

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
“___” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи програмно-визначасмого
сховища для NVMe”

Виконав здобувач вищої освіти
IV курсу, групи KI-22-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Третяк К.С.
«___» _____ 2026 р.

Керівник проекту
доктор філософії (PhD)
_____ Дреєва Г.М.
«___» _____ 2026 р.
Рецензент _____

Центральноукраїнський національний технічний університет
Факультет Механіко-технологічний
Кафедра Кібербезпеки та програмного забезпечення
Освітній ступінь бакалавр
Галузь знань 12 "Інформаційні технології"
Спеціальність 123 "Комп'ютерна інженерія"
Освітньо-професійна (освітньо-наукова) програма "Комп'ютерна інженерія"

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 27 » лютого 2026 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Третяку Кирилу Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи програмно-визначаємого сховища для NVMe*

2. Керівник роботи *Дреєва Ганна Миколаївна, доктор філософії (PhD)*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 133-02 від 27.02.2026 року

3. Строк подання студентом роботи до захисту *23.05.2026 р.*

4. Мета та завдання випускної кваліфікаційної роботи: *Метою роботи є розробка програмного забезпечення системи програмно-визначаємого сховища для NVMe*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2. Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи *1 аркуш*

Функціональна схема системи *1 аркуш*

Діаграма процесів *1 аркуш*

Блок-схема алгоритму роботи додатку *2 аркуша*

7. Дата видачі завдання « 27 » лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2026 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2026 р.	
3.	Розробка моделі компонента	20.03.2026 р.	
4.	Розробка структур даних	25.03.2026 р.	
5.	Розробка алгоритмів зв'язку та відображення	30.03.2026 р.	
6.	Програмування алгоритмів	10.04.2026 р.	
7.	Оформлення ПЗ	17.04.2026 р.	
8.	Попередній захист роботи	23.05.2026 р.	

Дата видачі завдання
« 27 » лютого 2026 р.

Підпис керівника

Дреєва Г.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 27 » лютого 2026 р.

Підпис здобувача

Третяк К.С.
(прізвище та ініціали)

АНОТАЦІЯ

Третяк К.С. Програмне забезпечення системи програмно-визначаємого сховища для NVMe. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2026.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи програмно-визначаємого сховища для NVMe.

Метою розробки є програмне забезпечення системи програмно-визначаємого сховища для NVMe.

Результат роботи – програмна реалізація системи програмно-визначаємого сховища для NVMe.

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 Visual C#.

Ключові слова: комп'ютерна інженерія, програмно-визначаєме сховище, NVMe

ABSTRACT

Tretiak K.S. Software for a software-defined storage system for NVMe. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for a software-defined storage system for NVMe.

The purpose of the development is software for a software-defined storage system for NVMe.

The result of the work is a software implementation of a software-defined storage system for NVMe.

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

A convenient user interface has been developed. Instructions for working with software are provided.

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

The program was developed in the Visual C# environment.

Keywords: computer engineering, software-defined storage, NVMe

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ	2
ВСТУП.....	3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ	4
1.1 Призначення системи.....	4
1.2 Область застосування.....	5
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ	7
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.....	7
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування.....	10
2.3 Розгорнута постановка завдання	14
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ	16
3.1 Опис функціонування системи	16
3.2 Розробка структурної схеми.....	22
3.3 Розробка функціональної схеми	31
3.4 Розробка діаграми процесів.....	42
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ.....	44
4.1 Розробка блок-схем та опис алгоритмів функціонування системи.....	44
4.2 Захист розробленого програмного забезпечення.....	56
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ	59
6 ОСНОВНІ ВИСНОВКИ.....	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	69

					ВКРБ-123.26.0027.00.00.ПЗ			
Вим.	Арк.	№ докум.	Підп.	Дата	Програмне забезпечення системи програмно-визначаємого сховища для NVMe	Літ.	Аркуш	Аркушів
Розроб.	Третяк К.С.					Б	1	76
Перев.	Дресва Г.М.							
Н.контр.	Коваленко А.С.					ЦНТУ КІ-22-1		
Затв.	Смірнов О.А.							

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АБГШ	—	аддитивний білий гаусів шум
БЧХ	—	код Боуза-Чоудхури-Хоквінгема
ЕОМ	—	електронно-обчислювальна машина
ІМС	—	інтегральна мікросхема
СЗД	—	системи зберігання даних
ARQ	—	автоматичний запит повторної передачі
CD-DA	—	Compact Disc Digital Audio
CIRC	—	Cross Interleaved Reed Solomon Code
EAB	—	Embedded Array Block, блок зосередженої пам'яті
ECC	—	error-correcting code, код корекції помилок
FEC	—	метод прямої корекції помилок
LDPC	—	коди Галлагера
NAK	—	негативне підтвердження

ВСТУП

Актуальність теми. Рішення програмно-визначеного сховища (SDS) – це альтернативні програми зберігання даних, де ресурси сховища відокремлені від базового фізичного обладнання. Це робить їх більш гнучкими, ніж традиційні програми мережевого сховища даних (SAN) або мережевого сховища даних (NAS), які підключені до обладнання. Програми SDS зазвичай працюють або на операційній системі сервера (ОС), або на віртуальних машинах (ВМ). Але деякі програми можуть працювати на ядрі гіпервізора, що дозволяє одночасно використовувати кілька віртуальних машин на одному пристрої. Ці програми також пов'язані з архітектурою програмно-визначених мереж (SDN) та програмно-визначених центрів обробки даних (SDDC). Ці продукти відображають ширший галузевий зсув до віртуалізації програмно-визначених функцій, таких як зберігання даних, мережа та безпека.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи програмно-визначаємого сховища для NVMe.

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

- Огляд існуючих систем програмно-визначаємого сховища для NVMe.
- Дослідження системи програмно-визначаємого сховища для NVMe.
- Програмна реалізація системи програмно-визначаємого сховища для NVMe.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі програмно-визначаємого сховища для NVMe.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи програмно-визначаємого сховища для NVMe, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		3

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Рішення SDS стали дуже популярними останнім часом з кількох ключових причин, зокрема:

- еволюціонуючий та дедалі більш неструктурований характер даних;
- зростаюча потреба клієнтів у можливостях віртуалізації даних;
- зростаючий попит на технології хмарного зберігання даних.

Типові випадки використання SDS включають:

- DevOps.
- Автоматизація робочих процесів.
- Віртуалізована інфраструктура робочих столів (VDI).
- Безпека та захист даних.
- Управління даними інтегрованих публічних та приватних хмарних систем.
- Забезпечення кращого доступу до даних для віддалених офісів.
- Підвищення ефективності зберігання даних шляхом усунення ізольованих структур.
- Подовження терміну служби більш традиційних систем зберігання.
- «Масштабування назовні» проти «масштабування нагору» ємностей сховища.

«Масштабування назовні» – це альтернатива більш традиційному методу збільшення ємності сховища. Воно передбачає додавання додаткового місця для зберігання за допомогою продуктів SDS, які не потребують фізичного під'єднання до апаратного забезпечення, що знаходиться нижче.

Масштабування сховища передбачає додавання більшої кількості фізичного сховища, такого як драйвери дисків, до початкової інфраструктури

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		4

сховища для збільшення ємності. Одним із недоліків масштабування сховища є обмежений обсяг можливостей оновлення. Для порівняння, при масштабуванні сховища обсяг ресурсів сховища, які можна додати, практично не обмежується.

1.2 Область застосування

Більшість продуктів SDS містять низку функцій та можливостей, спрямованих на управління сховищами та їхню функціональність, таких як:

- Абстракція ресурсів зберігання даних від базового обладнання.
- Віртуалізація об'єднаних ресурсів сховища.
- Можливість керування об'єднаним сховищем через один адміністративний інтерфейс.
- Керування ресурсами зберігання на основі політик.
- Можливість збільшення ємності сховища шляхом масштабування системи на кластер серверів.
- Сумісність зі стандартним галузевим обладнанням (серверами x86 та API).
- Доступ до різних типів інтерфейсів сховища, таких як блокове, файлове та об'єктне сховище.

Однак, під час вибору продукту SDS слід враховувати кілька варіантів сховища. До них належать різні варіанти масштабованого та/або блокового, файлового та об'єктного зберігання, програмне забезпечення на основі контейнерів, віртуалізоване програмне забезпечення та програмне забезпечення для гіперконвергентної інфраструктури (HCI).

Переваги SDS

Використання рішення SDS має кілька переваг порівняно з традиційними сховищами типу SAN або NAS. Найголовніше, що вони дозволяють користувачеві оновлювати ємність сховища, не змінюючи базове фізичне обладнання. Вони також дають змогу розширювати існуючу ємність сховища

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		5

практично до необмеженого ступеня. Продукція SDS також пропонує переваги, пов'язані з вартістю, які включають:

- нижчі початкові та експлуатаційні витрати;
- покупці уникають «прив'язаності» до одного постачальника програмного забезпечення (оскільки вони не прив'язані до жодного конкретного обладнання);
- усунути витрати шляхом «масштабування» поточної ємності сховища та додавання додаткових ресурсів сховища за потреби.

Ціноутворення

Стандартна модель ціноутворення для програмного забезпечення для зберігання даних передбачає ціноутворення за гігабайт, разом із вартістю встановлення та оновлення. Однак постачальники SDS пропонують продукти, ціни на які більше залежать від варіанту використання, тривалості використання та необхідного обсягу сховища. Тому ціни можуть бути не дуже стандартизованими.

Продавці зазвичай не розголошують інформацію про ціни на своєму веб-сайті, але надають її за запитом.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи програмно-визначаємого сховища для NVMe, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		6

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Постачальники сховищ зі штучним інтелектом

До 2026 року ринок штучного інтелекту для зберігання даних буде розділений на кілька чітких таборів. У реальних проектах команди зазвичай вибирають щонайменше дві з цих категорій та об'єднують їх, а не зосереджують все на одній платформі.

Платформи штучного інтелекту, повністю засновані на флеш-пам'яті

Постачальники цієї групи створюють системи лише на флеш-пам'яті, призначені для ферм графічних процесорів, великих навчальних кластерів та проектів «фабрик штучного інтелекту».

– VAST Data позиціонує свою платформу як «операційну систему штучного інтелекту», яка об'єднує сховище, базу даних та обчислення для великомасштабного штучного інтелекту та високопродуктивних обчислень. Її дезагрегована, масштабована флеш-архітектура забезпечує повну потужність графічних процесорів.

– Pure Storage FlashBlade//S – уніфікована файлово-об'єктна система, яка часто використовується в референтних архітектурах NVIDIA DGX та «AI Factory» для високопродуктивного навчання та аналітики ШІ.

Вам слід звернутися до цієї категорії, коли вам потрібна дуже висока продуктивність та відносно просте керування для великих, централізованих навчальних середовищ.

Зберігання об'єктів та озера даних штучного інтелекту

Ці платформи розташовані нижче як рівень ємності для неструктурованих

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		7

даних штучного інтелекту: зображень, відео, журналів, мультимодальних корпусів.

– Cloudian HyperStore – S3-сумісне об'єктне сховище, створене для озер даних штучного інтелекту та аналітики ексабайтного масштабу, з тісною інтеграцією з конвеєрами штучного інтелекту на базі NVIDIA.

– Scality RING – масштабоване об'єктне сховище, що використовується як високоємісне сховище штучного інтелекту з масштабуванням ексабайтного класу та довговічністю для великих неструктурованих наборів даних.

Вам варто використовувати їх, коли довгострокове зберігання, вартість за терабайт та довговічність мають таке ж значення, як і швидкість.

Корпоративні високопродуктивні обчислювальні платформи зберігання даних з «додатковими можливостями» штучного інтелекту

Традиційні постачальники сховищ даних оновили свої лінійки високопродуктивних обчислень (HPC) та корпоративних систем, щоб задовольнити вимоги штучного інтелекту до пропускну здатності та метаданих.

– HPE (Cray ClusterStor, Alletra тощо) – платформи високопродуктивних обчислень та флеш-пам'яті, налаштовані на щільні кластери GPU та великі паралельні завдання.

– IBM Storage Scale + ESS – наступник GPFS, що поєднує паралельну файлову систему з флеш-пристроями для багатовузлових кластерів штучного інтелекту та аналітики.

Це поширене явище в середовищах, які вже стандартизують роботу великого постачальника та хочуть мати сховище, готове до використання штучного інтелекту, без заміни всього.

Програмно-визначена оркестрація та оркестрація даних

Ця група менше зосереджується на базових дисках і більше на створенні єдиної структури даних на основі того, що у вас вже є.

– DataCore Nexus – програмно-визначене сховище, орієнтоване на високопродуктивні файлові сервіси з глобальним простором імен та розміщенням

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		8

на основі політик на різних сайтах та хмарах.

– Hammerspace – платформа даних, яка створює глобальну файлову систему на основі гетерогенного сховища, дозволяючи робочим навантаженням штучного інтелекту бачити один простір імен, тоді як політики переміщують дані між локальними сховищами та хмарами.

Оберіть їх для розподілених, багатосайтових конвеєрів штучного інтелекту або гібридних середовищ, де мобільність даних та єдине управління є критично важливими.

Критерії вибору постачальників сховищ на базі штучного інтелекту

Використовуйте цей контрольний список як швидкий помічник у прийнятті рішень: чесно дайте відповідь на кожен пункт для кожного постачальника, якого ви порівнюєте.

1. Відповідність робочому навантаженню. Чи обробляє платформа ваш основний шаблон: інтенсивний навчальний ввід/вивід, попередня обробка з великим обсягом метаданих або низькозатримковий висновок? Чи є реальні бенчмарки або історії клієнтів, близькі до ваших випадків використання?

2. Відповідність архітектурі. Чи повністю підключається він до ваших поточних або планованих кластерів GPU/CPU (мережа, тканини, модель масштабованого масштабування)? Чи можуть потужність та продуктивність зростати крок за кроком, а не лише за рахунок великих модернізацій навантажувачів?

3. Рівень інтеграції. Чи підтримує він протоколи та інструменти, які ви вже використовуєте (POSIX/NFS/SMB, S3, Kubernetes CSI, планувальники, фреймворки машинного навчання, такі як PyTorch або TensorFlow)? Чи є інтеграція моніторингу та резервного копіювання чіткою та задокументованою?

4. Вартість та фіксація. Чи розумієте ви 3-5-річну сукупну вартість володіння, включаючи ліцензії, підтримку, живлення, охолодження та розширення? Чи спирається вона на відкриті стандарти, такі як S3, NFS або NVMe-oF, щоб ви не опинилися в пастці, якщо вимоги зміняться?

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		9

5. Безпека та постачальник. Чи достатньо надійними є варіанти шифрування, контролю доступу, журналів аудиту та зберігання даних для вашого регуляторного профілю? Чи має постачальник надійну дорожню карту, клієнтів для виробничого III та стабільність для підтримки вас у довгостроковій перспективі?

Найбільш відомі на ринку рішеннями є:

- VMware vSAN.
- HPE StoreVirtual VSA.
- NetApp ONTAP Select.
- EMC ScaleIO.
- Huawei Fusion Storage.
- StarWind Virtual SAN.
- Datacore SANsymphony.

Інструмент майбутнього

Технологія NVMe початку розвиватися ще на початку 2000-х, але поки не змогла замінити класичні СЗД по цілому ряді причин – зараз ми їх обговорювати не будемо. Але виробники активно займаються розвитком своїх продуктів і інтерес до технологій NVMe росте. За нашими оцінками, найближчим часом вони стануть тим інструментом, що дозволить скорочувати вартість ІТ-інфраструктури при росту потреби в збільшенні ємності СЗД.

На закінчення відзначу, що в справжньому матеріалі я не намагався запропонувати варіанти вибору підходящого для вас рішення. Таке рішення потрібно вибирати, виходячи з навантаження, SLA і т.д. Т.е. це саме порівняльний аналіз можливостей, а не продуктивне тестування.

Але сподіваюся, що представлені результати порівняння допоможуть вам зорієнтуватися, заощадять час і полегшать завдання вибору, яке рішення підходить у вашім випадку.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		10

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об'єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних застосунків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські застосунки Windows, веб-служби XML, розподілені компоненти, застосунки типу “сервер-клієнт”, застосунки баз даних і багато яких інших. В Visual C# є розширений редактор коду, конструктори зі зручним користувацьким інтерфейсом, вбудований відладник і багато інших засобів, покликані спростити розробку застосунків мовою Visual C# версії 5.0 і .NET Framework версії 4.5.

Синтаксис Visual C# дуже виразний, але простий у вивченні. Усі, хто знаком з мовами C, C++ або Java з легкістю визнають синтаксис із фігурними дужками, характерний для мови Visual C#. Розроблювачі, що знають кожен із цих мов, як правило, зможуть домогтися ефективної роботи з мовою Visual C# за дуже короткий час. Синтаксис Visual C# робить простіше те, що було складно в C++, і забезпечує потужні можливості, такі як типи значень Nullable, перерахування, делегати, лямбда-вираження й прямий доступ до пам'яті, чого немає в Java. Visual C# підтримує універсальні методи й типи, забезпечуючи більше високий рівень безпеки й продуктивності, а також ітератори, що дозволяють при реалізації колекцій класів визначати власне поводження ітерації, що може легко використовуватися в клієнтському коді. В Visual C# 5.0 вираження LINQ (Language-Integrated Query) роблять строго-типізований запит першокласною конструкцією мови.

Як об'єктно-орієнтована мова, Visual C# підтримує поняття інкапсуляції, спадкування й поліморфізму. Всі змінні й методи, включаючи метод Main – крапку входу застосунка – інкапсулюється у визначення класів. Клас може

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		11

успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово `override`, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримує спадкування.

На додаток до основних описаних об'єктно-орієнтованих принципів, мова Visual C# спрощує розробку компонентів програмного забезпечення завдяки декільком інноваційним конструкціям мови, у число яких входять наступні:

- Інкапсульовані підписи методів, називані делегатами, які підтримують строго-типізовані повідомлення про події.
- Властивості, що виступають у ролі методів доступу для закритих змінних-членів.
- Атрибути з декларативними метаданими про типи під час виконання.
- Вбудовані коментарі XML-документації.
- LINQ (Language-Integrated Query), що пропонує вбудовані можливості запитів у різних джерелах даних.

Якщо буде потрібно забезпечити взаємодію з іншим програмним забезпеченням Windows, таким як об'єкти COM або власні бібліотеки DLL Win32, у мові Visual C# можна використовувати процес, що називається "Interop". Процес Interop дозволяє програмам на Visual C# виконувати практично будь-які дії, які може виконувати вихідний додаток на C++. Мова Visual C# підтримує навіть покажчики й поняття "небезпечного" коду для тих випадків, коли прямий доступ до пам'яті має вкрай важливе значення.

Процес побудови Visual C# у порівнянні з C і C++ простий і є більше гнучким, чим в Java. Немає окремих файлів заголовка, а методи й типи не потрібно повідомляти в певному порядку. У вихідному файлі Visual C# може бути визначене будь-яке число класів, структур, інтерфейсів і подій.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		12

Архітектура платформи .NET Framework

Програма мовою Visual C# виконується в середовищі .NET Framework – інтегрованому компоненті Windows, що містить віртуальну систему виконання (середовище CLR) і уніфікований набір бібліотек класів. Середовище CLR являє собою комерційну реалізацію корпорацією Майкрософт інфраструктури CLI, що є міжнародним стандартом, який лежить в основі створення середовищ виконання й розробки, у яких забезпечується тісна взаємодія між мовами й бібліотеками.

Вихідний код, написаний мовою Visual C#, компілюється в проміжну мову (IL) у відповідності зі специфікацією CLI. Код IL і ресурси, такі як растрові зображення й рядки, зберігаються на диску у файлі, що виконується, названому складанням, з розширенням EXE або DLL у більшості випадків. Складання містить маніфест із відомостями про типи складання, версії, мови й регіональні параметри та вимоги безпеки.

При виконанні програми на Visual C# складання завантажується в середовище CLR залежно від відомостей у маніфесті. Далі, якщо вимоги безпеки дотримані, середовище CLR виконує JIT-компіляцію для перетворення коду IL в інструкції машинного коду. Середовище CLR також надає інші служби, що відносяться до автоматичного збору сміття, обробки виключень і керуванню ресурсами. Код, виконуваний середовищем CLR, іноді називають "керованим кодом" у протиставлення "некерованому коду", що компілюється в машинний код, призначений для певної системи. Далі показані відносини під час компіляції й час виконання між файлами з вихідним кодом Visual C#, бібліотеками класів .NET Framework, складаннями й середовищем CLR.

Взаємодія між мовами є ключовою особливістю .NET Framework. Оскільки код IL, створюваний компілятором Visual C# відповідає специфікації CTS, код IL на основі Visual C# може взаємодіяти з кодом, створюваним версіями мов Visual Basic, Visual C++, Visual J# платформи .NET Framework і ще більш ніж 20 CTS-сумісних мов. В одному складанні може бути кілька модулів, написаних на різних мовах платформи .NET Framework, і типи можуть посилатися один на одного, як

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		13

якби вони були написані на одній мові. Крім служб часу виконання, в.NET Framework також є велика бібліотека, що складається з більш ніж 4000 класів, організованих по просторах імен, які забезпечують різноманітні корисні функції для будь-яких дій, починаючи від введення й виведення файлів для керування рядками для розбивки XML, і закінчуючи елементами керування Windows Forms. У звичайному застосунку мовою Visual C# бібліотека класів .NET Framework інтенсивно використовується для "устрою" коду.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускню кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи програмно-визначаємого сховища для NVMe.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) 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;

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		14

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) сформулювати висновки про виконаний обсяг робіт та одержані результати.

К6ПЗ-2026

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		15

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Високопродуктивні обчислення ефективні настільки, наскільки ефективна тканина, яка їх забезпечує. У 2026 році сховище стало основним фактором рентабельності інвестицій у проекти штучного інтелекту. Коли сховище не може наситити кластер графічних процесорів класу Blackwell, фінансовий вплив є негайним: приблизно 30 000 доларів США на кожен вузол щорічних капітальних витрат та витрат на енергію витрачається на цикли простою. Цей стан, відомий як голод вводу/виводу, є прямим результатом застосування протоколів зберігання даних людського масштабу до робочих навантажень машинного масштабу.

Традиційне сховище використовує модель «масштабування», де пара контролерів керує набором дисків. У 2026 році ця модель застаріла для штучного інтелекту. Сховище зі штучним інтелектом використовує підхід масштабування, де дані розподіляються по масивному кластеру незалежних вузлів. З додаванням вузлів ваша пропускна здатність та кількість IOPS зростають лінійно, запобігаючи проблемам з одним контролером, які є проблемою для застарілих NAS.

Сучасні конвеєри штучного інтелекту читають і записують паралельно. Завдяки використанню NVMe-over-Fabrics (NVMe-oF) та RDMA, дані переміщуються з флеш-носія до пам'яті графічного процесора з затримкою менше мілісекунди. Автоматизовані політики працюють у фоновому режимі, зберігаючи «гарячі» навчальні набори даних на NVMe, водночас прозора переносячи петабайти «холодних» журналів до дешевшого об'єктного сховища без порушення шляху до файлу.

Основні риси сучасного штучного інтелекту для зберігання даних

До 2026 року галузь стандартизує три ключові архітектурні риси для

вирішення проблеми вузького місця між дисками та графічними процесорами:

1. Уніфікований простір імен. Традиційне сховище часто вимагає від адміністраторів ручного переміщення даних між «гарячими» рівнями SSD та «холодними» рівнями HDD. Платформи 2026 усувають цю ручну витрату завдяки єдиному логічному пулу. З точки зору адміністратора, навчальні набори даних, сховища функцій та контрольні точки знаходяться в одному просторі імен. За лаштунками служба метаданих обробляє автоматизоване розміщення та реплікацію даних, гарантуючи, що командам не доведеться мікрокерувати апаратними ізоляціями.

2. Архітектура повністю флеш-пам'яті. III-сховища тепер обов'язково в першу чергу базуються на флеш-пам'яті. NVMe та високощільні SSD-накопичувачі забезпечують необхідну кількість IOPS для потокової передачі величезних пакетів даних на кластери графічних процесорів без зупинок. Видаляючи обертові носії з «гарячого шляху», системи підтримують стабільну затримку навіть за змішаних робочих навантажень, таких як одночасне випадкове читання для навчання та великі послідовні сканування для отримання даних.

3. Дезагрегована компонування інфраструктура. Застарілі системи змушували обчислення та сховище масштабуватися в «заблокованих» блоках. Архітектури 2026 розділяють ці доріжки. Ви можете розширити ємність сховища або пропускну здатність, не торкаючись кластера графічних процесорів, або додати більше обчислювальних вузлів, не замінюючи сховище. Це дозволяє уникнути руйнівних оновлень «великого вибуху» та дозволяє здійснювати детальне та економічно ефективне масштабування.

Отже, що таке штучний інтелект для зберігання даних? Визначення штучного інтелекту для зберігання даних підприємства

III-сховище – це високопродуктивна архітектура даних, розроблена для усунення обмежень вводу/виводу, забезпечуючи величезну паралельну пропускну здатність та субмілісекундну затримку для кластерів GPU. На відміну від традиційних NAS, III-сховище використовує NVMe-over-Fabrics (NVMe-oF)

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		17

та дезагреговану архітектуру для масштабування пропускної здатності незалежно від ємності. Воно розроблене для обробки високих вимог до паралельності великих мультимодальних моделей (LMM) та розподіленого навчання, гарантуючи, що конвеєри з великим обсягом даних, від озер необроблених об'єктів до високошвидкісних векторних індексів, залишаються повністю насиченими для максимальної рентабельності обчислень.

Тенденції 2026 року та типи сховищ на базі штучного інтелекту

Тільки у 2025 році витрати на інфраструктуру штучного інтелекту зросли приблизно на 166%, що відображає зростаючий попит на більші моделі, аналітику в режимі реального часу, мультимодальні архітектури та постійне перенавчання у виробничих конвеєрах штучного інтелекту/MLOps.

Середовища штучного інтелекту рідко покладаються на одну платформу сховища. На практиці команди поєднують кілька типів сховищ, кожен з яких відіграє різну роль у конвеєрі ШІ: збір необроблених даних, підготовка функцій, виконання навчальних завдань та обслуговування моделей у продакшені.

Шар сховища: озера, склади та будинки біля озера:

- Озера даних. Резервуар для необроблених, неструктурованих даних (журналів, зображень, потоків датчиків). У 2026 році вони зазвичай будуються на об'єктному сховищі для економічної ефективності в ексабайтному масштабі.
- Сховища даних. Структуровані репозиторії для очищених, змодельованих даних (інформація ERP/CRM). Ідеально підходить для моделей, що залежать від стабільних, реляційних вхідних даних.
- Озера даних. Поточний золотий стандарт. Вони поєднують гнучкість озера з управлінням сховища, дозволяючи розробці функцій та бізнес-аналітиці працювати на одній платформі.

Рівень продуктивності: файл, об'єкт і блок:

- Високопродуктивне сховище файлів (паралельні файлові системи). Ці платформи забезпечують централізований робочий простір зі звичними шляхами до файлів (POSIX), але використовують NVMe-oF (NVMe over Fabrics) як

базовий транспорт. Це дозволяє спільній файловій системі досягти низької затримки локального блочного сховища, підтримуючи при цьому масивний паралельний процес, необхідний кластерам графічних процесорів. Запускаючи паралельну файлову систему поверх RDMA-сумісної тканини NVMe-oF, команди можуть одночасно насичувати тисячі графічних процесорів без вузьких місць метаданих та накладних витрат на застарілі протоколи, характерних для традиційних SAN або NAS.

– Сховище об'єктів для озер даних. Основа для довгострокового зберігання. Сучасне S3-сумісне об'єктне сховище використовується для зберігання мультимодальних корпусів (відео/аудіо), які «розігріваються» до швидших рівнів лише за потреби.

– Блокове сховище для «гарячого шляху». Незамінний для потреб з найнижчою затримкою, таких як онлайн-сховища функцій та вбудовані індекси для виведення в режимі реального часу. Він забезпечує час відгуку менше мілісекунди, необхідний для виробничих програм RAG (Retrieval-Augmented Generation – генерація з доповненням пошуком).

Хмарне та гібридне сховище зі штучним інтелектом

Гібридні системи дозволяють безперешкодно переміщувати дані між локальними кластерами та хмарними спалахами. Це критично важливо для дотримання вимог 2026 року та контролю витрат: зберігання конфіденційних навчальних даних локально, а використання хмари для еластичних сплесків виводу.

Проблеми зі зберіганням даних на базі штучного інтелекту

Незважаючи на досягнення в апаратному забезпеченні, архітектори все ще стикаються з серйозними перешкодами:

– Зупинки контрольних точок: у міру зростання моделей запис «контрольної точки» (стану моделі) може заморозити кластер на кілька хвилин. Сховище має обробляти масивні пакети запису, не призупиняючи навчальні обчислення.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		19

– Затримка RAG: Сучасні «агентські» робочі процеси вимагають від штучного інтелекту миттєвого запиту до векторної бази даних. Затримка зберігання тут безпосередньо перетворюється на сприйнятливую людиною затримку у відповідях штучного інтелекту.

– Суверенітет даних: Переміщення петабайтів даних у хмару тягне за собою величезні комісії за вихід. Підприємства все частіше впроваджують гібридне сховище штучного інтелекту, щоб забезпечити локальне навчання, використовуючи хмару для пакетів виводів.

– Складність даних: конвеєри штучного інтелекту змішують текст, зображення, аудіо, відео, журнали та дані датчиків, більшість з яких неструктуровані. Це значно ускладнює класифікацію, контроль якості, керування версіями та відстеження «які дані навчили яку модель».

– Масштабування продуктивності: у міру зростання кластерів багато систем стикаються з вузькими місцями метаданих, гарячими точками або шумними сусідами. Графічні та центральні процесори зрештою очікують на операції вводу/виводу, оскільки рівень зберігання даних не може підтримувати стабільність затримки та пропускну здатність у більшому масштабі.

– Контроль витрат: сховище з великою кількістю флеш-пам'яті, тривале зберігання, кілька копій середовища (розробка, тестування, виробництво, відповідність) – все це збільшує потужність та бюджет. Без чітких правил розподілу на рівні та життєвого циклу витрати на сховище зростають швидше, ніж план розвитку штучного інтелекту.

– Ризик безпеки: навчальні дані містять конфіденційну особисту, фінансову та конфіденційну інформацію. Слабкі засоби контролю безпеки різко збільшують ризик порушення безпеки.

Варіанти використання штучного інтелекту для зберігання даних

Сховища, готові до використання штучного інтелекту, вже є частиною щоденних операцій у багатьох галузях промисловості, не лише в технологічних компаніях. Нижче наведено дані з напрямками, де вони мають найбільше

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		20

значення:

– Штучний інтелект у сфері охорони здоров'я. Швидка потокова передача КТ/МРТ-досліджень для діагностики за допомогою штучного інтелекту, безперервний доступ до геномних даних, безпечне зберігання електронних медичних карток та записів телемедицини відповідно до правил дотримання вимог.

– Фінансові послуги. Виявлення шахрайства в режимі реального часу, доступ з низькою затримкою для алгоритмічної торгівлі, довгострокове зберігання для моделювання ризиків.

– Роздрібна торгівля. Рекомендації в режимі реального часу, аналітика Інтернету речей, уніфіковані історії клієнтів для прогнозування ціноутворення, асортименту та попиту.

– Розумне виробництво. Зберігання даних часових рядів для прогнозного обслуговування, порівняння виробництва в реальному часі з еталонними даними, обміну наборами даних для оптимізації процесів та цифрових двійників.

– Автономна мобільність. Збір/відтворення журналів водіння, керування маркованими наборами даних для навчання штучного інтелекту, зберігання телеметрії автопарку для оптимізації маршрутів та прогнозного обслуговування.

– Цифрові медіа. Доступні відео/зображення/аудіо ресурси для автоматичного тегування/редагування, підтримка системи рекомендацій, відстеження поведінки в грі або потоковому переданні в режимі реального часу для персоналізації та контролю шахрайства.

ШІ-сховище зараз є ключовим елементом будь-якого серйозного стеку штучного інтелекту, оскільки класичне сховище не може встигати за зростанням обсягу даних та попитом на графічний процесор. Масштабовані платформи, платформи, що в першу чергу використовують флеш-пам'ять, та розумне поєднання файлових, об'єктних, блочних та хмарних рівнів забезпечують швидкість навчання та логічного висновку без значного збільшення витрат.

Вигода – це більше, ніж просто швидкість. Завдяки чіткому розумінню

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		21

ваших робочих навантажень та дисциплінованому підходу до вибору постачальників, сховище перестає бути вузьким місцем і стає потужним інструментом. Воно перетворює потужність графічного процесора на реальні результати, прискорюючи експерименти та надаючи вашим ініціативам у сфері штучного інтелекту необхідну інфраструктуру.

3.2 Розробка структурної схеми

Затримка завжди була ахіллесовою п'ятою мереж зберігання даних. З обертовими дисками кілька мілісекунд затримки не мали великого значення, оскільки сам фізичний носій був повільним. Але як тільки з'явилися флеш-пам'ять та SSD-накопичувачі, вузьке місце перемістилося з пристрою на стек протоколів та мережу. Навіть з локально підключеними SSD-накопичувачами NVMe програми можуть виконувати операції вводу-виводу за десятки мікрсекунд. Порівняйте це з традиційними протоколами SAN, такими як iSCSI або FCP, де кожен ввід-вивід може призвести до сотень мікрсекунд накладних витрат програмного забезпечення та мережі. Саме цю прогалину вирішує NVMe-oF.

Технічно, NVMe-oF розширює набір команд NVMe по всій мережевій структурі з мінімальною трансляцією. Він уникає рівня емуляції команд SCSI, з якого походить значна частина накладних витрат в iSCSI або Fibre Channel. Натомість NVMe-oF підтримує прямі черги надсилання та завершення між тканинами, дозволяючи запитам вводу/виводу передаватися безпосередньо між програмою та SSD з дуже мінімальним втручанням. Результатом є затримка в діапазоні 20–30 мікрсекунд через тканину, що близько до продуктивності локальних дисків NVMe.

Масштабованість не менш важлива. NVMe був створений з нуля для підтримки масового паралелізму з тисячами черг відправлення та завершення. NVMe-oF зберігає це по всій мережі. Замість однієї вузької черги команд, як у

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		22

застарілих протоколах, програми та хости можуть відкривати виділені черги, безпосередньо пов'язані з ядрами процесора. Така конструкція дозволяє інфраструктурі обробляти мільйони IOPS на хост без неефективності перемикання контексту або блокування черги. Для сучасних багатоядерних серверів, на яких працюють десятки контейнерів або віртуальних машин, це важливо для підтримки передбачуваної продуктивності в масштабі.

Ефективність замикає цикл. У традиційних стеках високі показники IOPS означають високе використання процесора; накладні витрати протоколу зменшують обчислювальні цикли, які повинні бути зарезервовані для програм. NVMe-oF значно зменшує цей ризик. Тести часто показують, що NVMe-oF може забезпечити до 3-4 разів більше IOPS на ядро процесора порівняно з iSCSI, що дозволяє центрам обробки даних консолідувати інфраструктуру без шкоди для продуктивності. Ось чому гіперскейлери та хмарні провайдери розглядають NVMe-oF не лише як перевагу в продуктивності, але й як оптимізацію сукупної вартості володіння.

З точки зору варіантів використання, це важливо в середовищах, де кожна мікросекунда на рахунок:

- Базы даних, що потребують часу відгуку менше мілісекунди за високої швидкості транзакцій.
- Конвеєри навчання штучного інтелекту/машинного навчання, де графічні процесори простоюють, якщо сховище не справляється з роботою.
- Периферійні робочі навантаження, де чутливі до затримки програми (автономні системи, 5G, Інтернет речей) не можуть переносити довгі шляхи зберігання даних.
- Аналітика в режимі реального часу, де потоки вхідних даних повинні оброблятися без вузьких місць.

У всіх цих сценаріях NVMe-oF гарантує, що сховище не буде обмежувальним фактором. Це дозволяє підприємствам проектувати

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		23

інфраструктуру, де мережа поводить майже як флеш-пам'ять із прямим підключенням, але з гнучкістю та масштабованістю спільного сховища.

Вибір правильної тканини: RDMA, Fibre Channel чи TCP?

NVMe-oF – це не окремий протокол, а фреймворк: він визначає, як команди NVMe можуть передаватися через різні мережеві структури. Кожен вид транспорту має свої сильні сторони, обмеження та найкращі сценарії. Розуміння цих компромісів є критично важливим для архітекторів, які хочуть максимізувати продуктивність без надмірного ускладнення операцій.

Коли команди NVMe переміщуються через тканину, вони не переміщуються по мережі в сирому вигляді. Натомість вони загорнуті в легкі контейнери, які називаються капсулами. Капсула може містити лише саму команду або, в деяких випадках, команду та пов'язані з нею дані. Така інкапсуляція дозволяє чітко розширити модель NVMe на основі черг на різні транспортні канали, такі як Fibre Channel, RDMA або TCP. Це додає дуже мало накладних витрат, зберігаючи при цьому ефективність черг прямого надсилання та завершення NVMe, тому NVMe-oF може забезпечити затримки, близькі до затримок локально підключених дисків.

RDMA (RoCE та iWARP)

RDMA (віддалений прямий доступ до пам'яті) є золотим стандартом для низької затримки в NVMe-oF. За своєю природою, RDMA обходить процесор хоста та ядро для передачі даних, переміщуючи дані безпосередньо з пам'яті одного хоста на інший. Це означає, що команда NVMe може бути видана та виконана з мінімальним залученням процесора, що часто призводить до затримок до 10–20 мікросекунд по всій тканині.

– RoCE (RDMA over Converged Ethernet) є найпоширенішим варіантом, але він вимагає безвтратної Ethernet-фабрики (досягається за допомогою мостів центрів обробки даних або PFC). Це може ускладнити проектування мережі та усунення несправностей.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		24

– iWARP, навпаки, працює поверх TCP і не потребує тканини без втрат. Однак, його використання в екосистемі обмежене, і більшість постачальників надають пріоритет RoCE для своїх рішень NVMe-oF.

– InfiniBand – це ще один транспортний метод, який реалізує RDMA безпосередньо. Він поширений у високопродуктивних обчислювальних середовищах, де критично важливими є наднизька затримка та надзвичайно висока пропускна здатність.

Найкращий варіант використання: високопродуктивні кластери, конвеєри штучного інтелекту/машинного навчання, фінансові послуги або будь-яке робоче навантаження, де найнижча можлива затримка не підлягає обговоренню.

Компроміси:

- Потрібні спеціалізовані мережеві карти з підтримкою RDMA.
- Може бути складним для налаштування та усунення несправностей (особливо з RoCE).
- Обмежена сумісність між різними постачальниками в середовищах з кількома постачальниками.

Оптоволоконний канал (FC-NVMe)

Fibre Channel – це надійний робочий інструмент для корпоративного зберігання даних. Завдяки FC-NVMe організації можуть виконувати команди NVMe поверх існуючих FC-фабрик без копіювання та заміни інфраструктури. Для підприємств, які значною мірою інвестують у SAN, це найприродніший спосіб впровадження NVMe-oF.

Перевагами FC є його зрілість, стабільність та інструментарій. Адміністратори сховищ, які роками керували середовищами FC, можуть перейти на FC-NVMe з мінімальним перенавчанням. Продуктивність висока, із затримками зазвичай у діапазоні 50–100 мікросекунд – не так низько, як у RDMA, але все ж таки є значним кроком у порівнянні з застарілим SCSI та FC.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		25

Найкращий варіант використання: підприємства з існуючими розгортаннями FC SAN, які бажають модернізуватися без капітального ремонту своїх мереж.

Компроміси:

- Потрібні адаптери FC HBA та комутатори FC (не можуть використовувати існуючі мережі Ethernet).
- Екосистеми постачальників вужчі порівняно з підходами на основі Ethernet.
- Операційні силоси: мережевим командам може бракувати досвіду в фінансовій сфері, який залишається спеціалізованим набором навичок.

TCP (NVMe/TCP)

Найновіший учасник, NVMe/TCP, використовує прагматичний підхід: він дозволяє передавати команди NVMe через стандартні мережі TCP/IP. Немає спеціалізованих мережових карт, немає вимог до Ethernet без втрат. Якщо у вас є IP-мережа, ви можете розгорнути NVMe/TCP.

Хоча TCP створює більше накладних витрат, ніж RDMA, сучасні функції розвантаження процесорів та мережових карт значно скоротили розрив у продуктивності. Затримка для NVMe/TCP зазвичай знаходиться в діапазоні 100–200 мікросекунд ; вище, ніж у RDMA, але все ще набагато нижче, ніж у iSCSI або застарілих протоколів. Для більшості корпоративних робочих навантажень цього «достатньо швидко», і простота розгортання часто переважає невеликий компроміс у затримці.

Найкращий варіант використання: організації, які бажають отримати переваги NVMe-oF без інвестування в спеціалізоване обладнання чи перебудови архітектури своїх мереж. Ідеально підходить для хмарних середовищ, центрів обробки даних на занедбаних ділянках та платформ, що базуються на Kubernetes.

Компроміси:

- Трохи вища затримка порівняно з RDMA та FC.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		26

– Залежить від процесора для транспортування, що може впливати на продуктивність за дуже великих навантажень (хоча розвантаження процесора та мережевої карти розвиваються для вирішення цієї проблеми).

– Екосистема все ще розвивається порівняно з RDMA та FC.

Рішення щодо тканини полягає не в тому, «яка найкраща загалом», а в тому, «яка найкраще підходить для мого робочого навантаження та середовища».

– Якщо наднизька затримка є критично важливою, і ви маєте навички керування безвратною Ethernet-фабрикою, оберіть RDMA (RoCE).

– Якщо у вас вже є стабільна FC SAN, FC-NVMe – це шлях з найменшим тертям.

– Якщо простота та широке впровадження важливіші, ніж вичавлювання останньої мікросекунди, NVMe/TCP – це найкращий вибір, що гарантує майбутнє.

На практиці багато організацій застосовують гібридний підхід: RDMA для своїх високопродуктивних кластерів, TCP для контейнерного сховища в Kubernetes та FC-NVMe для продовження терміну служби своїх інвестицій у SAN.

NVMe-oF у сучасних архітектурах

Справжня сила NVMe-over-Fabrics проявляється не лише в бенчмарках, але й у тому, як вона змінює дизайн сучасної інфраструктури. Розповсюджуючи характеристики низької затримки NVMe по всій мережі, NVMe-oF усуває одне з останніх великих вузьких місць у обчисленнях, орієнтованих на дані: продуктивність спільного сховища. Цей зсув впливає на кілька архітектурних моделей одночасно – від тісно інтегрованих кластерів до масово паралельних суперкомп'ютерних систем. Нижче ми розглянемо чотири ключові області, де NVMe-oF стає основоположним:

Гіперконвергентна інфраструктура (HCI)

Гіперконвергентні інфраструктурні проекти об'єднують обчислювальні, сховищні та мережеві ресурси в єдину систему. Проблема завжди полягала в

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		27

тому, що після спільного використання сховища між вузлами страждає стабільність продуктивності. Традиційні стеки створюють вузькі місця через накладні витрати протоколів та неефективні шляхи вводу/виводу.

Завдяки NVMe-oF вузли в кластері можуть надавати доступ до своїх локальних дисків NVMe вузлам-партнерам майже без додаткової затримки. Черги надсилання та завершення можна відображати по всій структурі, тому віддалений доступ майже ідентичний локальному доступу. На практиці це перетворює колекцію дисків, розкиданих по серверах, на єдиний високопродуктивний пул сховищ.

Це має дві основні переваги: робочі навантаження зі суворими вимогами до затримки можуть виконуватися безпосередньо на HCI без потреби в окремій SAN, а продуктивність лінійно масштабується в міру додавання вузлів. Для змішаних середовищ, що використовують бази даних, аналітичні механізми та віртуальні робочі столи, це усуває один з найбільших компромісів гіперконвергенції.

Програмно-визначене сховище

Платформи програмно-визначених сховищ (SDS) об'єднують сховище на кількох вузлах у логічний пул, абстрагований та керований програмним забезпеченням. Ахіллесовою п'ятою завжди була мережа: незалежно від швидкості дисків, міжвузловий зв'язок визначає загальну продуктивність.

NVMe-oF допомагає системам SDS досягати характеристик, близьких до локальних. Завдяки скороченню накладних витрат на тканину, запит на читання або запис, що проходить через вузли, має затримку в десятки мікросекунд, а не в сотні. Це дозволяє SDS підтримувати робочі навантаження, чутливі до затримки, які раніше були призначені для виділених масивів.

Паралелізм протоколу також підтримує середовища з кількома клієнтами або кількома програмами. Тисячі черг на відправку та завершення можуть бути призначені кожному клієнту або робочому навантаженню, що зменшує конкуренцію та ефекти шумних сусідів. На практиці це означає передбачувану

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		28

продуктивність, навіть коли десятки незалежних клієнтів використовують один розподілений пул сховища.

Паралельні файлові системи

У високопродуктивних обчисленнях та масштабній аналітиці даних паралельні файлові системи дозволяють тисячам клієнтів одночасно отримувати доступ до одного й того ж набору даних. Ці системи часто мають вузькі місця не через швидкість передачі даних, а через затримку та пропускну здатність мережі, що з'єднує обчислення та сховище.

NVMe-oF вирішує цю проблему, забезпечуючи прямий доступ з низькою затримкою від обчислювальних вузлів до цільових сховищ на базі NVMe. Замість того, щоб запити вводу-виводу проходили через кілька рівнів трансляції, команди видаються безпосередньо через всю структуру. Завдяки транспортуванню RDMA затримки можуть сягати десятків мікросекунд навіть при масштабуванні до тисяч вузлів. Завдяки транспортуванню TCP організації можуть розгортати паралельні файлові системи через стандартний Ethernet, досягаючи при цьому стрибків продуктивності порівняно зі застарілими NFS або iSCSI.

Результатом є ефективніше використання обчислювальних кластерів. Центральні та графічні процесори витрачають менше часу на очікування даних і більше часу на їх обробку. Для наукових симуляцій, навчання великомасштабних моделей штучного інтелекту або аналізу наборів даних петабайтного масштабу ці покращення безпосередньо скорочують час отримання результатів.

Контейнерне сховище

Контейнери за своєю суттю є ефемерними, але програми, які вони запускають, часто такими не є. Робочі навантаження з відстеженням стану, такі як бази даних, системи обміну повідомленнями та конвеєри штучного інтелекту, потребують постійного сховища, яке може відповідати гнучкості моделі контейнера.

NVMe-oF дозволяє платформам зберігання даних, що базуються на контейнерах, надавати доступ до постійних томів з таким самим низьким

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		29

профілем затримки, як і локальні диски NVMe, зберігаючи при цьому гнучкість спільної інфраструктури. Поди можуть динамічно підключати та відключати блочні томи, при цьому час відгуку вимірюється в мікросекундах замість мілісекунд.

Оскільки підтримка NVMe-oF вже інтегрована в сучасні операційні системи, драйвери контейнерних сховищ можуть реалізувати її без додаткових рівнів емуляції. Це зменшує складність, водночас гарантуючи, що високопродуктивні робочі навантаження (наприклад, бази даних зі збереженням стану в кластерах Kubernetes) більше не потребують компромісу між гнучкістю та швидкістю.

Справжня історія NVMe-over-Fabrics не про набори команд чи мікросекунди, скорочені на шляху вводу/виводу. Йдеться про те, як розвивається інфраструктура, коли сховище перестає бути обмежувальним фактором. Як тільки сховище може масштабуватися паралельно з обчисленнями та мережею, з'являються нові шаблони проектування – архітектури, які є більш гнучкими, ефективними та узгодженими з тим, як програми фактично потребують даних.

Потужність NVMe-oF полягає в тому, що він іде на другий план. Програмам не потрібно знати, чи є їхні дані локальними чи віддаленими; розробникам не потрібно йти на компроміс між гнучкістю та продуктивністю; архітекторам не потрібно вибирати між ефективністю та масштабуванням. Коли NVMe-oF встановлено, структура сховища просто не відстає.

Заглядаючи в майбутнє, роль NVMe-oF, ймовірно, посилиться, оскільки нові прискорювачі, інтелектуальні мережеві пристрої та пам'яттєво-семантичні тканини потраплять у центри обробки даних. Але його мета залишиться незмінною: усунення відстані як обмеження, щоб дані могли переміщуватися так швидко та безперешкодно, як того вимагають сучасні робочі навантаження. Для організацій питання не в тому, чи швидший NVMe-oF. Питання в тому, чи готові вони розробляти системи, які повною мірою використовуватимуть переваги світу, де продуктивність сховищ більше не є вузьким місцем.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		30

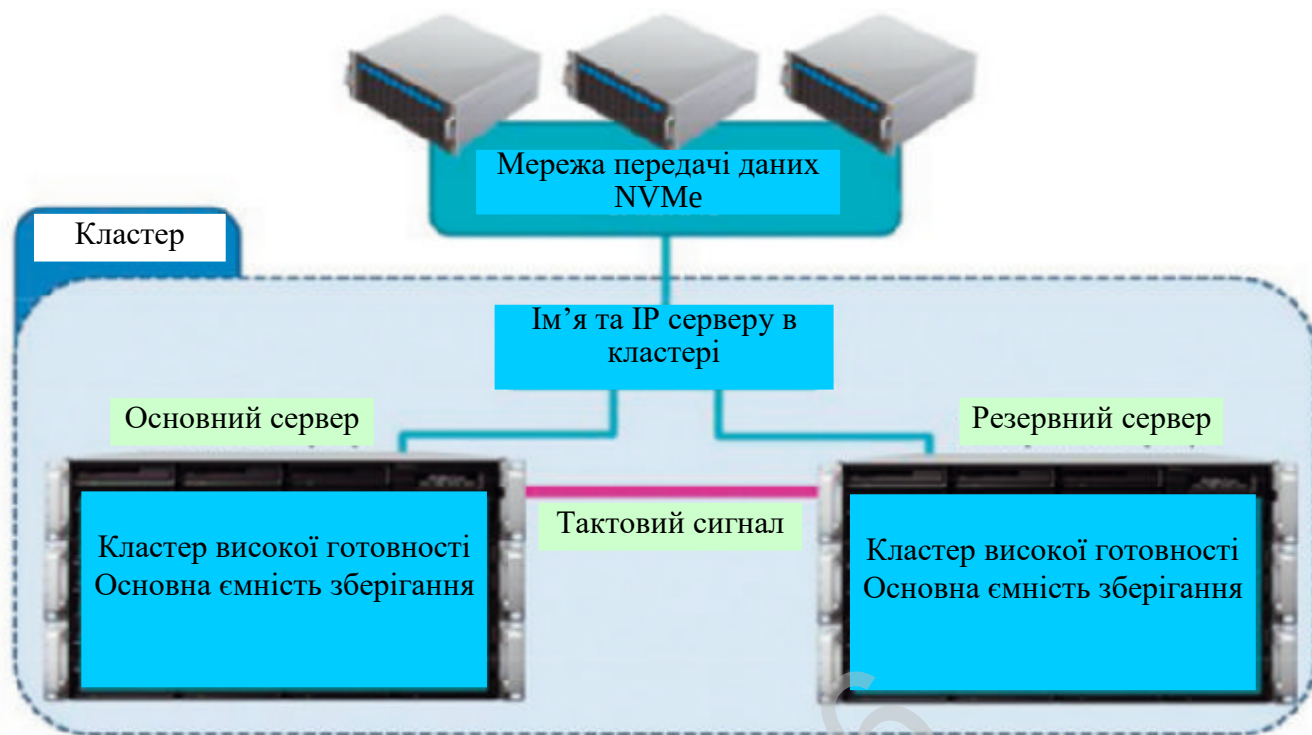


Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. З неї бачимо, що розроблена система підвищення стійкості до уражень інформації яка записана на дисках системи програмно-визначаємого сховища для NVMe складається з наступних основних функціональних блоків:

- сам носій інформації – система програмно-визначаємого сховища для NVMe;
- кодер Ріда-Соломона, який використовується, коли відбувається запис інформації на диск системи програмно-визначаємого сховища для NVMe, при цьому необхідно враховувати, що об'єм інформації, яка записується на диск системи програмно-визначаємого сховища для NVMe, повинна бути меншою, ніж об'єм диску системи програмно-визначаємого сховища для NVMe, в зв'язку, з тим, що коди Ріда-Соломона відносяться до кодів з надмірністю, за рахунок якої й відбувається кодування;

– декодер Ріда-Соломона, який використовується при читанні даних з відповідного диску системи програмно-визначаємого сховища для NVMe.

Функціонально блок, який утримує кодер Ріда-Соломона включає в себе наступні блоки:

- дані, які потрібно захистити;
- поліном для кодування;
- відмовостійки коди.

Коди Ріда-Соломона базуються на спеціальному розділі математики – полях Галуа (GF) або кінцевих полях. Арифметичні дії (+, -, x, / і т.д.) над елементами кінцевого поля дають результат, що також є елементом цього поля. Кодер та декодер Ріда-Соломона повинні вміти виконувати ці арифметичні операції. Ці операції для своєї реалізації вимагають спеціального устаткування або спеціалізованого програмного забезпечення.

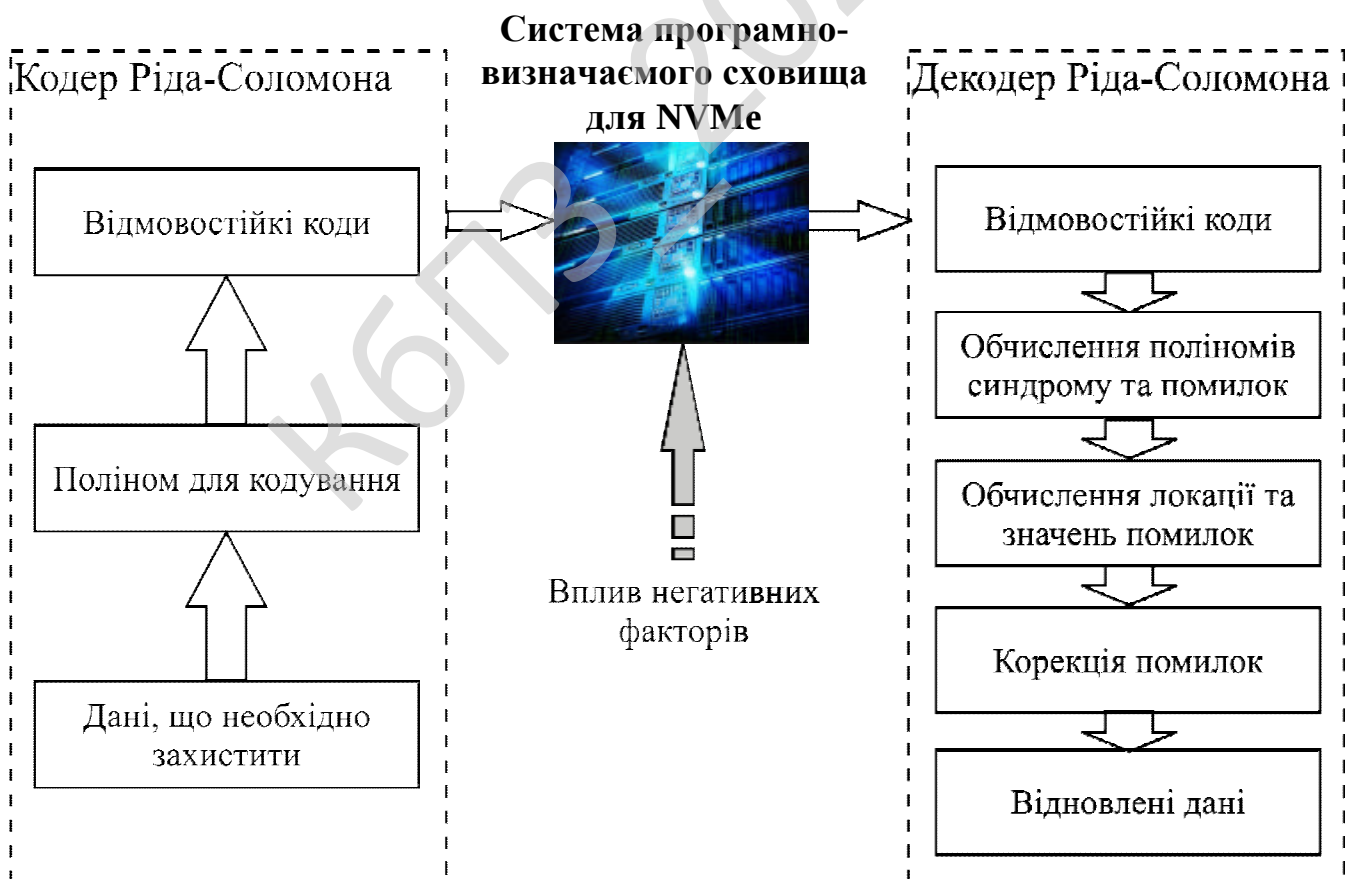


Рисунок 3.2 – Функціональна схема системи

Кодове слово Ріда-Соломона формується із залученням спеціального полінома. Всі коректні кодові слова повинні ділитися без залишку на ці утворюючі поліноми. Загальна форма утворюючого полінома має вигляд:

$$g(x) = (x - a^i)(x - a^{i+1}) \dots (x - a^{i+2t}),$$

а кодове слово формується за допомогою операції:

$$c(x) = g(x) \cdot i(x),$$

де $g(x)$ є утворюючим поліномом, $i(x)$ являє собою інформаційний блок, $c(x)$ – кодове слово, що називається простим елементом поля.

$2t$ символів парності в кодовому слові Ріда-Соломона визначаються з наступного співвідношення:

$$p(x) = i(x) \cdot x^{n-k} \bmod g(x)$$

Перейдемо до розгляду іншого функціонального блоку – декодеру Ріда-Соломона.

Цей функціональний блок містить у собі наступні блоки:

- відмовостійкий код;
- блок обчислення поліномів синдрому та помилок;
- блок обчислення локації та значень помилок;
- блок корекції помилок;
- відновлені за допомогою коді Ріда-Соломона дані.

Декодер працює наступним чином.

Введемо позначення:

- $r(x)$ – Отримане кодове слово.
- S_i – Синдроми.
- $L(x)$ – Поліном локації помилок.
- X_i – Положення помилок.
- Y_i – Значення помилок.
- $c(x)$ – Відновлене кодове слово.
- v – Число помилок.

Отримане кодове слово $r(x)$ являє собою вихідне (передане) кодове слово

$c(x)$ плюс помилки: $r(x) = c(x) + e(x)$.

Декодер Ріда-Соломона намагається визначити позицію й значення помилки для числа t помилок (або $2t$ втрат) і виправити помилки й втрати.

Обчислення синдрому

Обчислення синдрому схоже на обчислення парності. Кодове слово Ріда-Соломона має $2t$ **синдромів**, це залежить тільки від помилок (а не переданих кодових слів). Синдроми можуть бути обчислені шляхом підстановки $2t$ коріння утворюючого полінома $g(x)$ в $r(x)$.

Знаходження позицій символічних помилок

Це робиться шляхом рішення системи рівнянь із t невідомими. Існує кілька швидких алгоритмів для рішення цього завдання. Ці алгоритми використовують особливості структури матриці кодів Ріда-Соломона й сильно скорочують необхідну обчислювальну потужність. Робиться це у два етапи:

1. Визначення полінома локації помилок. Це може бути зроблене за допомогою алгоритму Berlekamp-Massey або алгоритму Евкліда. Алгоритм Евкліда використовується частіше на практиці, тому що його легше реалізувати, однак, алгоритм Berlekamp-Massey дозволяє одержати більш ефективну реалізацію встаткування й програм.

2. Знаходження кореня цього полінома. Це робиться із залученням алгоритму пошуку Chien.

Знаходження значень символічних помилок

Тут також потрібно вирішити систему рівнянь із t невідомими. Для рішення використовується швидкий алгоритм Forney.

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

Опис коду Ріда-Соломона

Коди Ріда-Соломона – недвійкові циклічні коди, що дозволяють виправляти помилки в блоках даних. Елементами кодового вектора є не біти, а групи біт (блоки). Дуже поширені коди Ріда-Соломона, що працюють із байтами

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		34

(октетами). У цей час широко використовується в системах відновлення даних з компакт-дисків, при створенні архівів з інформацією для відновлення у випадку ушкоджень, у завадостійкому кодуванні. Код Ріда-Соломона був винайдений в 1960 році співробітниками лабораторії Лінкольна Массачусетського технологічного інституту Ірвином Рідом і Густавом Соломоном. Ідея використання цього коду була представлена в статті «Polynomial Codes over Certain Finite Fields». Перше застосування код Ріда-Соломона одержав в 1982 році в серійному випуску компакт-дисків. Ефективний алгоритм декодування був запропонований в 1969 році Елвином Берлекемпом і Джеймсом Мессі.

Коди Ріда-Соломона є важливим частковим випадком БЧХ-коду, корінь полінома, що породжує, якого лежать у тім же полі, над яким і будується код ($m = 1$).

Коди Боуза-Чоудхури-Хоквінгхема (БЧХ коди) – у теорії кодування це широкий клас циклічних кодів, застосовуваних для захисту інформації від помилок. Відрізняється можливістю побудови коду із заздалегідь певними коригувальними властивостями, а саме, мінімальною кодовою відстанню.

Поліном, що породжує

Визначення поліномом, що породжує, циклічного (n,k) коду C називається такий ненульовий $g(x) = \sum_{i=0}^r g_i x^i$ поліном з C , ступінь якого найменша й коефіцієнт при старшому ступені $g_r = 1$.

Теорема 3.1

Якщо C – циклічний (n,k) код і $g(x)$ – його поліном, що породжує, тоді ступінь $g(x)$ дорівнює $r = n - k$ і кожне кодове слово може бути єдиним чином представлено у вигляді $c(x) = t(x)g(x)$, де ступінь $t(x)$ менше або дорівнює $k - 1$.

Теорема 3.2

$g(x)$ – поліном, що породжує, циклічного (n,k) коду є дільником двочлена $x^n - 1$

Наслідок: у такий спосіб як поліном, що породжує, можна вибирати будь-який поліном, дільник $x^n - 1$. Ступінь обраного полінома буде визначати кількість

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		35

перевірочних символів r , число інформаційних символів $k = n - r$.

Матриця, що породжує

Поліноми $g(x), xg(x), x^2g(x), \dots, x^{k-1}g(x)$ лінійно незалежні, інакше $m(x)g(x) = 0$ при ненульовому $m(x)$, що неможливо.

Значить кодові слова можна записувати, як і для лінійних кодів, наступним чином:

$$\bar{m}G = (m_0, m_1, \dots, m_{k-1}) \begin{bmatrix} g(x) \\ xg(x) \\ \dots \\ x^{k-1}g(x) \end{bmatrix} = m(x)g(x), \quad (3.1)$$

де G є матрицею, що породжує, $m(x)$ – інформаційним поліномом.

Матрицю G можна записати в символьній формі:

$$G = \begin{bmatrix} g_0 & g_1 & \dots & g_{r-1} & g_r & 0 & \dots & 0 \\ 0 & g_0 & \dots & g_{r-2} & g_{r-1} & g_r & \dots & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 0 & g_0 & g_1 & \dots & g_r \end{bmatrix}$$

Перевірочна матриця

Для кожного кодового слова циклічного коду справедливо $c(x) = 0 \pmod{g(x)}$. Тому перевірочну матрицю можна записати як $H = [1 \ x \ x^2 \ \dots \ x^{n-2} \ x^{n-1}] \pmod{g(x)}$.

Тоді:

$$\bar{c}H^T = \sum_{i=0}^{n-1} c_i x^i \pmod{g(x)}. \quad (3.2)$$

Нехай α – елемент поля $GF(q)$ порядку n . Якщо α – примітивний елемент, то його порядок дорівнює $q-1$, т.е. $\alpha^{q-1}=1$, $\alpha^i \neq 1$, $0 < i < q-1$.

Тоді нормований поліном $g(x)$ мінімального ступеня над полем $GF(q)$, коріннями якого є $d-1$ підряд, що йдуть ступенів, $\alpha^{l_0}, \alpha^{l_0+1}, \dots, \alpha^{l_0+d-1}$ елемента α , є поліномом, що породжує, коду над полем $GF(q)$ $g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1}) \dots (x - \alpha^{l_0+d-1})$; де l_0 – деяке ціле число (у тому числі 0

і 1), за допомогою якого іноді вдається спростити кодер. Звичайно покладається $l_0 = 1$. Ступінь багаточлена $g(x)$ дорівнює $d - 1$.

Довжина отриманого коду n , мінімальна відстань d (мінімальна відстань d лінійного коду є мінімальним із всіх відстаней Хеммінга всіх пар кодових слів). Код містить $r = d - 1 = \deg(g(x))$ перевірочний символ, де $\deg()$ позначає ступінь полінома; число інформаційних символів $k = n - r = n - d + 1$. У такий спосіб $d = n - k - 1$ і код Ріда-Соломона є роздільним кодом з максимальною відстанню (є оптимальним у змісті границі Синглтона).

Кодовий поліном $c(x)$ може бути отриманий з інформаційного полінома $m(x)$, $\deg m(x) \leq k - 1$, шляхом перемножування $m(x)$ і $g(x)$: $c(x) = m(x)g(x)$

Властивості

Код Ріда-Соломона над $GF(q^m)$, що виправляє t помилок, вимагає $2t$ перевірочних символів і з його допомогою виправляються довільні пакети довжиною t і менше. Відповідно до теореми про границю Рейгера, коди Ріда-Соломона є оптимальними з погляду співвідношення довжини пакета й можливості виправлення помилок – використовуючи $2t$ додаткових перевірочних символів виправляються t помилок (і менш).

Теорема (границя Рейгера). Кожний лінійний блоковий код, що виправляє всі пакети довжиною t і менш, повинен містити щонайменше $2t$ перевірочних символів.

Виправлення багаторазових помилок

Код Ріда-Соломона є одним з найбільш потужних кодів, що виправляють багаторазові пакети помилок. Застосовується в каналах, де пакети помилок можуть утворюватися настільки часто, що їх уже не можна виправляти за допомогою кодів, що виправляють одиночні помилки. $(q^m - 1, q^m - 1 - 2t)$ -код Ріда-Соломона над полем $GF(q^m)$ з кодовою відстанню $d = 2t + 1$ можна розглядати як $((q^m - 1)m, (q^m - 1 - 2t)m)$ -код над полем $GF(q)$, що може виправляти будь-яку комбінацію помилок, зосереджену в t або меншому числі блоків з m символів.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		37

Найбільше число блоків довжини m , які може торкнутися пакет довжини l_i , де $l_i \leq mt_i - (m - 1)$, не перевершує t_i , тому код, що може виправити t блоків помилок, завжди може виправити й будь-яку комбінацію з r пакетів загальної довжини l , якщо $l + (m - 1) \leq mt$.

Практична реалізація

Кодування за допомогою коду Ріда-Соломона може бути реалізовано двома способами: систематичним і несистематичним.

При несистематичному кодуванні інформаційне слово множиться на якийсь полином, що неприводиться, у полі Галуа. Отримане закодоване слово повністю відрізняється від вихідного й для добування інформаційного слова потрібно виконати операцію декодування й уже потім можна перевірити дані на зміст помилок. Таке кодування вимагає більші витрати ресурсів тільки на добування інформаційних даних, при цьому вони можуть бути без помилок.

При систематичному кодуванні до інформаційного блоку з k символів приписуються $2t$ перевірочних символів, при обчисленні кожного перевірочного символу використовуються всі k символів вихідного блоку.

У цьому випадку немає витрат ресурсів при добуванні вихідного блоку, якщо інформаційне слово не містить помилок, але кодер/декодер повинен виконати $k(n - k)$ операцій додавання й множення для генерації перевірочних символів. Крім того, тому що всі операції проводяться в полі Галуа, те самі операції кодування/декодування вимагають багато ресурсів і часу. Швидкий алгоритм декодування, заснований на швидкому перетворенні Фур'є, виконується за час порядку $\ln n^2$.

Кодування

При операції кодування інформаційний поліном множиться на багаточлен, що породжує. Множення вихідного слова S довжини k на не приводиться полином, що, при систематичному кодуванні можна виконати в такий спосіб:

- До вихідного слова приписуються $2t$ нулів, виходить поліном $T = Sx^{2t}$.
- Цей поліном ділиться на поліном, що породжує, G , перебуває залишок

$R, Sx^{2t} = QG + R$, де Q – частка.

– Цей залишок й буде коригувальним кодом Ріда-Соломона, він приписується до вихідного блоку символів. Отримане кодове слово $C = Sx^{2t} + R$.

Кодер будується зі регістрів зсуву, суматорів і перемножувачів. Регістр зсуву складається з комірок пам'яті, у кожній з яких перебуває один елемент поля Галуа.

Наведений як приклад кодер Ріда-Соломона генерує 16 коригувальних байт, що дозволяє виправляти до 8 і виявляти до 16 помилок у кадрі даних.

Перемножувачі на константи $GF(0)...GF(15)$ у полі Галуа реалізуються в такий спосіб: спочатку вихідне число й константа перетворюються в індексну форму, потім складаються в межах байта без обліку переносу.

Результатом операції є результат додавання, перетворена обернено в поліноміальну форму.

При переході від однієї форми подання даних до іншої доцільно використовувати таблицю істинності розміром 256 байт, що становить ємність одного ЕАВ (Embedded Array Block – блок зосередженої пам'яті). Для реалізації кодера потрібно 16 таких перемножувачів, при цьому те саме число множиться на різні константи, що дозволяє використовувати для його перекладу в індексну форму один ЕАВ.

Для перекладу результатів у поліноміальну форму потрібно вже 16 таких таблиць, що вимагає застосування ІМС FPGA дуже великої ємності. У запропонованій схемі використовується тактування кодера із частотою, в 8 разів перевищуючу частоту надходження байт даних.

Це дає можливість використовувати дві пари «суматор – ЕАВ», мультиплексуючи константи на входах суматорів і дозволяючи роботу регістрів-накопичувачів у моменти появи відповідних даних на виходах засувки ЕАВ.

На структурній схемі кодера (рисунок 3.2) символу «С» відповідають дві константи $GF(n)$.

Символ «L» у логіці регістрів-накопичувачів відповідає наступний: вихід

компаратора нуля (символи CMP0 і SYNC) дозволяє роботу схеми «АБО що виключає », на входи якої подаються вихід попереднього регістра й ЕАВ. Якщо ж вектор зворотного зв'язку дорівнює "0", схема пропускає дані з виходу попереднього регістра-накопичувача на вхід наступного.

У результаті кодер з урахуванням схеми синхронізації (на рисунку не показана) займає 255 LE (Logic Element – логічний елемент) і 3 ЕАВ, що дозволяє розмістити його в ІМС EPF10K10. Після оптимізації розміщення схеми на кристалі FPGA швидкодія схеми досяглася 11,57 МГц (частота надходження байт даних, далі – байтова частота).

При використанні ІМС EPF10K20, у складі якої 6 ЕАВ, використовуючи 4 пари "суматор – ЕАВ", можна тактувати кодер із частотами, що перевищують байтову частоту не в 8, а в 4 рази, що дозволить підняти її до 25...30 МГц.

Декодування

Декодер, що працює по авторегресивному спектральному методі декодування, послідовно виконує наступні дії:

- Обчислює синдром помилки.
- Будує поліном помилки.
- Знаходить корінь даного полінома.
- Визначає характер помилки.
- Виправляє помилки.

Обчислення синдрому помилки

Обчислення синдрому помилки виконується синдромним декодером, що ділить кодове слово на багаточлен, що породжує. Якщо при діленні виникає остача, то в слові є помилка. Остача від ділення є синдромом помилки.

Побудова полінома помилки

Обчислений синдром помилки не вказує на положення помилок. Ступінь полінома синдрому дорівнює $2t$, що багато менше ступеня кодового слова n . Для одержання відповідності між помилкою і її положенням у повідомленні будується поліном помилок.

Поліном помилок реалізується за допомогою алгоритму Берлекемпа-Мессі, або за допомогою алгоритму Евкліда. Алгоритм Евкліда має просту реалізацію, але вимагає більших витрат ресурсів. Тому частіше застосовується більше складний, але менш затратоємний алгоритм Берлекемпа-Мессі. Коефіцієнти знайденого полінома безпосередньо відповідають коефіцієнтам помилкових символів у кодовому слові.

Алгоритм Евкліда виконується наступним чином.

Нехай a і b суть цілі числа, не рівні одночасно нулю, і послідовність чисел $a, b, r_1 > r_2 > r_3 > r_4 > \dots > r_n$ визначена тим, що кожне r_k це остача від ділення перед-попереднього числа на попереднє, а передостаннє ділиться на останнє націло, тобто

$$\begin{aligned} a &= bq_0 + r_1 \\ b &= r_1q_1 + r_2 \\ r_1 &= r_2q_2 + r_3 \\ &\dots \\ r_{n-1} &= r_nq_n \end{aligned} \tag{3.3}$$

Тоді (a,b) , найбільший загальний дільник a і b , дорівнює r_n , останньому ненульовому члену цієї послідовності.

Існування таких $r_1, r_2, \dots$, тобто можливість ділення з остачею m на n для будь-якого цілого m і цілого $n \neq 0$, доводиться індукцією по m .

Коректність цього алгоритму впливає з наступних двох тверджень:

- Нехай $a = bq + r$, тоді $(a,b) = (b,r)$.
- $(0,r) = r$. для будь-якого ненульового r .

Знаходження корню

На цьому етапі шукаються коріння полінома помилки, що визначають положення перекручених символів у кодовому слові. Реалізується за допомогою процедури Ченя, рівносильній повному перебору. У поліном помилок послідовно підставляються всі можливі значення, коли поліном звертається в нуль – коріння знайдені.

Визначення характеру помилки і її виправлення

По синдрому помилки й знайдених корінь полінома за допомогою алгоритму Форні визначається характер помилки й будується маска перекручених символів. Ця маска накладається на кодове слово за допомогою операції XOR і перекручені символи відновлюються. Після цього відкидаються перевірочні символи й виходить відновлене інформаційне слово.

Застосування

У даний момент коди Ріда-Соломона мають дуже широку область застосування завдяки їхній здатності знаходити й виправляти багаторазові пакети помилок.

Запис і зберігання інформації

Код Ріда-Соломона використовується при записі й читанні в контролерах оперативної пам'яті, при архівуванні даних, запису інформації на жорсткі диски (ЕСС), запису на система програмно-визначаємого сховища для NVMe. Навіть якщо ушкоджено значний обсяг інформації, зіпсовано кілька секторів дискового носія, то коди Ріда-Соломона дозволяють відновити більшу частину загубленої інформації. Також використовується при записі на такі носії, як магнітні стрічки й штрихкоди.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходить 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.

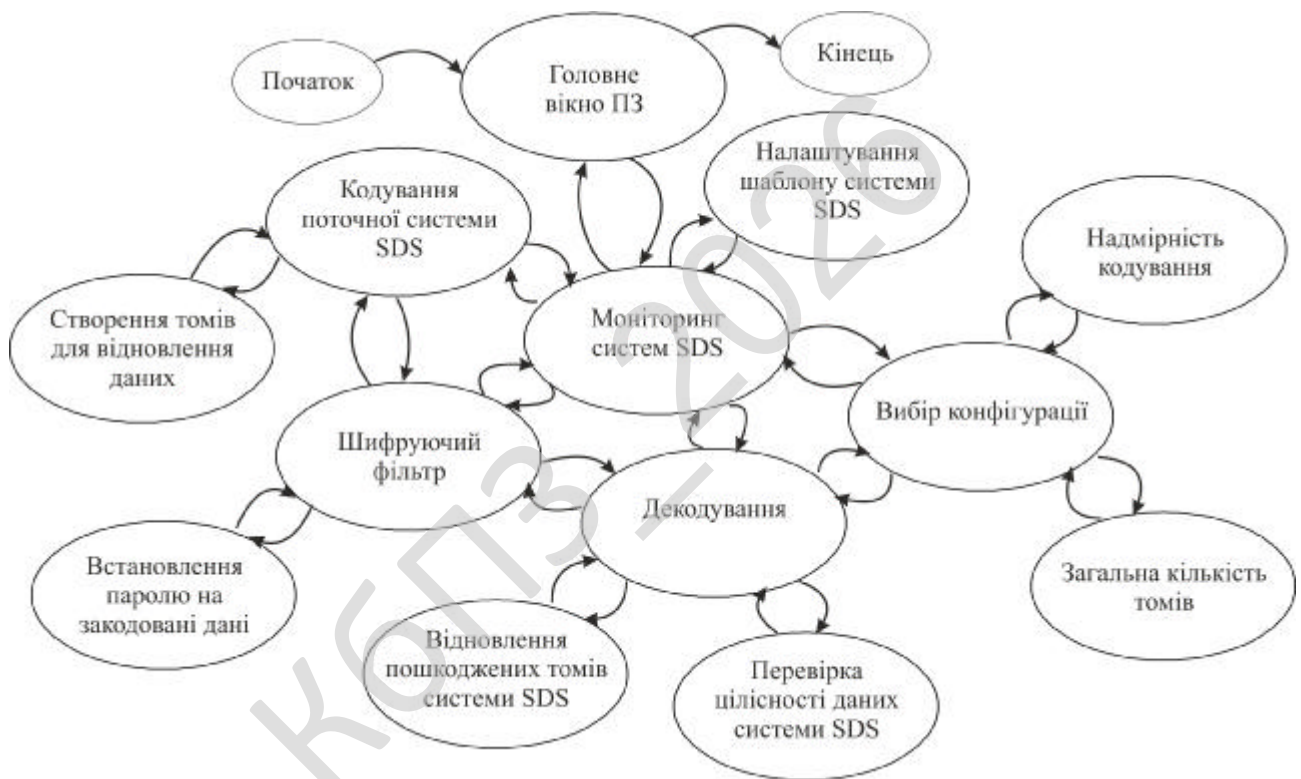


Рисунок 3.3 – Діаграма взаємодії процесів

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 ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

В першу чергу розглянемо архітектурний шаблон який використовувався у роботі – подійно-орієнтована архітектура (Event-driven architecture, надалі EDA) це шаблон архітектури програмного забезпечення, який призначений для створення подій, їх виявлення, споживання і реагування на них.

Подія може бути визначена як значна зміна стану. Наприклад, коли споживач купує автомобіль, стан автомобіля змінюється з "на продаж" до "продано". Архітектура системи дилера автомобілів може трактувати цю зміну стану як подію, поява якої може стати відомою іншим програмам даної архітектури.

З формальної точки зору, те, що виробляється, публікується, поширюється, виявляється і споживається (як правило, асинхронно) є повідомленням, яке називають сповіщенням про подію (або нотифікацією), а не самою подією, яка є зміною стану, що викликає появу повідомлення.

Події не подорожують, вони просто відбуваються. Проте термін подія часто використовується метонімічно для позначення самого нотифікаційного повідомлення, що може призвести до певної плутанини.

Цей архітектурний шаблон може застосовуватися при проектуванні і реалізації ПЗ і систем, які передають події між слабкозв'язаними компонентами програмного забезпечення і сервісами (службами).

Подійно-орієнтована система як правило складається з емітерів подій (або агентів) і споживачів подій (або стоків).

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		44

Стоки несуть відповідальність за здійснення реагування на появу події. Реакція не завжди може бути повністю забезпечена самим стоком. Наприклад, стік, може бути відповідальним лише за фільтрацію, трансформацію і відправку події до іншого компонента або він може забезпечити повністю самостійну реакцію на таку подію. Перша категорія стоків може бути заснована на традиційних компонентах, таких як проміжне програмне забезпечення, орієнтоване на обробку повідомлень (message oriented middleware, MOM), в той час, як друга категорія стоків (самостійна реакція в режимі он-лайн) може вимагати більш придатної платформи (фреймворку) для виконання транзакцій.

Розробка ПЗ і систем в подійно-орієнтованій архітектурі дозволяє їм бути сконструйованими способом, який більш відповідає вимогам до їх створення, оскільки такі системи в більшій мірі пристосовуються до непередбачуваних і асинхронних середовищ.

Подійно-орієнтована архітектура (EDA) може доповнювати сервісно-орієнтовану архітектуру (SOA), оскільки сервіси (служби) можуть бути активовані тригерами, які ініціюються при настанні подій.

Ця парадигма особливо корисна, коли стік не забезпечує власного виконання будь-яких дій.

Подійно-орієнтована SOA (SOA-2) розвиває архітектури SOA і EDA для забезпечення більш глибокого і надійного рівня сервісів за рахунок використання раніше невідомих причинно-наслідкових зв'язків, щоб сформувати новий шаблон подій. Цей новий шаблон бізнес-аналітики дає поштовх до небаченого раніше зростання рівня автоматизації підприємства за рахунок привнесення додаткової цінної інформації в описану раніше модель діяльності.

Обчислювальна техніка та сенсорні пристрої (сенсори, датчики, контролери) можуть виявляти зміни стану об'єктів або умов і створювати події, які потім можуть бути оброблені сервісом (службою) або системою.

Подія може складатися з двох частин: заголовка події та тіла події. Заголовок події може включати в себе інформацію таку як, наприклад, назва події,

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		45

часова мітка події і тип події. Тіло події – це частина, яка описує факт, що стався в дійсності. Тіло події не слід плутати з шаблоном або логікою, яка може бути застосована як реакція на саму подію.

Архітектура, керована подіями, складається з чотирьох логічних рівнів (шарів). Вона починається з виявлення факту, його технічного подання у формі події і закінчується непустою множиною реакцій на цю подію.

Генератор подій

Першим логічним шаром є генератор подій, який виявляє факт і представляє цей факт подією. Оскільки фактом може бути практично все, що може бути сприйнято, то ним може бути і генератор подій. Наприклад, генератором може бути клієнт електронної пошти, система електронної комерції або певний тип датчика.

Перетворення різних даних, отриманих від датчиків, в єдину стандартизовану форму даних, які можуть бути оцінені, є основною проблемою при розробці та реалізації цього шару. Однак, враховуючи, що подія є строго декларативною, можна легко застосовувати будь-які операції трансформації, тим самим усуваючи необхідність забезпечення високого рівня стандартизації.

Канал подій

Канал подій – це механізм, через який інформація від генератора подій передається до обробника подій (event engine) або стоку.

Це може бути з'єднання TCP/IP або вхідний файл будь-якого типу (простий текст, формат XML, e-mail тощо). В один і той же час може бути відкрито кілька каналів подій. Як правило, оскільки обробник подій повинен працювати в режимі, наближеному до реального часу, канали подій зчитуються асинхронно. Події зберігаються в черзі, очікуючи наступної обробки механізмом обробки подій.

Механізм обробки подій

Механізм обробки подій (event processing engine) є місцем, де подія ідентифікується і вибирається відповідна реакція на нього, яка потім виконується. Це також може призвести до породження ряду тверджень. Якщо подія, яка

надійшла до механізму обробки подій, є наприклад такою «Запаси продукту ID досягли нижнього допустимого рівня», це може ініціювати, наприклад, такі реакції як «Замовити продукт ID» і «Сповістити персонал».

Наступна подійно-орієнтована дія (післядія)

Щодо того, як можуть проявлятися наслідки події, слід відмітити, що вони можуть проявитись багатьма різними способами і у різноманітних формах (наприклад, повідомлення електронної пошти, надіслане комусь, або ПЗ, що виводить деяке попередження на екран). Залежно від рівня автоматизації, який забезпечується стоком (механізмом обробки подій), ці дії можуть виявитись зайвими.

Є три основні стилі обробки подій: простий, потоковий і складний. Часто ці три стилі використовуються спільно у розвинутій подійно-орієнтованій архітектурі.

Проста обробка подій

Проста обробка подій стосується подій, які безпосередньо належать до специфічних вимірних змін умов. У випадку простої обробки подій, мають справу з появою відомих подій, що ініціюють післядію (післядії). Проста обробка подій зазвичай використовується для управління потоком робіт в реальному часі, скорочуючи тим самим час затримки і вартість робіт.

Наприклад, прості події можуть створюватись (породжуватись) датчиком, що виявляє зміну тиску в шині або температуру навколишнього середовища.

Обробка потоку подій

При обробці потоку подій (event stream processing, далі ESP) відбуваються як звичайні, так і відомі події. Звичайні події (заявки, передачі RFID) перевіряються на те, чи є вони відомими, і передаються інформаційним передплатникам. Обробка потоку подій зазвичай використовується для управління потоком інформації в реальному часі і на рівні підприємства, що дозволяє своєчасно приймати рішення.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		47

Обробка складних подій

Обробка складних подій (Complex event processing (CEP)) дозволяє за шаблонами простих і звичайних подій проводити аналіз того, чи напустила складна подія. Обробка складних подій полягає в оцінюванні взаємного впливу подій і в наступному виконанні дій. При цьому, типи подій (відомих або звичайних) можуть перетинатись, а події можуть виникати протягом тривалого періоду часу.

Кореляція подій може бути причинною, тимчасовою або просторовою. CEP вимагає використання складних інтерпретаторів подій, визначення і підбору шаблонів подій, а також відповідних кореляційних методів. Обробка складних подій зазвичай використовується для виявлення і реагування на аномальну поведінку, загрози і можливості у бізнесі.

Під час роботи над бакалаврською дипломною 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.

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.

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 елемента. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елемента. 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.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

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 програмно-визначаємого сховища для NVMe.

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.

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

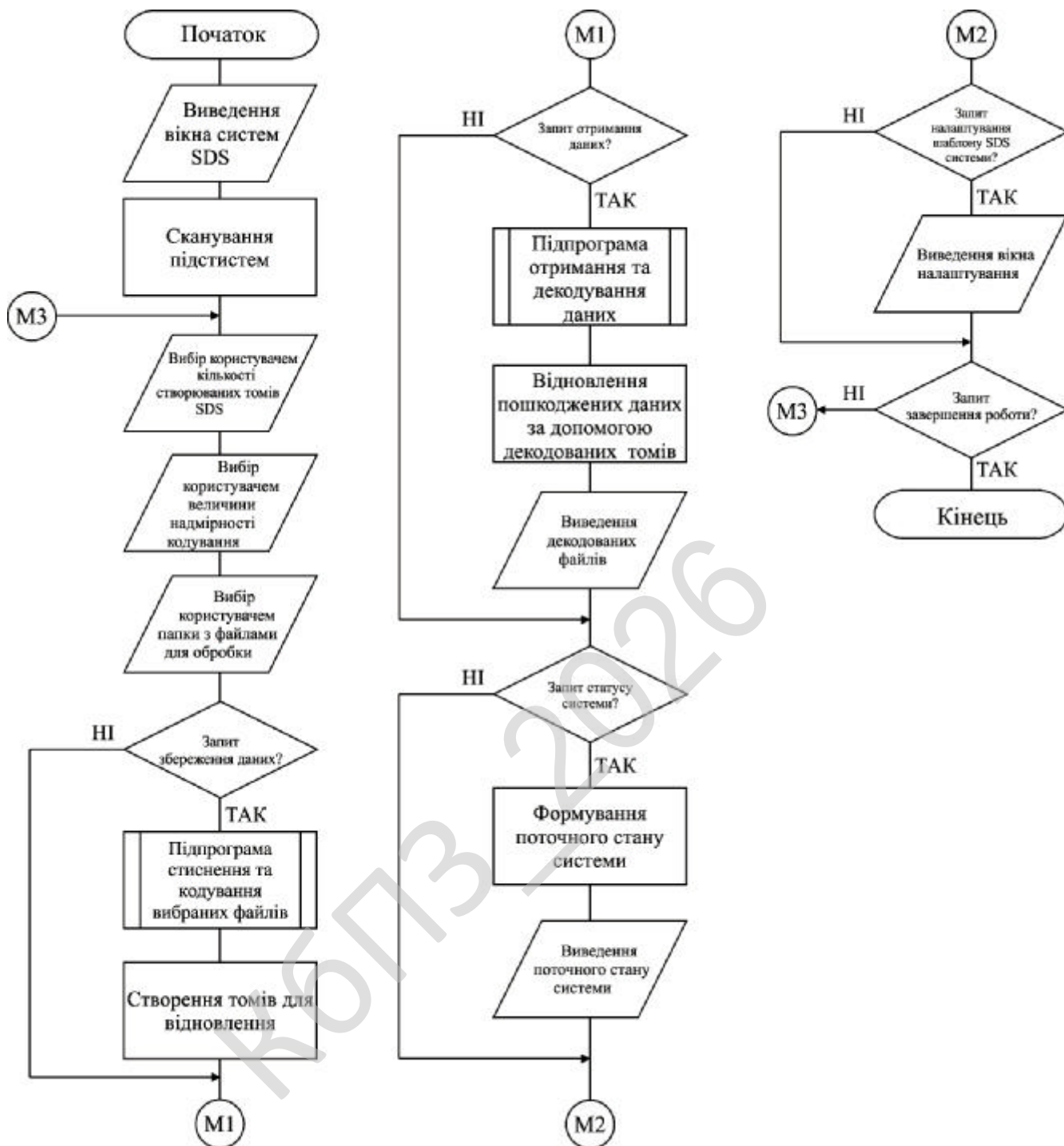


Рисунок 4.1 – Блок-схема основної програми

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності.

Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

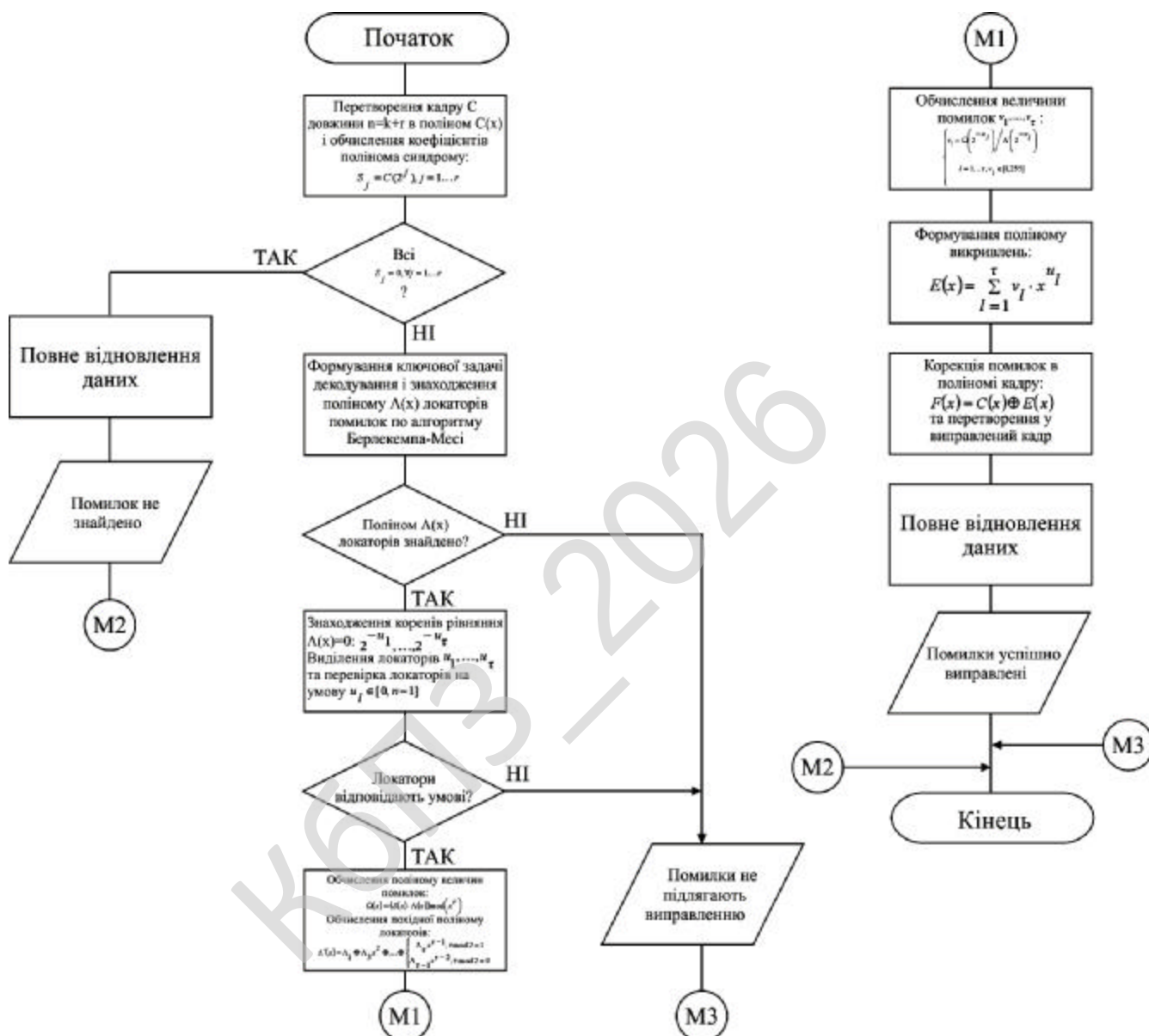


Рисунок 4.2 – Блок-схема роботи підпрограми

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проєктована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

- асоціації (association relationship);
- включення (include relationship);
- розширення (extend relationship);
- узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		54

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилалися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		55

з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Blowfish, який є симетричним алгоритмом шифрування, тобто таким, у якому ключ шифрування дорівнює ключу дешифрування. Він є мережею Фейштеля, у якій кількість ітерацій дорівнює 16. Довжина блоку дорівнює 64 бітам, ключ може мати будь-яку довжину в межах 448 біт. Хоча перед початком будь-якого шифрування виконується складна фаза ініціалізації, саме шифрування даних виконується досить швидко.

Алгоритм призначений в основному для додатків, у яких ключ міняється нечасто, до того ж існує фаза початкового рукостискання, під час якої відбувається автентифікація сторін і узгодження загальних параметрів і секретів. При реалізації на 32-бітних мікропроцесорах з більшим кешем даних Blowfish значно швидше DES.

Алгоритм складається із двох частин: розширення ключа й шифрування даних. Розширення ключа перетворює ключ довжиною, принаймні, 448 біт у кілька масивів підключів загальною довжиною 4168 байт.

В основі алгоритму лежить мережа Фейштеля з 16 ітераціями. Кожна ітерація складається з перестановки, що залежить від ключа, і підстановки, що залежить від ключа й даних. Операціями є XOR і додавання 32-бітних слів.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		56

Blowfish використовує велику кількість підключів. Ці ключі повинні бути обчислені заздалегідь, до початку будь-якого шифрування або дешифрування даних. Елементи алгоритму:

1. P – масив, що складається з вісімнадцяти 32-бітних підключів:

$$P_1, P_2, \dots, P_{18}.$$

2. Чотири 32-бітних S-boxes с 256 входами кожний. Перший індекс означає номер S-box, другий індекс – номер входу.

$$S_{1,0}, S_{1,1}, \dots S_{1,255};$$

$$S_{2,0}, S_{2,1}, \dots S_{2,255};$$

$$S_{3,0}, S_{3,1}, \dots S_{3,255};$$

$$S_{4,0}, S_{4,1}, \dots S_{4,255};$$

Шифрування

Входом є 64-бітний елемент даних X , що ділиться на дві 32-бітні половини, X_l і X_r .

$$X_l = X_l \text{ XOR } P_i$$

$$X_r = F(X_l) \text{ XOR } X_r$$

$$\text{Swap } X_l \text{ and } X_r$$

Функція F

Розділити X_l на чотири 8-бітних елементи A, B, C, D .

$$F(X_l) = ((S_{1,A} + S_{2,B} \bmod 2^{32}) \text{ XOR } S_{3,C}) + S_{4,D} \bmod 2^{32}$$

Дешифрування відрізняється від шифрування тим, що P_i використовуються у зворотному порядку.

Генерація підключів

Підключи обчислюються з використанням самого алгоритму Blowfish.

1. Ініціалізувати перший P -масив і чотири S-boxes фіксовані рядки.

2. Виконати операцію XOR P_1 з першими 32 бітами ключа, операцію XOR P_2 із другими 32 бітами ключа й т.д. Повторювати цикл доти, поки весь P -масив не буде побітово складний з усіма бітами ключа. Для коротких ключів виконується конкатенація ключа із самим собою.

3. Зашифрувати нульовий рядок алгоритмом Blowfish, використовуючи підключи, описані в пунктах (1) і (2).

4. Замінити P_1 і P_2 виходом, отриманим на кроці (3).

5. Зашифрувати вихід кроку (3), використовуючи алгоритм Blowfish з модифікованими підключами.

6. Замінити P_3 і P_4 виходом, отриманим на кроці (5).

7. Продовжити процес, замінюючи всі елементи P -масиву, а потім всі чотири S -boxes, виходами відповідним чином модифікованого алгоритму Blowfish.

Для створення всіх підключів потрібна 521 ітерація.

К6ПЗ-2026

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		58

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ системи програмно-визначаємого сховища для NVMe яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні блоки:

– Блок Пули та томи зберігання відображає основні об’єкти сховища. У ньому показано NVMe пули та логічні томи, їхній стан, загальну ємність, використаний обсяг, вільний простір, тип RAID або спосіб реплікації. Індикатор заповнення дає змогу швидко оцінити рівень використання ресурсу зберігання.

– Блок Стан інфраструктури показує ключові показники роботи системи. До них належать кількість активних вузлів, продуктивність у IOPS, середня затримка, пропускна здатність та загальне використання ємності. Ці показники використовуються для оперативного контролю продуктивності NVMe сховища.

– Блок NVMe вузли кластера містить таблицю з переліком вузлів системи. У таблиці відображено назву вузла, його роль, IP адресу та поточний стан. Вузли можуть виконувати роль контролера, вузла зберігання або шлюзу NVMe over Fabrics. Такий блок дає змогу контролювати доступність кожного компонента кластера.

– Блок Керування системою містить основні дії адміністратора. Кнопка Створити том призначена для додавання нового логічного тому. Кнопка Підключити вузол використовується для введення нового NVMe вузла до складу кластера. Кнопка Перевірити стан запускає перевірку доступності вузлів, стану пулів і працездатності сховища. Кнопка Оптимізувати балансування виконує перерозподіл навантаження між вузлами для підвищення стабільності та продуктивності системи.

– Поле Поточна політика розміщення показує активний режим розподілу даних між вузлами. У прикладі використовується автоматичне балансування, яке

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		59

дає змогу рівномірно розміщувати дані з урахуванням завантаження та доступної ємності. Поле Маршрут доступу показує технологію доступу до сховища. У прикладі використовується NVMe over Fabrics через TCP.

– Блок Журнал подій фіксує основні операції системи. У ньому відображаються запуск кластера, реєстрація вузлів, створення томів, попередження про деградований стан, помилки доступності та завершення перебалансування. Журнал подій використовується для контролю роботи системи, діагностики несправностей і подальшого аналізу дій адміністратора.

– Навігаційне меню: Файл; Інструменти; Параметри; Довідка.

– Розділу обрання групи файлів дисків системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

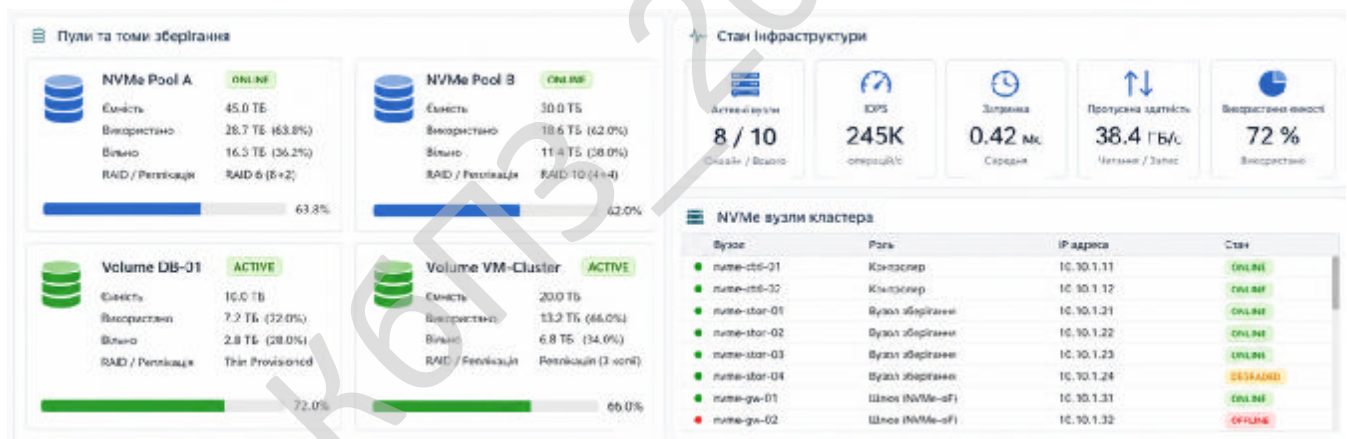


Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям,

які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

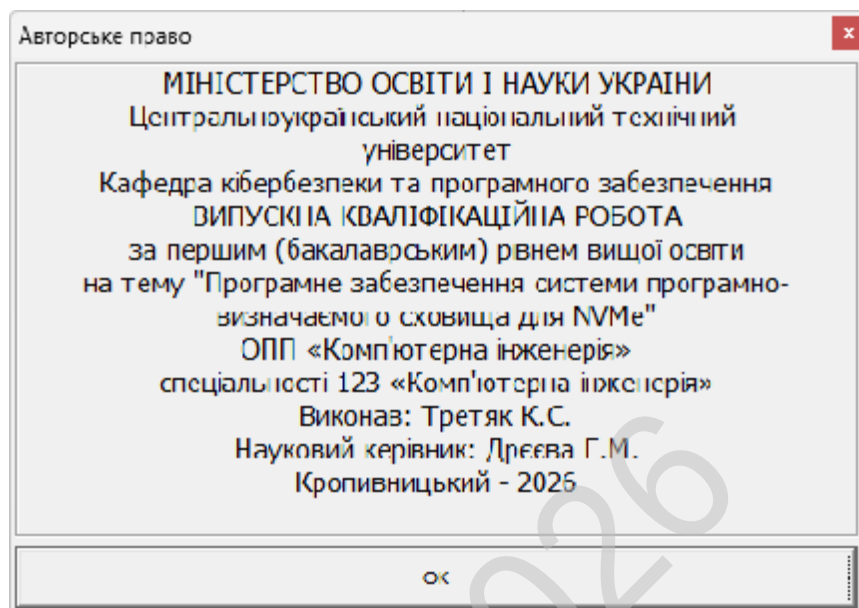


Рисунок 5.2 – Авторське право

Тенденція абстрагування ПЗ від нижчележачого устаткування дає шанс українським розроблювачам запропонувати передові рішення й вийти на перспективний ринок, не маючи ресурси західних компаній. І вони намагаються цей шанс використовувати: цілий ряд вітчизняних компаній займаються розробками в області програмно обумовлених мереж і систем зберігання.

П'ять-десять років тому все остерігалися використовувати програмно-визначаємі сховища для NVMe, але зараз вони не уступають апаратним платформам ні по функціональності, ні по надійності. Як і інше програмно-визначаємі сховища для NVMe, рішення може бути встановлене на будь-яке устаткування x86, а компоненти можна використовувати стандартні, доступні на ринку: сервери x86 і диски оригінальних виробників.

Оскільки замовники усе ще консервативні й поки не готові відмовлятися від переваг, які дає покупка готового рішення в порівнянні із самостійною

установкою програмного забезпечення на сервери, продукт поставляється й у вигляді апаратного комплексу. При виникненні яких-небудь збоїв замовники не хочуть з'ясовувати, хто винуватий – постачальник ПЗ або устаткування, а купуючи апаратно-програмний комплекс, вони одержують протестироване рішення, за надійну роботу якого повністю відповідає постачальник.

Розроблена програма має дуже простий і інтуїтивно зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

- Випуск.
- Встановлення та активація.
- Деактивація.

- Адаптація.
- Обновлення.
- Вмонтування.
- Відстежування версій.
- Видалення.
- Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати ІТ рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене ІТ рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження.

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		63

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 чорної скриньки.

Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

- Як виконуються функції програми.
- Як приймаються вихідні дані.
- Як виробляються результати.
- Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^{10} . Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		64

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чію поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

- Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ІТс).

- Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його ОТ).

Будь-який спосіб тестування «чорної скриньки» повинен:

- Виявити такі вхідні дані, які з високою ймовірністю належать набору ІТс;
- Сформулювати такі очікувані результати, які з високою ймовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

- Некоректних чи відсутніх функцій.
- Помилки інтерфейсу.
- Помилки у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

- Помилки характеристик (необхідна ємність пам'яті і т.д.).

- Помилки ініціалізації та завершення.

Обрано умови розповсюдження – Freeware.

Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		65

Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільноті як вільне програмне забезпечення.

Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

КБПЗ-2026

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		66

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи програмно-визначаємого сховища для NVMe.

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

Рішення завдання полягало у вирішенні наступних задач:

- Був проведений огляд існуючих систем програмно-визначаємого сховища для NVMe.
- Досліджена система програмно-визначаємого сховища для NVMe.
- На основі отриманих результатів досліджень створена програмна реалізація системи програмно-визначаємого сховища для NVMe.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання програмно-визначаємого сховища для NVMe.

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

Програма реалізована на мові високого рівня Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи програмно-визначаємого сховища для NVMe. 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

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		67

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 Blowfish.

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.

К6ПЗ-2026

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		68

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Е. Таненбаум, Д. Уезеролл «Комп'ютерні мережі». – [5-е вид.]. – 2016. – 960 с.
2. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.
3. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.
4. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.
5. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.
6. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.
7. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.
8. 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.
9. 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.
10. Смірнова Т.В., Коноплицька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		69

технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка*. 2024. №4(24), С. 6-27.

11. 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.

12. 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.

13. 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.

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

15. 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.

16. 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.

17. 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.

18. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		70

телекомунікаційних систем». *Проблеми інформатизації та управління*, № 2(70). 2022. С. 28-37.

19. 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.

20. 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.

21. 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.

22. 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.

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

24. 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.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		71

25. 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.
26. 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.
27. 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.
28. 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.
29. 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.
30. Smirnov O. Kuznetsov A., Zaichenko 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.
31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. Smirnov A.A., Kuznetsov 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.

40. 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

41. 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.

42. 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.

43. Смірнов О.А., Коноплицька-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.

44. 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.

45. Смірнов О.А., Дреєва Г.М., Дреєв 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.

46. Смірнов О.А., Коноплицька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		74

систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

47. 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).

48. Смірнов О.А., Дреєва 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

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

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

51. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Коноплицька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

52. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

					ВКРБ-123.26.0027.00.00.ПЗ	Арк.
Вим.	Арк.	№ докум.	Підпис	Дата		75

53. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

54. Смірнов О.А., Смірнов С.А., Дідик А.К., Дреєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". - Випуск 5 (142). - Х.: ХУПС - 2016. - С. 148-152.

55. Смірнов О.А., Смірнов С.А. Дідик А.К., Дреєв О.М. Моделі системи нейромережових експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.

56. Смірнов О.А., Смірнов С.А., Дідик А.К., Дреєв А.М. Спосіб контролю ліній зв'язку телекомунікаційної системи антивірусу. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. - 2016. - С. 121-127.

57. Смірнов О.А., Смірнов С.А., Дідик А.К. Метод безпечної маршрутизації метаданих у хмарні антивірусні системи. Системи озброєння та військова техніка. - Випуск 2 (46) - Х.: ХУПС - 2016. - С. 146-149.

58. Смірнов О.А., Кавун С.В., Доренський О.П., Вялкова В.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 151 с.

59. Смірнов О.А., Кавун С.В., Коваленко О.В., Дреєв О.М. Мережні інформаційні технології. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 159 с.