує навантаження на маршрутизатори й може привести до зниження швидкодії цих пристроїв.

Видимо, у доступному для огляду майбутньому неявна пріоритезація, не потребуюча серйозних обчислювальних потужностей маршрутизатора, залишиться більше популярною, чим явна. Крім того, при явному завданні пріоритетів значно ускладнюється керування мережею. Кінцеві користувачі, швидше за все, будуть набудовувати своє програмне забезпечення на запит найвищого з можливих рівнів послуг. Відповідно, адміністраторові мережі прийдеться розробляти правила керування користувачами й, можливо, навіть побудовувати служби з гарантованою якістю для кожного користувача окремо.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерської роботи, наведена на рисунку 3.3.

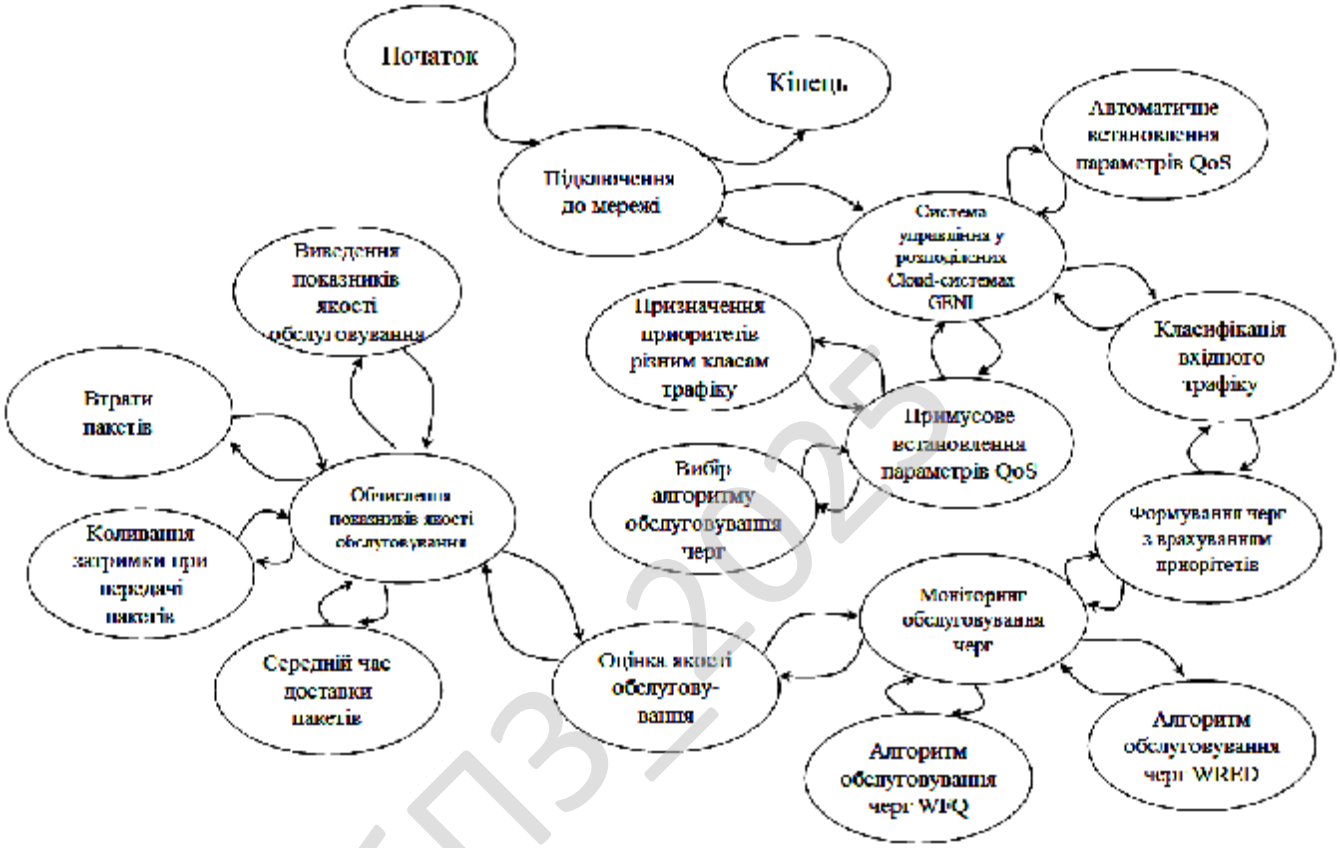

Рисунок 3.3 – Діаграма взаємодії процесів

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми. З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання. При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю системи управління у розподілених Cloud-системах GENI. Під час роботи над магістерською роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується. Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

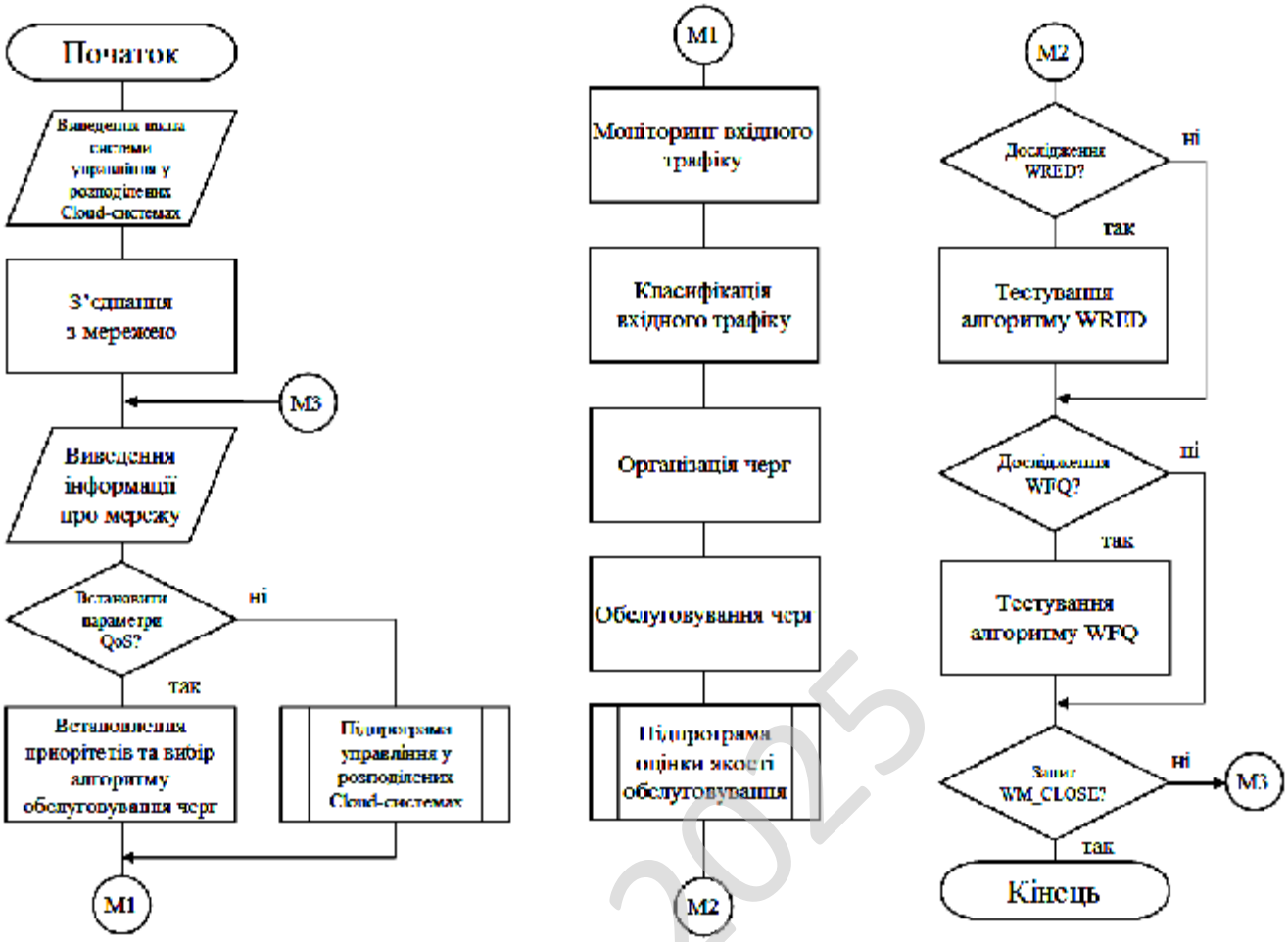

Рисунок 4.1 – Блок-схема основної програми

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

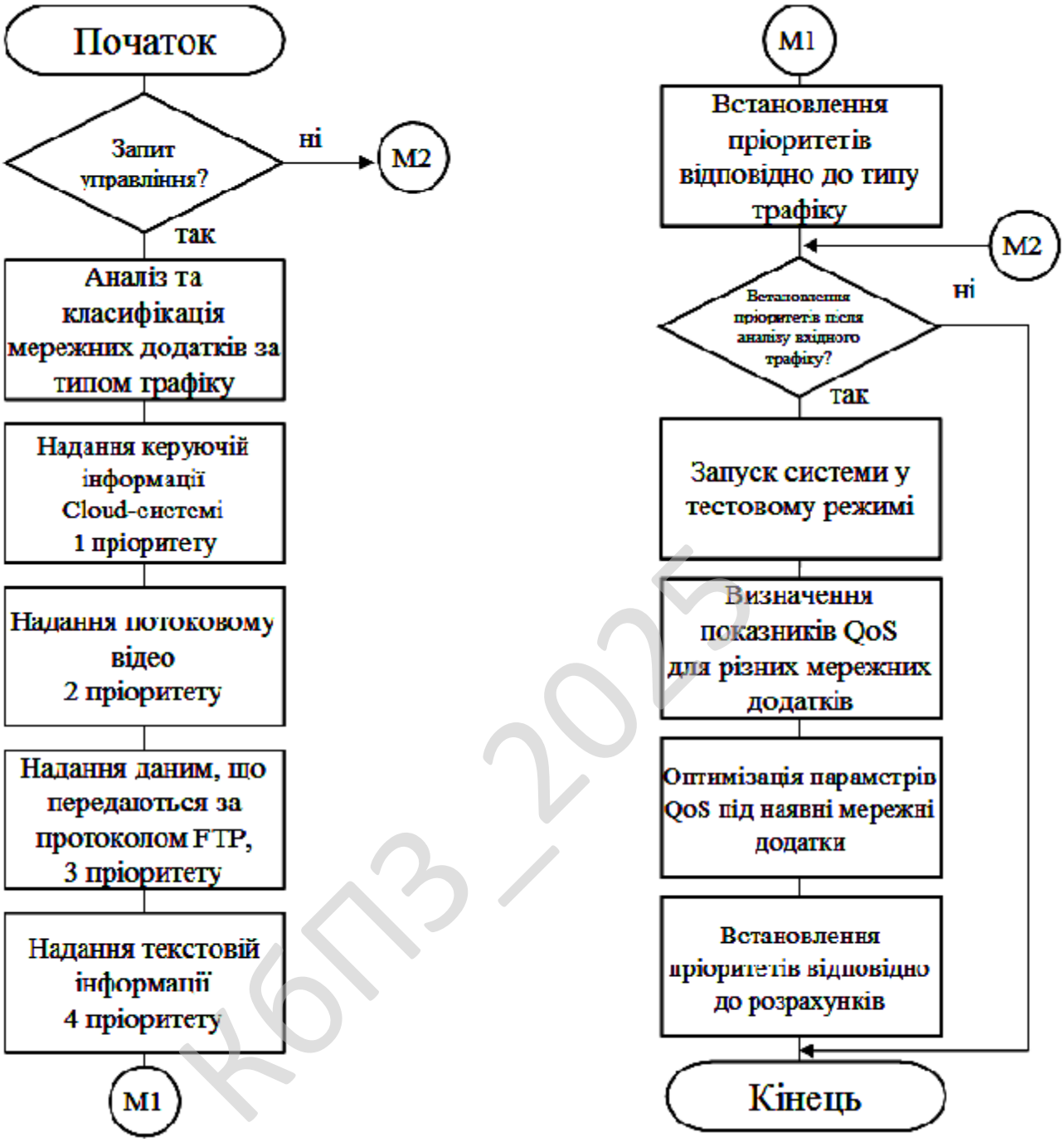

Рисунок 4.2 – Блок-схема роботи підпрограми

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Приклад вихідного коду та опис класів розробленої системи

Розроблювана система на мові Python призначається для дослідження та програмної реалізації процесів управління у розподілених Cloud системах GENI у

межах магістерської випускної кваліфікаційної роботи. Система моделює типове експериментальне середовище GENI, підтримує опис вузлів, ресурсів, каналів, формування зрізів та виконання сценаріїв керування ресурсами на основі політик і моніторингових даних. Такий підхід дає можливість відпрацьовувати алгоритми управління без прямого підключення до реального стенду GENI.

У якості базової моделі ресурсу використовується клас GeniResource. Цей клас описує абстрактний ресурс інфраструктури GENI з ідентифікатором, типом, словником ємностей та додатковими метаданими. Методи has_capacity, allocate та release реалізують перевірку доступності ресурсу, резервування частини ємності та повернення зарезервованих величин. Таким чином система уніфіковано представляє як процесорні, так і інші ресурси інфраструктури.

Клас GeniNode описує вузол у розподіленій Cloud системі GENI. Вузол містить ідентифікатор, розташування, перелік ресурсів, мітки, конфігурацію та поточний стан. Метод find_resource знаходить ресурс потрібного типу на вузлі, а метод is_available повертає доступність вузла для розміщення експериментального навантаження. Наявність міток у полі tags дає можливість розрізняти ролі вузлів наприклад ядро чи крайова частина топології.

Клас GeniLink представляє логічний канал зв’язку між двома вузлами. Об’єкт містить пропускну здатність у мегабітах за секунду, затримку у мілісекундах та довільний словник метаданих. Це дозволяє моделювати обмеження мережевої інфраструктури під час дослідження алгоритмів управління трафіком у середовищі GENI.

Експериментальний зріз моделюється класом GeniSlice. Об’єкт зрізу містить ідентифікатор, назву, опис, перелік вузлів, перелік каналів, поточний стан та словник параметрів. Методи add_node і add_link додають відповідно вузли та канали до зрізу. Поле state відображає етап життєвого циклу зрізу визначення, резервування, запуск, зупинка, звільнення. Поле parameters містить налаштування сценарію наприклад патерн трафіку чи пріоритет.

Для зберігання опису інфраструктури використовується клас

TopologyRepository. Він містить словники вузлів та каналів і забезпечує операції реєстрації, отримання за ідентифікатором та отримання списків. Метод find_nodes_by_tag повертає вузли з заданою міткою що зручно при автоматичному підборі ресурсів під конкретний сценарій. Аналогічно клас SliceRepository відповідає за зберігання експериментальних зрізів і надає методи реєстрації, пошуку та отримання всього переліку.

Підсистема моніторингу складається з класів MonitoringRecord, MonitoringRepository та MonitoringService. Об’єкт MonitoringRecord описує один вимір для вузла у певний момент часу і містить завантаження процесора, використання пам’яті, вхідний та вихідний трафік і словник додаткових метрик. Клас MonitoringRepository зберігає усі записи моніторингу, дозволяє додавати нові значення, отримувати історію та останній запис для конкретного вузла. Клас MonitoringService реалізує періодичний збір метрик. Він проходить по всіх доступних вузлах, генерує псевдовипадкові значення завантаження і записує їх у репозиторій. Запуск і зупинка служби моніторингу здійснюється окремими методами, цикл збору виконується у фоні через допоміжний потік виконання.

Взаємодія з зовнішньою інфраструктурою GENI абстрагується класом GeniApiClient. У поточній реалізації він працює як псевдо клієнт та оперує виключно локальними моделями. Метод discover_resources повертає перелік доступних вузлів, метод reserve_slice встановлює стан зрізу як зарезервований, методи start_slice, stop_slice та release_slice змінюють стан відповідно до етапу життєвого циклу. Водночас усі дії журналюються через стандартний модуль logging. У реальному розгортанні логіка клієнта може замінюватися викликом API GENI з використанням тих самих інтерфейсів.

Підсистема керування реалізується через класи PolicyContext, PolicyRule та PolicyEngine. Об’єкт PolicyContext інкапсулює посилання на репозиторії топології, зрізів, моніторингу та клієнт GENI і передається у логіку правил. Клас PolicyRule описує одне правило керування, що містить назву, ім’я метрики, порогове значення, тип порівняння і дію яка виконується у разі спрацювання.

Метод is_triggered перевіряє чи перевищує значення метрики заданий поріг. Клас PolicyEngine містить перелік правил, оцінює моніторингові дані для окремого вузла чи для всієї інфраструктури і викликає відповідні дії. Під час оцінки система обчислює середнє значення завантаження процесора та пам’яті за усі записи для вузла і порівнює їх з порогами правил.

Функція create_default_policy_engine створює типовий набір правил для експериментів. У ній визначається дія scale_out_action яка додає новий вузол до топології при перевищенні середнього завантаження процесора певного вузла. Ця дія створює новий об’єкт GeniResource з меншим числом ядер, формує вузол GeniNode з міткою автоматичного масштабування і реєструє його у TopologyRepository. На основі цієї дії формується правило PolicyRule з назвою cpu_scale_out яке спрацьовує при перевищенні порога завантаження процесора. Таким чином у системі реалізується простий механізм масштабування у бік збільшення ресурсу.

Створення і виконання експериментального сценарію описується класом ExperimentScenario. Об’єкт містить ідентифікатор сценарію, назву, опис, кількість необхідних вузлів, словник вимог до ресурсів та параметри зрізу. Метод build_slice проходить по доступних вузлах, відбирає ті що мають ресурс типу cpu з достатньою ємністю, викликає метод allocate для резервування і формує зріз GeniSlice з переліком обраних вузлів та параметрами сценарію. Таким чином логіка підбору ресурсів реалізується окремо від механізму керування.

Центральним компонентом є клас ExperimentOrchestrator. Він отримує на вході репозиторії, клієнт GENI, двигун політик та службу моніторингу і координує повний життєвий цикл експерименту. Метод prepare_topology формує прикладну топологію якщо вона ще відсутня, створює декілька вузлів з ресурсами типу cpu та один канал між ними. Метод run_scenario будує зріз на основі сценарію, реєструє його у репозиторії, резервує ресурси через GeniApiClient, запускає службу моніторингу, переводить зріз у стан виконання і у циклі оцінює політики, записуючи повідомлення у журнал. Після завершення

інтервалу експерименту оркестратор зупиняє зріз, зупиняє моніторинг та звільняє ресурси.

Функція build_demo_system будує повністю налаштовану систему для демонстрації. У ній створюються репозиторії, клієнт GENI, служба моніторингу, двигун політик та оркестратор. Функція main формує сценарій ExperimentScenario для керування зрізом у GENI з двома вузлами, вимогами до процесорних ядер та параметрами трафіку, після чого передає його оркестратору і запускає експеримент. По завершенню роботи система виводить у журнал стан зрізу що дає можливість фіксувати результати дослідження.

У цілому розроблена система на Python реалізує повний цикл управління у моделі розподіленої Cloud системи GENI включаючи опис ресурсів, побудову зрізів, моніторинг, виконання політик та автоматичне масштабування. Архітектура з чітким розділенням моделей, репозиторіїв, служби моніторингу, клієнта GENI, двигуна політик та оркестратора підвищує керованість і розширюваність рішення і дає можливість розширювати систему під конкретні дослідницькі задачі магістерської роботи.

```
import logging
import random
import threading
import time
from dataclasses import dataclass, field
from datetime import datetime
from statistics import mean
from typing import Dict, List, Optional, Callable, Any


# Налаштування базового логування
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s]
%(message)s")


# Модель ресурсу GENI
@dataclass
class GeniResource:
    """Описує абстрактний ресурс у інфраструктурі GENI"""

    resource_id: str
    kind: str
    capacity: Dict[str, float] = field(default_factory=dict)
    metadata: Dict[str, Any] = field(default_factory=dict)

    def has_capacity(self, requirements: Dict[str, float]) -> bool:
        """Перевіряє чи ресурс задовольняє задані вимоги"""
        for key, value in requirements.items():
```

```
            if self.capacity.get(key, 0.0) < value:
                return False
        return True

    def allocate(self, requirements: Dict[str, float]) -> bool:
        """Резервує частину ємності ресурсу якщо це можливо"""
        if not self.has_capacity(requirements):
            return False
        for key, value in requirements.items():
            self.capacity[key] = self.capacity.get(key, 0.0) - value
        return True

    def release(self, requirements: Dict[str, float]) -> None:
        """Повертає раніше зарезервовану ємність ресурсу"""
        for key, value in requirements.items():
            self.capacity[key] = self.capacity.get(key, 0.0) + value


# Модель вузла GENI
@dataclass
class GeniNode:
    """Описує вузол у розподіленій Cloud системі GENI"""

    node_id: str
    location: str
    resources: List[GeniResource] = field(default_factory=list)
    tags: List[str] = field(default_factory=list)
    status: str = "online"
    configuration: Dict[str, Any] = field(default_factory=dict)

    def find_resource(self, kind: str) -> Optional[GeniResource]:
        """Повертає ресурс потрібного типу якщо він є на вузлі"""
        for res in self.resources:
            if res.kind == kind:
                return res
        return None

    def is_available(self) -> bool:
        """Повертає поточний стан доступності вузла"""
        return self.status == "online"


# Модель каналу зв'язку GENI
@dataclass
class GeniLink:
    """Описує логічне з'єднання між двома вузлами"""

    link_id: str
    node_a: str
    node_b: str
    bandwidth_mbps: float
    latency_ms: float
    metadata: Dict[str, Any] = field(default_factory=dict)
# Модель зрізу GENI
@dataclass
class GeniSlice:
    """Описує експериментальний зріз у GENI"""

    slice_id: str
    name: str
    description: str
    node_ids: List[str] = field(default_factory=list)
    link_ids: List[str] = field(default_factory=list)
```

```
    state: str = "defined"
    parameters: Dict[str, Any] = field(default_factory=dict)

    def add_node(self, node_id: str) -> None:
        """Додає вузол до зрізу"""
        if node_id not in self.node_ids:
            self.node_ids.append(node_id)

    def add_link(self, link_id: str) -> None:
        """Додає канал до зрізу"""
        if link_id not in self.link_ids:
            self.link_ids.append(link_id)


# Репозиторій топології GENI
class TopologyRepository:
    """Зберігає вузли та канали експериментальної топології"""

    def __init__(self) -> None:
        self.nodes: Dict[str, GeniNode] = {}
        self.links: Dict[str, GeniLink] = {}

    def add_node(self, node: GeniNode) -> None:
        """Реєструє вузол у репозиторії"""
        self.nodes[node.node_id] = node

    def add_link(self, link: GeniLink) -> None:
        """Реєструє канал у репозиторії"""
        self.links[link.link_id] = link

    def get_node(self, node_id: str) -> Optional[GeniNode]:
        """Повертає вузол за ідентифікатором"""
        return self.nodes.get(node_id)

    def get_link(self, link_id: str) -> Optional[GeniLink]:
        """Повертає канал за ідентифікатором"""
        return self.links.get(link_id)

    def list_nodes(self) -> List[GeniNode]:
        """Повертає перелік усіх вузлів"""
        return list(self.nodes.values())

    def list_links(self) -> List[GeniLink]:
        """Повертає перелік усіх каналів"""
        return list(self.links.values())

    def find_nodes_by_tag(self, tag: str) -> List[GeniNode]:
        """Повертає вузли що містять задану мітку"""
        return [n for n in self.nodes.values() if tag in n.tags]

# Репозиторій зрізів GENI
class SliceRepository:
    """Зберігає експериментальні зрізи"""

    def __init__(self) -> None:
        self.slices: Dict[str, GeniSlice] = {}

    def add_slice(self, slice_obj: GeniSlice) -> None:
        """Реєструє зріз"""
        self.slices[slice_obj.slice_id] = slice_obj

    def get_slice(self, slice_id: str) -> Optional[GeniSlice]:
        """Повертає зріз за ідентифікатором"""
```

```
        return self.slices.get(slice_id)

    def list_slices(self) -> List[GeniSlice]:
        """Повертає перелік усіх зрізів"""
        return list(self.slices.values())

# Модель запису моніторингу
@dataclass
class MonitoringRecord:
    """Описує набір метрик для одного вузла"""

    timestamp: datetime
    node_id: str
    cpu_load: float
    memory_usage: float
    net_in_mbps: float
    net_out_mbps: float
    custom_metrics: Dict[str, float] = field(default_factory=dict)


# Репозиторій моніторингових даних
class MonitoringRepository:
    """Зберігає історію моніторингу"""

    def __init__(self) -> None:
        self.records: List[MonitoringRecord] = []

    def add_record(self, record: MonitoringRecord) -> None:
        """Додає новий запис моніторингу"""
        self.records.append(record)

    def get_records_for_node(self, node_id: str) -> List[MonitoringRecord]:
        """Повертає усі записи для заданого вузла"""
        return [r for r in self.records if r.node_id == node_id]

    def get_last_record_for_node(self, node_id: str) ->
Optional[MonitoringRecord]:
        """Повертає останній запис для вузла"""
        node_records = self.get_records_for_node(node_id)
        if not node_records:
            return None
        return node_records[-1]

# Служба моніторингу
class MonitoringService:
    """Виконує періодичний збір метрик з вузлів"""

    def __init__(
        self,
        topology_repo: TopologyRepository,
        monitoring_repo: MonitoringRepository,
        sample_interval: float = 5.0,
    ) -> None:
        self.topology_repo = topology_repo
        self.monitoring_repo = monitoring_repo
        self.sample_interval = sample_interval
        self._running = False
        self._thread: Optional[threading.Thread] = None

    def _sample_once(self) -> None:
        """Виконує один цикл збору метрик"""
        now = datetime.utcnow()
        for node in self.topology_repo.list_nodes():
```

```
            if not node.is_available():
                continue
            cpu = random.uniform(0.0, 100.0)
            mem = random.uniform(0.0, 100.0)
            net_in = random.uniform(0.0, 100.0)
            net_out = random.uniform(0.0, 100.0)
            record = MonitoringRecord(
                timestamp=now,
                node_id=node.node_id,
                cpu_load=cpu,
                memory_usage=mem,
                net_in_mbps=net_in,
                net_out_mbps=net_out,
            )
            self.monitoring_repo.add_record(record)

    def _run_loop(self) -> None:
        """Внутрішній цикл збору метрик"""
        while self._running:
            self._sample_once()
            time.sleep(self.sample_interval)

    def start(self) -> None:
        """Запускає фон збір моніторингу"""
        if self._running:
            return
        self._running = True
        self._thread = threading.Thread(target=self._run_loop, daemon=True)
        self._thread.start()

    def stop(self) -> None:
        """Зупиняє службу моніторингу"""
        self._running = False


# Клієнт для взаємодії з GENI
class GeniApiClient:
    """Інкапсулює виклики до зовнішнього інтерфейсу GENI"""

    def __init__(self, topology_repo: TopologyRepository, slice_repo:
SliceRepository) -> None:
        self.topology_repo = topology_repo
        self.slice_repo = slice_repo

    def discover_resources(self) -> List[GeniNode]:
        """Повертає перелік доступних вузлів"""
        logging.info("Виконується псевдо запит discover_resources")
        return self.topology_repo.list_nodes()

    def reserve_slice(self, slice_obj: GeniSlice) -> None:
        """Резервує ресурси для зрізу"""
        logging.info("Резервування зрізу %s", slice_obj.slice_id)
        slice_obj.state = "reserved"

    def start_slice(self, slice_obj: GeniSlice) -> None:
        """Запускає зріз"""
        logging.info("Запуск зрізу %s", slice_obj.slice_id)
        slice_obj.state = "running"

    def stop_slice(self, slice_obj: GeniSlice) -> None:
        """Зупиняє зріз"""
        logging.info("Зупинка зрізу %s", slice_obj.slice_id)
        slice_obj.state = "stopped"
```

```
    def release_slice(self, slice_obj: GeniSlice) -> None:
        """Вивільняє ресурси зрізу"""
        logging.info("Звільнення зрізу %s", slice_obj.slice_id)
        slice_obj.state = "released"

# Контекст для виконання політик
@dataclass
class PolicyContext:
    """Передає доступ до репозиторіїв у логіку правил"""

    topology_repo: TopologyRepository
    slice_repo: SliceRepository
    monitoring_repo: MonitoringRepository
    api_client: GeniApiClient
```

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 7624:2014 («Калина»). При розробці національного стандарту ДСТУ 7624:2014 (блоковий шифр «Калина» і режими його роботи) було ухвалене рішення забезпечити прозорість проектування й використовувати консервативний підхід із застосуванням добре досліджених конструкцій, що забезпечують запас стійкості для безпечного застосування алгоритму в умовах істотного прогресу криптоаналітичних технік і засобів обробки даних.

Національний стандарт підтримує розмір блоку й довжину ключа шифрування 128, 256 і 512 біт (довжина ключа дорівнює розміру блоку або у два рази перевищує його), забезпечуючи нормальний, високий і надвисокий рівень стійкості (зараз це єдиний у світі стандарт блокового шифрування, що підтримує 512-бітові симетричні ключі). Різні варіанти забезпечують гнучкість вибору параметрів для розроблювачів систем криптографічного захисту, що дозволяє одержати як найвищий рівень швидкодії, так і найбільший запас стійкості перетворення.

Високорівнева конструкція використовує добре досліджену Square-подібну SPN-структуру, застосовувану в алгоритмах AES/Rijndael, Whirlpool, «Стрибог», «Коник» і багатьох інших. Циклове перетворення побудоване на базі таблиць підстановки (S-блоків) і множення на МДР-матрицю над кінцевим

полем, забезпечуючи необхідні криптографічні властивості. Застосування саме такої конструкції дозволяє забезпечити доказову стійкість до диференціального, лінійному й ін. видам криптоаналізу, одночасно забезпечуючи ефективну реалізацію на широкому спектрі програмних і програмно-апаратних платформ. При виборі розміру МДР-матриці був прийнятий в увагу розмір кешу L1 сучасних і перспективних процесорів, що дозволило оптимізувати швидкодія програмної реалізації шифру.

Альтернативний варіант, який розглядався при розробці циклової функції, – ARX перетворення (Addition-rotation-xor). Цей підхід реалізований у шифрі SPECK (розроблений Агентством національної безпеки США й переданий у відкритий доступ), у блокових алгоритмах, на основі яких побудовано сімейство геш-функцій SHA-0,1,2 і ін. Перевагою походу є компактність і швидкодія перетворення. Але, у той же час, є й істотний недолік, пов'язаний з відсутністю методів, що дозволяють виконати строге аналітичне обґрунтування криптографічної стійкості таких розв'язків. Навіть із наймогутнішими у світі можливостями для аналізу, США кілька раз були змушено модифікувати свої стандарти гешування через знайдені уразливості: з 1993 по 1995 рр. діяв SHA-0, з 1995 по 2001 рр. застосовувався SHA-1, з тих пор використовується SHA-2. Через питання, що піднімаються, до стійкості й цієї версії, у США з 2008 по 2012 рр. був проведений міжнародний конкурс SHA-3. До теперішнього часу розроблений проект стандарту FIPS 202, що описує нову криптографічну геш-функцію.

Таким чином, консервативний і прозорий підхід до проектування нового національного стандарту України обумовив вибір добре перевіреної конструкції на базі S-блоків і лінійного перетворення, для якої можливо забезпечити доказову стійкість до різних видів криптоаналізу.

При порівнянні характеристик S-блоків і інших симетричних перетворень можна відзначити, що саме національний стандарт України забезпечує найбільшу нелінійність булевих функцій, що дає додатковий запас стійкості до лінійного криптоаналізу. Ще більше значення нелінійності для взаємо- однозначної

підстановки можна забезпечити, застосовуючи, наприклад, аффінно-тквівалентні статечні функції в кінцевому полі, але такі перетворення, використані в AES, Camellia і ін. алгоритмах, ставлять шифр під погрозу реалізації алгебраїчної атаки (цей метод криптоаналізу був успішно застосований проти шифру Keeloq, використовуваного в системах автомобільної безпеки).

У якості схеми розгортання ключів була запропонована нова конструкція з наступними властивостями:

– забезпечення криптографічної стійкості до відомих методів аналізу, відсутність «слабких» ключів, які можуть погіршити властивості перетворення;

– зручність програмної й програмно-апаратної реалізації (для формування циклових ключів застосовуються тільки операції, використовувані при шифруванні);

– висока обчислювальна складність відновлення ключа шифрування по одному або декільком цикловим ключам.

Остання властивість забезпечує додатковий захист до атак на реалізацію, коли зловмисник намагається атакувати інженерні розв'язки (вимірюючи споживаний пристроєм струм, навмисне викликаючи збої в роботі через навмисний перегрів шифратора та ін.). Ця особливість є істотною перевагою для ряду додатків, зокрема, при реалізації шифрування на смарт-картах, USB-токенах та ін., коли ключ прошитий у пристрої й повинен бути захищений від зовнішнього доступу (наприклад, у модулях доступу до платних цифрових ТБ-каналівм і ін.)

Кількість циклів шифрування залежить від довжини ключа: 10 циклів для 128-бітового, 14 циклів для 256-бітового й 18 циклів для 512-бітового ключа шифрування.

У порівнянні з іншими алгоритмами на основі Square-подібної SPN-структури, блоковий шифр «Калина» має наступні істотні конструктивні відмінності:

– початкове й кінцеве забілювання з використанням модульного додавання (264) для підвищення складності криптоаналітичних атак;

– застосування чотирьох різні S-блоків (замість одного) із властивостями для захисту від алгебраїчних атак, і при порівнянні характеристик з іншими блоковими й потоковими шифрами забезпечують найбільшу нелінійність булевих функцій (104), що дає додатковий запас стійкості перетворення;

– збільшений розмір МДР-перетворення, що поліпшує криптографічні властивості й дозволяє оптимізувати швидкодія на сучасних 64-бітових платформах;

– нову односпрямовану схему формування циклових ключів, що забезпечує захист від атак, ефективність програмної й програмно-апаратної реалізації, разом з додатковою стійкістю до методів аналізу спеціального виду.

Оцінка криптографічної стійкості до диференціального, лінійному, алгебраїчному, інтегральному й іншим методам аналізу (практичний критерій) показала, що шифр є стійким при 6 циклах для 128-бітового блоку, 7 циклах для 256-бітового й 9 циклах для 512-бітового (кожний додатковий цикл забезпечує експонентний ріст складності криптоаналізу). Таким чином, шифр, що містить 10, 14 і 18 циклів для блоку 128, 256 і 512 біт відповідно, забезпечує захист від розглянутих видів аналізу й має істотний запас стійкості.

Крім блокового шифру, ДСТУ 7624:2014 визначає режими роботи, що відповідають як ISO 10116:2006, так і додаткові, призначені для сучасних систем криптографічного захисту IP-трафіка, прозорого шифрування носіїв інформації й ін. У стандарті визначені обсяги повідомлень, після обробки яких потрібна обов'язкова зміна ключа. Крім того, приводяться рекомендації розроблювачам, що обертають увагу на необхідність запобігання атак з використанням особливостей реалізації засобів шифрування.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання магістерської роботи.

Розроблене програмне забезпечення системи управління у розподілених Cloud-системах GENI складається з наступних функціональних блоків:

– Навігаційне меню: Файл; Моніторинг; Тестування; Параметри; Довідка.

– Підрозділ представлення параметрів вибору та обчислення.

– Вікна виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

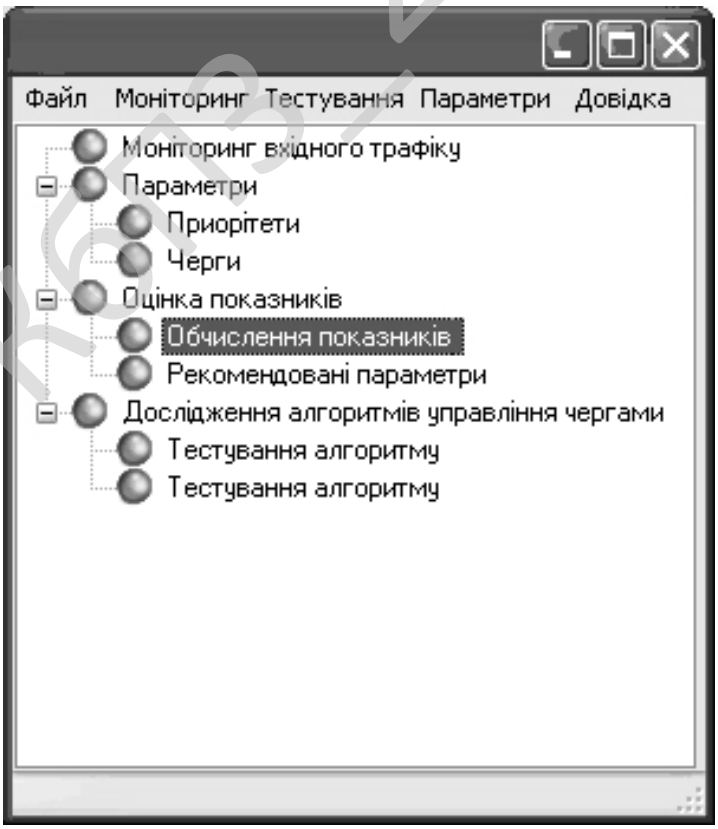

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

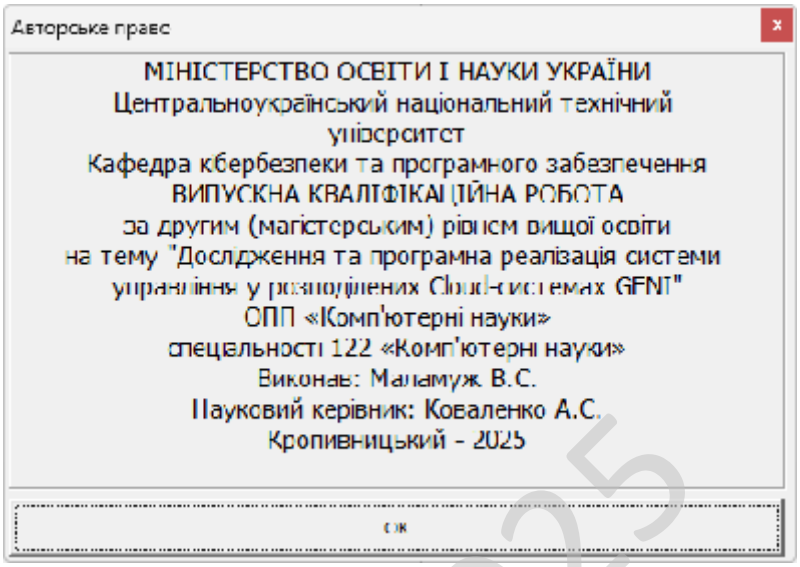

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи управління у розподілених Cloud-системах GENI.

Метою розробки є дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI.

Об’єктом дослідження є процес управління у розподілених Cloud-системах GENI.

Предметом дослідження є методи управління у розподілених Cloud-системах GENI.

Методи дослідження базуються на методах хмарних обчислень, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод управління у розподілених Cloud-системах GENI.

– Розроблено вітчизняний продукт управління у розподілених Cloud-системах GENI, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати цього дослідження можуть бути особливо цікавими для великих корпорацій, що працюють із географічно розподіленими дата-центрами або користуються мультихмарними рішеннями. Такі компанії стикаються з проблемами синхронізації, надлишкових витрат і складністю моніторингу ефективності використання ресурсів. Впровадження системи GENI дозволяє централізовано контролювати всі елементи хмарної інфраструктури, знижуючи технічні ризики та фінансові втрати.

Окрім корпоративного сектору, ця система становить інтерес для державних установ, що мають критичні ІТ-сервіси. Для таких організацій важливо забезпечити безперервність роботи, надійність доступу до даних і швидке реагування у разі збоїв. GENI здатна інтегрувати різні платформи, що робить її корисною для національних інформаційних систем.

Також результат дослідження може зацікавити хмарних провайдерів і науково-дослідні центри, які займаються питаннями оптимізації продуктивності. GENI пропонує інноваційний підхід до розподіленого управління, який може бути адаптований у рамках освітніх і дослідницьких лабораторій, що займаються розробкою моделей майбутніх мережевих архітектур.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості впровадження системи GENI можна використати метод експертних оцінок, заснований на думці фахівців у сфері

хмарних обчислень, кібербезпеки та управління інфраструктурою. До групи експертів можуть увійти IT-директори, аналітики, адміністратори дата-центрів і представники компаній-провайдерів Cloud-сервісів. Вони оцінюють систему за критеріями: технічна інноваційність, економічна вигода, масштабованість, складність впровадження та відповідність ринковим потребам.

Після обробки даних, отриманих від десяти експертів, середня оцінка може становити 8,7 бала з 10, що свідчить про високу привабливість рішення. Найвищі бали зазвичай отримують критерії масштабованості й економічної ефективності, адже GENI дозволяє суттєво знизити витрати на обслуговування розподілених систем. Водночас деякі експерти можуть відзначити відносну складність впровадження, що потребує кваліфікованих кадрів.

Результати експертного аналізу демонструють, що ринок готовий до таких рішень, оскільки зростає кількість організацій, що переходять до мультихмарних архітектур. Отже, проєкт має високу потенційну привабливість як для внутрішніх IT-підрозділів, так і для компаній, які спеціалізуються на створенні інфраструктурних сервісів.

7.3 Вибір методу оцінки вартості ПЗ

Найдоцільніше використовувати комбінацію витратного методу та методу порівняльного аналізу (benchmarking). Витратний метод дозволяє точно обчислити всі необхідні інвестиції – від закупівлі серверного обладнання до розробки ПЗ, тестування, ліцензування й навчання персоналу. Він допомагає зрозуміти реальні фінансові потреби на етапі запуску.

Порівняльний аналіз, у свою чергу, дозволяє співставити вартість GENI-рішення з подібними системами управління хмарними середовищами, наприклад VMware vRealize або OpenStack Heat. Такий підхід дає змогу визначити конкурентні переваги проєкту за ціною впровадження та експлуатаційними витратами.

Крім того, до розрахунку варто включити непрямі вигоди: скорочення простоїв, економію енергоспоживання, зменшення кількості необхідного персоналу. Завдяки цим параметрам метод стає більш наближеним до реальних умов і дозволяє обґрунтовано оцінити комерційну ефективність впровадження GENI у великих компаніях.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Організація використовує кілька незалежних хмарних середовищ (AWS, Azure, Google Cloud і приватні дата-центри). Через відсутність єдиного управлінського контуру спостерігається низька узгодженість ресурсів, дублювання обчислювальних процесів і надмірне споживання енергії.

Впровадження системи управління на базі GENI-архітектури дозволяє створити єдиний координаційний рівень, який здійснює автоматичне розподілення навантаження, моніторинг продуктивності й динамічну оптимізацію використання ресурсів у розподілених Cloud-системах. Це забезпечує зменшення витрат на інфраструктуру, підвищення швидкодії сервісів і стабільність доступу до даних. Вхідні дані зафіксовано в таблиці 7.1.

Розрахунок економічного ефекту демонструє наступне: поточні щорічні витрати без GENI – 31 500 000 грн/рік, нові витрати після впровадження GENI – 19 200 000 грн/рік, щорічна економія – 12 300 000 грн/рік, чистий економічний ефект – 10 300 000 грн у перший рік, термін окупності ≈ 0,16 року (2 місяці), рентабельність інвестицій – 615 %.

Додаткові нефінансові вигоди: підвищення ефективності IT-команд – автоматизація моніторингу та управління звільняє адміністраторів від рутинних задач; оптимізація енергоспоживання – завдяки розподілу навантаження скорочується кількість серверів, що працюють на низьких навантаженнях; підвищення відмово стійкості – GENI забезпечує резервування вузлів і швидке

свідновлення у разі збоїв; масштабованість і адаптивність – система легко інтегрується з новими провайдерами Cloud-послуг без зміни архітектури; прозора аналітика – єдиний інтерфейс управління забезпечує моніторинг витрат, продуктивності й SLA у режимі реального часу.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	**До впровадження**	**Після впровадження**	**Економічний ефект**
Кількість хмарних вузлів	250	250	—
Середнє завантаження ресурсів	55%	85%	+30% ефективності
Місячні витрати на оренду Cloud-ресурсів	1 800 000 грн	1 250 000 грн	–550 000 грн
Витрати на адміністрування систем	450 000 грн	250 000 грн	–200 000 грн
Втрати через простої систем (Downtime)	300 000 грн	100 000 грн	–200 000 грн
Початкові інвестиції у впровадження GENI-рішення (розробка, інтеграція, навчання персоналу)	—	—	2 000 000 грн

Отже, система демонструє високий рівень економічної ефективності (ROI ≈ 615 %), значне підвищення продуктивності хмарних систем і помітне скорочення витрат на підтримку IT-інфраструктури.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Просування проєкту варто розпочати зі створення демонстраційного стенду, який покаже потенційним клієнтам, як GENI працює в реальних умовах. Такий пілот може бути запущений на обмеженій кількості серверів і використовуватись для порівняння з існуючими інструментами управління. Це допоможе на практиці показати зменшення витрат і підвищення ефективності.

Наступним етапом має стати співпраця з провідними Cloud-провайдерами або інтеграторами, які зможуть пропонувати систему GENI своїм клієнтам як додатковий рівень оптимізації. Важливо розробити маркетингову стратегію, орієнтовану на економічні аргументи – скорочення витрат і підвищення відмовостійкості.

Після пілотних впроваджень варто представити успішні кейси на спеціалізованих IT-форумах, конференціях або у профільних виданнях. Наявність підтверджених результатів і позитивних відгуків стане вирішальним фактором у подальшому масштабуванні GENI на ринок корпоративних рішень.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Оптимізація каналів збуту може базуватися на партнерській моделі. Доцільно укласти угоди з компаніями-інтеграторами, які вже займаються побудовою хмарних рішень. Вони можуть виступати посередниками між розробником і кінцевим клієнтом, отримуючи частину прибутку за впровадження. Це дозволить розширити охоплення ринку без суттєвого збільшення маркетингового бюджету.

Іншим напрямом може бути впровадження ліцензійної моделі SaaS, яка передбачає оплату за підпискою. Це зробить систему доступною навіть для середнього бізнесу, який прагне контролювати витрати на Cloud-ресурси, але не має великих коштів для купівлі ліцензії. Такий підхід також створить стабільний грошовий потік для розробника.

Додатково можна створити навчальну програму та сертифікацію для адміністраторів, які працюватимуть із системою GENI. Це допоможе підвищити рівень лояльності клієнтів і забезпечить стабільну підтримку проєкту в довгостроковій перспективі.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовими факторами успіху є технологічна стабільність, масштабованість і довіра користувачів. GENI має забезпечити безперебійне керування розподіленими середовищами, при цьому бути достатньо гнучкою, щоб адаптуватися під різні архітектури – від приватних хмар до гібридних інфраструктур.

Важливим чинником є також простота інтеграції. Якщо система може без проблем впроваджуватися у вже існуючу IT-екосистему підприємства, вона має набагато більше шансів на успіх. Зменшення часу розгортання та мінімізація ручного налаштування сприяють зростанню задоволеності клієнтів.

Не менш суттєвою є якість сервісної підтримки та постійні оновлення. Користувачі цінують не лише технологію, а й упевненість, що її розробники швидко реагують на нові виклики. Успіх GENI визначатиметься не лише технічною досконалістю, а й умінням команди презентувати продукт як надійне рішення, що реально економить кошти й час компаній.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Загальна комп'ютеризація суспільства призвела до того, що використання комп'ютерів стало повсюдним у всіх сферах економіки та народного господарства. Застосування персональних комп'ютерів і ЕОМ дозволило значно підвищити продуктивність праці, змінити характер і зміст праці [4]. Комп'ютеризація, поряд з незаперечними перевагами, тягне за собою і багато проблем. Для того, щоб активне застосування комп'ютерних технологій не стало додатковим чинником погіршення здоров'я, вкрай необхідно щоб робоче місце відповідало гігієнічним вимогам. Темою дипломного проекту є розробка та дослідження та реалізація програмного продукту, тому актуально буде розгляд умов праці програміста.

8.1 Аналіз умов праці на робочому місці програміста

Робота програміста пов'язана з постійною роботою на ЕОМ, яка відбувається у кімнаті розмірами 5 м×7,2 м×2,8 м. Одна з її більших стін має шість двостулкових вікон, розмірами 2,2 м×1,8 м, які виходять на північний схід. Вікна розташовані рівномірно по всій довжині стіни. Підлога в кімнаті покрита лінолеумом, всі стіни пофарбовані світло оранжевого кольору до висоти 2,8 м, а далі підвісна стеля. Уздовж стін розташовані комп'ютерні столи. На них розташовуються 2 персональні комп'ютери й інша оргтехніка (сканер принтери, телефони й ксерокс). Столи мають пластикове покриття. Габарити їхньої робочої поверхні 1250 мм×850 мм. Висота столів 750 мм. Висота стільців від рівня підлоги становить 430 мм.

Згідно НПАОП 0.00-1.28-10 «Правила охорони праці під час електронно-обчислювальних машин» площа повинна задовольняти умові – не менш 6 м^2 на одне робоче місце. Кратність повітрообміну в приміщенні вузла також регламентується ДСанПіН 3.3.2.007 – 98, вона повинна становити 20 м^3/год на

одне місце. Виконання даних вимог забезпечить підтримку в приміщенні вузла оптимального значення вологості й складу повітря.

Відповідно ДБН В.2.5–28–2006 роботу програміста можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення вузла можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при сполученому висвітленні), повинен становити 0,5%, освітленість при штучному висвітленні повинна становити 300 лк.

За результатами виміру освітленості відділом охорони праці величина освітленості від системи загального штучного висвітлення лежить у межах 200-250 лк, що не відповідає вимогам, які пред'являються до приміщення.

Відповідно ДСанПіН 3.3.2.007-98 рівні звукового тиску в робочому приміщенні не повинні перевищувати в октавних смугах із середньо геометричними частотами наступних значень, наведених у таблиці 8.1.

У приміщенні перебувають наступні джерела шуму: електродвигуни внутрішнього вентилятора ЕОМ; працюючі принтери; працюючі дисководи. Шум, вироблений вентилятором можна класифікувати як постійний, всі інші джерела шуму, як імпульсні. Відповідно паспорта на приміщення рівень звуку, дБА, обмірюваний за шкалою (А) шумоміра досяг величини 28,3 дБА при роботі всього встаткування вузла, включаючи й ксерокс. Це дозволяє зробити висновок про відповідність рівня звуку в приміщенні вимогам нормативних актів.

Ергономічні вимоги до робочого місця працюючого з ВДТ ЕОМ і ПЕОМ нормуються НПАОП 0.00 – 1.28 – 10**.** Оптимальне положення тіла того, що працює забезпечується відповідною конструкцією робочого місця, а також регуляцією висоти робочої поверхні, сидіння, простори й підставки для ніг.

Даного місця програміста не мають регульованих параметрів. Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту дані в таблиці 8.2.

Таблиця 8.1 – Допустимі спектри рівнів звукового тиску

Робоче місце	Рівень звукового тиску, дБ, в октавних смугах із середньогеометричними частотами, Гц								Рівень шуму і еквівалентний рівень шуму, дБА
	63	125	250	500	1000	2000	4000	8000	
Приміщення конструкторських бюро, програмістів обчислювальних машин, лабораторій для теоретичних робіт	71	61	54	49	45	42	40	38	50

Таблиця 8.2 – Відмінності реальних параметрів робочого місця від параметрів відповідні вимоги нормативного акту

Ріст людини, см	Висота робочої поверхні мм,	Висота простору для ніг, мм	Висота робочого сидіння, мм
175	765 (740)	655 (600)	450 (440)

У дужках зазначені реальні значення параметрів робочого місця; всі вони не відповідають параметрам, зазначеним у стандарті.

Параметри мікроклімату можуть мінятися в широких межах, тоді як необхідною умовою життєдіяльності людини є підтримка сталості температури тіла завдяки властивості терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище.

У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах ДСН 3.3.6.042 – 99 встановлені величини параметрів мікроклімату, що створюють комфортні умови [1]. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення (див. табл. 8.3).

Таблиця 8.3 – Параметри мікроклімату для приміщень, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні	22 – 24°С
	Відносна вологість	40 – 60%
	Швидкість руху повітря	до 0,1 м/с
Теплий	Температура повітря в приміщенні	23 – 25°С
	Відносна вологість	40 ... 60%
	Швидкість руху повітря	0,1 ... 0,2 м/с

8.2 Розробка заходів з умов поліпшення охорони праці

Перерахуємо проведені заходи щодо забезпечення умов праці на робочому місці програміста.

З точки зору забезпечення електробезпеки до цих заходів можна віднести: устаткування розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв; періодична перевірка всіх приладів і пристроїв; щорічна здача іспитів з охорони праці.

З точки зору забезпечення оптимальних умов мікроклімату, рівня звуку і освітленості до цих заходів можна віднести: організацію природної вентиляції, за допомогою дефлектора, для забезпечення необхідного повітрообміну в приміщенні вузла; організацію системи центрального опалювання, для підтримки оптимальної температури в холодний період року; організацію штучного загального освітлення, для забезпечення необхідних умов зорової роботи, що

відповідають, оформлення паспорта на приміщення вузла, з занесенням в нього вимірювань освітленості і рівня звуку, проведених відділом охорони праці.

З точки зору забезпечення пожежної безпеки до цих заходів можна віднести наявність схеми евакуації з приміщення вузла, у випадку пожежі, повішену на вхідні двері.

Аналіз умов праці на робочому місці інженера-програміста показав, що на робочому місці не виконуються вимоги ергономіки. Для виконання їх можна запропонувати заміну не регульованого сидіння на крісло з регульованими ергономічними параметрами, а також заміну використовуваного столу на робоче місце оператора ЕОМ.

Різними фірмами в сукупності розроблено понад 11 схем регулювань параметрів робочого крісла, які забезпечують: плавне переміщення сидіння по висоті за допомогою газової пружини; плавна зміна нахилу спинки і сидіння; регулювання пружинного противотиску спинки крісла на спину оператора; перестановку спинки по висоті; зміна глибини сидіння шляхом зміни вигину краю сидіння; синхронне повторення рухів оператора сидінням і спинкою в правильному кутовому співвідношенні; синхронне повторення спинкою крісла рухів верхньої частини тулуба того, що сидить; амортизацію сидіння [4].

Найбільш популярними моделями комп'ютерних крісел, які володіють чотирма основними регулюваннями (висоти сидіння, висоти, глибини і нахилу спинки), є італійські, фінські моделі «Senior», «Volos», «Ergo», «Toronto», «Bini», «Metro», «NewStar», «Capris», «Fenix», «Xenus», «Quintus» і тому подібні. Крісла такої конструкції, відрегульовані відповідно до зростання і ваги оператора, а також характеру виконуваної роботи, дозволяють знизити навантаження на опорно-руховий апарат людини, що працює за комп'ютером.

8.3 Дослідження інформаційного навантаження на програміста

Програміст, у залежності від підготовки і досвіду, вирішує задачі різної складності, але в загальному випадку робота програміста будується по алгоритму, представленому у таблиці 8.4.

Таблиця 8.4 – Загальний алгоритм виконання завдань програмістом

Етап	Зміст	Витрата часу, %
I	Постановка задачі.	6.25
II	Вивчення матеріалу за поставленою задачею	
III	Визначення методу рішення задачі	6.25
IV	Складання алгоритму рішення задачі	12.5
V	Програмування	25
VI	Налагодження програми, складання звіту	50

Даний алгоритм відображає загальні дії програміста при рішенні поставленої задачі незалежно від її складності.

Підрахуємо кількість членів алгоритму і їхню частоту (імовірність) щодо загального числа, прийнятого за одиницю. Імовірність повторення і-ої ситуації визначається по формулі:

$$P_i = k/n,$$

де k – кількість повторень кожного елемента одного типу, n – сумарна кількість повторень від джерела інформації, одного типу.

Таблиця 8.5 – Робота програміста

Етап	Член алгоритму	Зміст роботи	Літерне позначення
I	1	Одержання першого варіанта технічного завдання	A
	2	Складання і уточнення технічного завдання	Ri
	3	Одержання остаточного варіанта технічного завдання	Сіл
	4	Складання переліку матеріалів, що	HII2
II	5	Вивчення матеріалів по тематиці задачі	Аг
	6	Вибір метолу рішення	C2J3
	7	Уточнення й узгодження обраного методу	B2
III	8	Остаточний вибір методу рішення	СчТ4
	9	Аналіз вхідної і вихідної інформації	B2
	10	Вибір мови програмування	C4TS
	11	Визначення структури програми	H3C5q
IV	12	Складання блок-схеми програми	C6q2
V	13	Логічний аналіз програми і коректування її	F1H4W2
	14	Компіляція програми	F2
	15	Виправлення помилок	D1W2
VI	16	Редагування програми	F2H5B3W
	17	Виконання програми	F3
	18	Аналіз результатів виконання	H6W5
	19	Тестування	C7W6
	20	Підготовка звіту про роботу	F4

Таблиця 8.6 – Результати розрахунку зведемо в таблицю:

Джерело	Члени алгоритму	Символ	Кількість	Частота повторень
1	Аферентні – усього		6	1,00
	Вивчення технічної	А	2	0,33
	Спостереження	Р	4	0,67
2	Еферентні – усього		18	1,00
	Уточнення	В	3	0,17
	Вибір найкращого	С	8	0,44
	Виправлення	0	1	0.06
	Аналіз отриманих	н	6	0,33
	Виконання	к	0	0
3	Логічні умови		13	1,00
	Прийняття рішень на основі вивчення	І	5	0,39
	Графічні матеріали	Ч	2	0.15
	Отриманого тексту	V	6	0.46
	Усього:		37	

Кількісні характеристики (Табл. 8.6) дозволяють розрахувати інформаційне навантаження програміста [8]. Ентропія інформації елементів кожного джерела інформації розраховується по формулі, біт/сигн:

$$H_j = -\sum_{i=1}^{m} p_i \log_2 p_i \quad \text{біт/сигн}$$

де m – число однотипних членів алгоритму розглянутого джерела інформації.

$H_1 = 2 \times 2 + 2 \times 4 = 12$ біт/сигн,

$H_2 = 3 \times 1{,}585 + 8 \times 3 + 0 + 6 \times 2{,}585 = 44{,}265$ біт/сигн,

$H_3 = 5 \times 2{,}323 + 2 \times 1 + 6 + 2{,}585 = 29{,}125$ біт/сигн.

Потім визначається загальна ентропія інформації, біт/сигн:

$$H_s = H_1 + H_2 + H_3,$$

де H_I, H_2, H_3 – ентропія аферентних, еферентних елементів і логічних умов відповідно.

$$H_s = 10 + 44{,}265 + 29{,}125 = 83{,}39.$$

Далі визначається потік інформаційного навантаження біт/хв,

$$\Phi = \frac{H \cdot N}{t}$$

де

N – сумарне число всіх членів алгоритму;

t – тривалість виконання всієї роботи, хв.

Від кожного джерела в інформації (члена алгоритму) у середньому надходить 3 інформаційних сигнали в годину, час роботи – 225 годин,

$$\Phi = \frac{83{,}39 \cdot 37 \cdot 3 \cdot 225}{13500} = 2{,}6 \text{ біт/хв},$$

Розраховане інформаційне навантаження порівнюється з припустимою. При необхідності приймається рішення про зміни в трудовому процесі.

Умови нормальної роботи виконуються при дотриманні співвідношення:

$$\Phi_{\text{доп.мін}} < \Phi_{\text{розр}} < \Phi_{\text{доп.макс}}$$

де

$\Phi_{\text{доп.мін}}$, і $\Phi_{\text{доп.макс}}$ мінімальний і максимальний припустимі рівні інформаційних навантажень (0,8 і 3,2 біт/с відповідно);

$\Phi_{\text{розр}}$ – розрахункове інформаційне навантаження 0,8 < 2,6 <3,2 відповідає нормі.

Висновки до розділу

У цій частині дипломної проекту були розглянуті вплив факторів трудового і виробничого середовища програмістів, дослідження інформаційного навантаження на програміста. Дотримання умов, що визначають оптимальну організацію робочого місця програміста і навантаження, отримані ним в процесі роботи, дозволить зберегти гарну працездатність протягом усього робочого дня, підвищить, як у кількісному, так і в якісному відносинах продуктивність праці програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи управління у розподілених Cloud-системах GENI.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів управління у розподілених Cloud-системах GENI.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем управління у розподілених Cloud-системах GENI.

– Досліджена система управління у розподілених Cloud-системах GENI.

– На основі отриманих результатів досліджень створена програмна реалізація системи управління у розподілених Cloud-системах GENI.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання управління у розподілених Cloud-системах GENI.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 7624:2014.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маламуж В.С. Дослідження та програмна реалізація системи управління у розподілених Cloud-системах GENI // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

2. Оліфер В.Г. Комп'ютерні мережі. Принципи, технології, протоколи. Підручник / В.Г. Оліфер, Н.А.Оліфер. – [5-е вид.]. – 2016. – 944 с.

3. Е. Таненбаум, Д. Уезеролл «Комп'ютерні мережі». – [5-е вид.]. – 2016. – 960 с.

4. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.

5. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.

6. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

7. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

8. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

9. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

10. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп'ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка» (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.

11. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». International Review on Modelling and Simulations 18 (1), 2025. pp. 32-42.

12. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». Кібербезпека: освіта, наука, техніка. 2024. №4(24), С. 6-27.

13. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». Підводні технології, 2024, № 13, с. 28-35.

14. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». CEUR Workshop Proceedings, 2023, 3628, pp. 106-115.

15. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». Advanced Information Systems, 2023, 7(2), pp. 49-56.

16. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». CEUR Workshop Proceedings, Volume 3530, 2023, pp. 256-265.

17. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

18. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

19. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

20. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». Проблеми інформатизації та управління, № 2(70). 2022. С. 28-37.

21. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Системи управління, навігації та зв'язку, 2022, № 3(69). С. 93-98.

22. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Вісник Хмельницького національного університету. Серія: «Технічні науки», № 2 (307). С. 46-52. 2022.

23. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» Системи управління, навігації та зв'язку, 2022, № 1(67). С. 84-89.

24. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». Communications in Computer and Information Science, 2021, vol 1486. Springer, Cham. pp 169-184.

25. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

26. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». Journal of theoretical and applied information technology Vol.98. No 21, 2020, P. 3334-3346.

27. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

28. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

29. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». CEUR Workshop Proceedings Volume 2616, 2020, Pages 125-136.

30. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

31. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

32. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties».

International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

33. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

34. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

35. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

36. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

37. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

38. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 353-358.

39. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising

Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 347-352.

40. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

41. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», Telecommunications and Radio Engineering. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

42. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». Сучасні інформаційні системи. 2021. Т. 5, № 4. С. 79-95

43. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

44. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

45. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

46. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у Кібербезпека та інформаційні технології: монографія. – Х.: ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

47. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

48. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

49. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

50. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології: монографія / за заг. ред. В. С. Пономаренка. – Х.: Вид. Рожко С.Г. 2019. С. 123-139

51. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

52. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.