Центральноукраїнський національний технічний університет
Центр заочної та дистанційної освіти
Кафедра кібербезпеки та програмного забезпечення

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи Software Defined Storage”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-24Мз
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Молчанов Ю.В.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Минайленко Р.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Молчанов Ю.В. Дослідження та програмна реалізація системи Software Defined Storage. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи Software Defined Storage.

Метою розробки є дослідження та програмна реалізація системи Software Defined Storage.

Об'єктом дослідження є процес Software Defined Storage.

Предметом дослідження є методи Software Defined Storage.

Методи дослідження базуються на методах теорії побудови комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи Software Defined Storage.

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, Software Defined Storage

ABSTRACT

Molchanov Y.V. Research and software implementation of the Software Defined Storage system. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software designed for the Software Defined Storage system has been developed.

The purpose of the development is the research and software implementation of the Software Defined Storage system.

The object of the research is the Software Defined Storage process.

The subject of the research is the Software Defined Storage methods.

The research methods are based on the methods of the theory of computer network construction, methods of mathematical statistics, and methods of software development.

The result of the work is the software implementation of the Software Defined Storage system.

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 PCs with Windows 10/11.

The program was developed in Visual C#.

Keywords: computer engineering, Software Defined Storage

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АБГШ	–	аддітивний білий гаусів шум
БЧХ	–	код Боуза-Чоудхури-Хоквингема
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. Дані не чекають на бюджетні цикли. Вони зростають саме тоді, коли ваша система зберігання вирішує нагадати вам, хто насправді володіє вашим часом безвідмовної роботи. NAS та SAN все ще мають своє застосування, але їх масштабування схоже на плату за захист постачальникам обладнання – дороге, негнучке та розраховане на вчорашні робочі навантаження.

У 2025 році, коли конвеєри штучного інтелекту, аналітичні завдання та дані користувачів множаться швидше, ніж ваші сповіщення моніторингу, вам потрібне сховище, яке гнучко адаптується до потреб користувачів. Саме тут на допомогу приходять програмно-визначені сховища (Software Defined Storage, SDS): жодних оновлень навантажувачів, жодної прив'язки до постачальника, лише програмний рівень, який дозволяє масштабуватися в горизонтальному напрямку.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи Software Defined Storage.

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

– Огляд існуючих систем Software Defined Storage.

– Дослідження системи Software Defined Storage.

– Програмна реалізація системи Software Defined Storage.

Об’єктом дослідження є процес Software Defined Storage.

Предметом дослідження є методи Software Defined Storage.

Методи дослідження базуються на методах теорії побудови комп’ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод Software Defined Storage.

– Розроблено вітчизняний продукт Software Defined Storage, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі Software Defined Storage.

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

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

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи Software Defined Storage, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

SDS – це архітектура зберігання, що використовує рівень програмного забезпечення для виділення ресурсів і керування нею на серверах. Головний принцип таких систем – повний перенос організації зберігання з апаратних контролерів на програмний рівень. Програмно-визначаєме сховище, по суті, є програмним забезпеченням для зберігання організації зберігання даних, забезпечення безпеки на основі політик і керування зберіганням даних незалежно від апаратних засобів. Програмне забезпечення, що дозволяє створити середовище зберігання, може також забезпечувати керування для таких функцій, як дедуплікація даних, реплікація, моментальні знімки й резервне копіювання.

Головна причина появи SDS – бажання позбутися від залежності, пов'язаної з використанням устаткування одного вендору, гнучко нарощувати (або скорочувати) застосовувані ресурси, вирішувати нові функціональні завдання, а в підсумку домогтися скорочення експлуатаційних витрат при росту ефективності використання обчислювальних потужностей.

SDS-системи, що архітектурно повторюють класичні масиви, мають аналогічні технічні обмеження: фіксована кількість контролерів (нод), інтерфейсних портів взаємодії й накопичувачів (жорстких дисків, SSD).

Але перехід на програмно-визначаєму реалізацію функціонала уможливив використання технологій, які раніше були недоступні(наприклад, віртуалізація зовнішніх дискових масивів). Включення продуктів на основі SDS у набори рішень дозволило виробникам значно збільшити функціонала в цих сегментах.

Найбільші популярність і поширення на даний момент мають SDS-Продукти з об'єктним принципом зберігання даних. Їхні основні споживачі – середні й великі сервісні провайдери послуг по хмарному зберіганню: Dropbox,

Amazon, Google, Apple і ін. У випадку приватних хмар застосування об'єктного підходу гарантує уніфікацію обчислювальних компонентів і динамічне розширення інфраструктури за рахунок додавання універсальних нових нод (серверів), які можуть забезпечувати сервіс зберігання одночасно з іншими ролями.

На відміну від класичних рішень, які відповідають за гарантоване зберігання всього масиву даних (у рамках одного ресурсу), об'єктний принцип передбачає зберігання безлічі копій частин масиву даних – так званих об'єктів.

Схоронність даних забезпечується при доступності хоча б одного екземпляра з безлічі. За рахунок застосування об'єктного зберігання стала можлива організація резервування даних залежно від місця розташування елементів SDS. Наприклад, можна дублювати об'єкти на ресурсах різних вузлів (серверів – учасників SDS), розміщених у різних стійках ЦОД. Виходячи із цього, допускається втрата частини компонентів за умови збереження однієї або більше копій кожного з об'єктів.

Основні переваги:

– Економічність Програмно-визначаємі сховища дають можливість відмовитися як від пропрієтарних рішень на користь стандартизованого встаткування, так і від традиційної мережі зберігання даних (SAN) Fibre Channel на користь більше універсального стандарту Ethernet.

– Гнучкість. Програмно-визначаємі сховища можуть бути побудовані на встаткуванні будь-якого виробника або навіть різних виробників у рамках однієї системи. SDS легко масштабуються, оскільки програмний рівень організації надає дуже широкі можливості по керуванню ресурсами.

– Відказостійкість. При виникненні проблем на основний SDS за допомогою засобів реплікації дані легко переносяться резервну систему.

– Продуктивність. Програмно-визначаємі сховища підтримують підключення PCIe NVMe-карт, що істотно збільшує можливості систем.

SDS рішення:

– Ceph. Універсальна SDS, що дозволяє реалізувати блоковий, файловий і об'єктний доступ до даних у межах однієї системи. Завдяки підтримці асинхронної реплікації має порівняно більше низькі вимоги до швидкості передачі даних. Ще однією перевагою Ceph є відсутність обмежень на масштабування вузлів.

– GlusterFS. Серед програмно-визначаємих сховищ GlusterFS виділяється більше простим налаштуванням і підтримкою. Багаторівневий підхід до зберігання даних припускає використання вже випробуваних файлових систем на реальних дисках. GlusterFS може легко масштабуватися до петабайт дискового простору, що доступно користувачеві під однією точкою монтування. На відміну від інших розподілених файлових систем, GlusterFS не вимагає окремий сервер для зберігання метаданих.

– VMmware Virtual San. SDS від VMware убудована в лідируючий на корпоративному ринку гіпервізор vSphere, що полегшує первинне налаштування й наступне керування. Рішення пропонує відмінну продуктивність, можливість побудови «розтягнутих» кластерів серверів віртуалізації або розгортання у філіях усього на двох серверах.

– SANsymphony. Система від DataCore дозволяє віртуалізувати існуючі СЗД, збільшити їхню продуктивність за рахунок кешування в оперативній пам'яті або на твердотільних дисках, створювати розподілені програмні сховища на базі дисків, установлених локально в сервери.

1.2 Область застосування

Областю застосування є системи зберігання даних. Система зберігання даних це комплексне програмно-апаратне рішення по організації надійного зберігання інформаційних ресурсів і надання гарантованого доступу до них.

Класифікувати СЗД можна по різному

За типом організації зберігання даних

Існує кілька видів СЗД, от деякі з них.

Види СЗД:

– Direct-attached storage (DAS).

– Network-attached storage (NAS).

– Storage area network (SAN).

За способом зберігання даних

Block storage

Файли розбиваються на «шматочки» однакового розміру, кожний із власною адресою, але без метаданих. Зберігання на рівні блоків лежить в основі роботи традиційного жорсткого диска.Приклад – ситуація, коли драйвер HDD пише й зчитує блоки по адресах на відформатованому диску. Такі СЗД використовуються багатьма застосунками, наприклад, більшістю реляційних СУБД. У мережах доступ до блокових хостам організується за рахунок SAN за допомогою протоколів Fibre Channel і iSCSI.

Переваги:

– Блокові сховища мають набір інструментів, які забезпечують підвищену продуктивність: хост-адаптер шини розвантажує процесор і звільняє його ресурси для виконання інших завдань. Тому блокові системи зберігання часто використовуються для віртуалізації. Також добре підходять для роботи з базами даних.

Недоліки:

– Недоліками блокового сховища є висока вартість і складність у керуванні. Ще один мінус блокових сховищ (який ставиться й до файлових, про які далі) – обмежений обсяг метаданих. Будь-яку додаткову інформацію доводиться обробляти на рівні застосунків і баз даних.

File storage

Тут, дані зберігаються як файли й папки, зібрані в ієрархічну структуру, і доступні через клієнтські інтерфейси по ім'ю, назві каталогу й ін.

Переваги:

Серед плюсів файлових сховищ виділяють простоту. Файлу привласнюється ім'я, він одержує метадані, а потім «знаходить» собі місце в каталогах і підкаталогах. Файлові сховища звичайно дешевше в порівнянні із блоковими системами, а ієрархічна топологія зручна при обробці невеликих обсягів даних. Тому з їхньою допомогою організуються системи спільного використання файлів і системи локального архівування.

Недоліки:

Основний недолік файлового сховища – відсутність масштабованості. У міру нагромадження великої кількості даних – знаходити потрібну інформацію в купі папок і вкладень починає займати усе більше й більше часу. Із цієї причини файлові системи зберігання не використовуються в дата-центрах, де важлива швидкість.

Object storage

Відрізняються від файлових і блокових відсутністю файлової системи. Деревоподібну структуру файлового сховища тут заміняє плоский адресний простір. Ніякої ієрархії – просто об'єкти з унікальними ідентифікаторами, що дозволяють користувачеві або клієнтові витягати дані.

– Переваги:

Масштабованість, розширюваність метаданих, висока швидкість доступу до інформації.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи Software Defined Storage, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

У цей час на ринку представлена велика кількість продуктів, позиціонуємих виробниками як програмно-визначаємі системи зберігання (SDS – Software Defined Storage). Для огляду існуючих пропозицій доцільно ввести класифікацію, що буде містити критерії для порівняння представлених продуктів і дозволить, у першу чергу, визначити, ставиться продукт до SDS чи ні.

Почнемо з визначення ідеальної SDS. Програмно-визначаєма система зберігання – це програмне забезпечення, що дозволяє використовувати дискові ресурси стандартних обчислювальних вузлів (як правило, серверів архітектури x 86-x64), як це робиться зараз у традиційних системах зберігання з жорсткими дисками й SSD.

Під дане визначення, з деякими застереженнями, попадає велику кількість продуктів. Розділимо їх на групи по характерних ознаках.

Класичні SDS

До цієї групи ми відносимо продукти, які споконвічно розроблялися для об'єднання дискових ресурсів серверів у єдиний пул. Загальна ознака таких SDS – масштабована (scale-out) архітектура, що дозволяє нарощувати продуктивність і обсяг дискових ресурсів додаванням нових вузлів.

CEPH. Продукт споконвічно розроблявся як відкрита розподілена файлова система й відказостійке сховище даних. Використання протоколу TCP і серверів стандартної архітектури визначають низьку вартість зберігання. Scale-out архітектура й алгоритми розподілу даних по всіх вузлах кластера забезпечують високу продуктивність і відказостійкість. CEPH застосовується хостинг-

провайдерами, у високопродуктивних кластерах і у випадках коли потрібні масштабованість, понад Пбайт даний, надійність і продуктивність. Як багато продуктів, розроблювальні співтовариством, впровадження й супровід CEPH вимагає значних працезатрат висококваліфікованого персоналу й тому його не можна розглядати як заміну основної (general) системи зберігання в корпоративному сегменті. У той же час продукт може з успіхом використовуватися в середовищах з типізованим навантаженням і більшими обсягами даних.

Red Hat Storage Server. Розроблений одним з ведучих Linux-виробників на базі Red Hat Enterprise Linux, даний продукт позиціонується як рішення по зберіганню даних для приватних, публічних і гібридних хмарних середовищ, для зберігання й ефективного використання медіа-контенту, а також для високопродуктивних обчислень. Цей продукт також можна віднести до класичних SDS.

Не маючи функціонала необхідним для успішного застосування бізнес-застосунків, яким традиційно потрібен блоковий доступ, продукт забезпечує високу продуктивність у середовищах з більшою кількістю паралельних процесів уведення/виводу. Можливість гнучкого масштабування, простота заміни встаткування, що вийшов з ладу, а також підтримка виробника забезпечують надійну схоронність даних.

EMC ScaleIO. Має найбільш розвинені засоби розгортання й керування серед продуктів даної групи. EMC ScaleIO не тільки має високу продуктивність і масштабованістю, але може з успіхом замінити й універсальну СЗД середнього класу. Може бути використаний не тільки для рішення багатьох типових завдань, але й для таких бізнес-застосунків, як бази даних. Потрібно відзначити, що доступ до даних під керуванням EMC ScaleIO можна одержати тільки на рівні блокового пристрою, використовуючи спеціальний драйвер.

Таблиця 2.1 – Характеристики класичних SDS

Характеристики		CEPH	RedHat Storage	EMC ScaleIO
Платформа	Підтримувана архітектура	x86/x 86-64	x86/x 86-64	x86/x 86-64
	Підтримувані ОС	CentOS, Debian, Fedora, RHEL, Ubuntu	RedHat Linux	Microsoft Windows Red Hat Enterprise Linux CentOS Linux SUSE Linux
Вартість	Схема ліцензування	OpenSource, Комерційна підтримка	За вузол	За сиру ємність
Інтерфейс	Об'єктний доступ	S3, Swift	Так	Немає
	Блоковий доступ	iSCSI, власний клієнт	Немає	Власний клієнт
	Файловий доступ	Драйвер файлової системи для Linux. (kernel, FUSE)	Драйвер файлової системи для Linux. (kernel, FUSE)	Немає

Продовження таблиці 2.1

Характеристики		CEPH	RedHat Storage	EMC ScaleIO
Захист даних	Можливість побудови систем з довільною й гарантованою надмірністю	Так	Немає	Так
	Віддалена асинхронна реплікація	Немає	Так	Немає
	Самостійне відновлення	Так	Так	Так
	Миттєві знімки	Читання	Немає	Читання/запис
	Робота з «сирими» дисками	Так	Немає	Так
	Шифрування даних	Немає	Немає	Так
	Гарантоване видалення даних	Немає	Немає	Немає
	Захист від видалення даних. Режим WORM	Немає	Так	Немає

Продовження таблиці 2.1

Характеристики		CEPH	RedHat Storage	EMC ScaleIO
Функціонал	Інтеграція із традиційними СЗД	Немає	Немає	Немає
	Дедуплікація	Немає	Немає	Немає
	Стиск	Немає	Немає	Немає
	Flash-Cache	Так	Немає	Немає
	Багаторівневе зберігання (Tiering)	Немає	Немає	Так
	Гео-розподілені рішення	Немає	Немає	Немає

SDS на основі традиційних систем зберігання

SDS цієї групи створюються шляхом виділення й адаптації програмної складової традиційних СЗД для серверів стандартної архітектури без використання спеціального апаратного забезпечення. При цьому зберігається функціонал, властивій базовій системі зберігання, і забезпечується інтеграція із традиційними СЗД, на базі яких вона розроблена. Такий підхід дозволяє з максимальною ефективністю використовувати переваги SDS і функціонал існуючих СЗД.

NetApp Data ONTAP Edge. Даний продукт дозволяє створити центр обробки даних на базі одного сервера. Установлювана віртуальна машина на платформі VMware дозволяє використовувати внутрішні диски сервера як систему зберігання NetApp для віртуальних машин, розгорнутих на даному вузлі. Виробник рекомендує NetApp Data ONTAP Edge для передачі даних з віддалених офісів у центр і архівації їх у системі зберігання NetApp FAS. Таким чином, забезпечується можливість використовувати добре зарекомендували себе технології NetApp по реплікації й дзеркалюванню даних між SDS і традиційними

СЗД. Використання даного рішення обмежується тим, що неможливо об'єднати ресурси декількох серверів і працювати з обсягом даних понад 4 Тбайт.

HP StoreVirtual VSA. Це рішення від компанії HP – яскравий представник SDS на основі традиційних систем зберігання. HP StoreVirtual VSA для ОС HP Lefthand має найбагатшого функціонала й може використовуватися для створення програмно-визначаємих масштабованих СЗД. Варто відзначити такі можливості, як створення рівнів зберігання (tearing) і автоматичне переміщення даних між ними, захист даних і відсутність єдиних точок відмови завдяки технології Network RAID, роботу з усіма провідними гіпервізорами VMware vSphere, Microsoft Hyper-V і KVM. Інтеграція із системами розгортання ПЗ й хмарними сервісами, а також доступність широкого спектра послуг підтримки HP роблять даний продукт одним з найбільш універсальних. Він може застосовуватися як провайдерами послуг, так і замовниками з корпоративного сегмента.

Таблиця 2.2 – Характеристики SDS на основі традиційних систем зберігання

Характеристики		DataONTAP Edge	HP StoreVirtual VSA	HCP-VM (Virtual Machine)
Платформа	Підтримувані гіпервізори	VMware ESXi	ESXi, Hyper-V, KVM	VMware vSphere Hypervisor
Вартість	Схема ліцензування	За вузол	За вузол. Обмеження по обсязі на вузол.	За ємність
Продуктивність	Обмеження по масштабуванню ємності	10TB	50 ТБ/вузол, 32 вузла	40 вузлів, 4,7ПБ
	QoS	Немає	Так	

Продовження таблиці 2.2

Характеристики		DataONTAP Edge	HP StoreVirtual VSA	HCP-VM (Virtual Machine)
Захист Даних	Віддалена асинхронна реплікація	SnapMirror	Так	Так
	Миттєві знімки	Читання/сапис	Читання/сапис	Читання/сапис
	Шифрування даних	Немає	Немає	Так
	Гарантоване видалення даних	Немає	Немає	Так
	Захист від видалення даних. Режим WORM	Немає	Немає	Так
Функціонал	Дедуплікація	Так	Немає	Так
	Стиск	Так	Немає	Так
	Багаторівневе зберігання (Tiering)	Немає	Так	Так
	Гео-розподілені рішення	Так	Так	Так
Інтерфейс	Об'єктний доступ	Немає	Немає	S3, Swift
	Блоковий доступ	iSCSI	iSCSI	Немає
	Файловий доступ	CIFS, NFS	Немає	HTTP, SMB, NFS, WebDAV

SDS у складі обчислювальних комплексів

SDS даної групи дозволяють сполучити функцію віртуалізації обчислювальних ресурсів і віртуалізацію зберігання. Сервери, використовувані в якості хост-машин для віртуального середовища оснащуються жорсткими дисками, які поєднуються в SDS за допомогою спеціалізованого модуля гіпервізора.

Рішення групи «усе-в-одному» забезпечують гнучкість і простоту масштабування обчислювального комплексу, економію на виділеної СЗД. Обчислювальними ресурсами, необхідними для обробки запитів на зберігання, систему забезпечує хост-сервер.

Впровадження подібних рішень дозволяє замовникові трансформувати наявну інфраструктуру в Software Defined Datacenter, домігшись повної незалежності від апаратної конфігурації встаткування.

Таблиця 2.3 – Характеристики SDS у складі обчислювальних комплексів

Характеристики		Vmware vSAN	Nutanix
Платформа	Підтримувана архітектура	x86/x 86-64	Власна x86 " усе-в-одному" і сертифіковані сервера x86
	Підтримувані гіпервізори	VMware ESXi	ESXi, Hyper-V, KVM
Вартість	Можливість використання загальнодоступних компонентів	Так	Немає. Обов'язкова наявність комутаторів з низькими затримкам
	Схема ліцензування	за CPU на сервері. або для VDI за користувача	За вузол

Продовження таблиці 2.3

Характеристики		Vmware vSAN	Nutanix
Захист даних	Можливість побудови систем з довільною й гарантованою надмірністю	Так	Так
	Віддалена асинхронна реплікація	Немає	Так
	Миттєві знімки	Немає	Читання/сапис
	Шифрування даних	Немає	Так
	Гарантоване видалення даних	Немає	Так
Продуктивність	Автоматичне балансування навантаження між вузлами	Так	Міграція даних "у слід за навантаженням" у локальному кластері.
	Обмеження по масштабуванню ємності	Не обмежений обсяг	
	Лінійне масштабування продуктивності	Так	Так
	QoS	Засобами VMware	Так

Продовження таблиці 2.3

Характеристики		Vmware vSAN	Nutanix
Функціонал	Дедуплікація	Немає	Так
	Стиск	Так	Так
	Flash-Cache	Так, обов'язкова вимога – наявність.	Так
	Багаторівневе зберігання (Tiering)	Немає	Так
Інтерфейс	Об'єктний доступ	Власний інтерфейс тільки для віртуальних машин vmware	Немає
	Блоковий доступ	немає	iSCSI
	Файловий доступ	немає	NFS

Програмно-апаратні комплекси

Дані рішення, як правило, являють собою закінчений комплекс устаткування й програмного забезпечення системи зберігання. Вони мають більшу надійність і забезпечуються розширеною технічною підтримкою виробника, протестировани на сумісність і працюють стабільно.

Як правило, дані системи служать трьом основним цілям.

– Продовження життя існуючих СЗД, шляхом віртуалізації їхньої ємності. Використовуючи віртуальні системи зберігання даних, замовник абстрагується від апаратної складової й одержує можливість проводити обслуговування, міграцію й розширення ємності шляхом додавання нових систем прозоро для застосунка. Такі рішення дозволяють поєднувати невеликі системи зберігання для збільшення обсягу, продуктивності й відказостійкості.

– Розширення функціонала існуючих СЗД. Як правило, система SDS має більше широкий спектр можливостей і дозволяє працювати з більшістю

апаратних платформ. Системи доповнюють уже існуючі або системи низького рівня багатим набором функцій.

– Створення єдиного пула ресурсів зберігання. Єдиний пул дозволяє підвищити ефективність зберігання, шляхом створення рівнів зберігання даних з автоматичною міграцією, а також спрощує керування й моніторинг.

Таблиця 2.4 – Характеристики програмно-апаратних комплексів

Характеристики		IBM SVC	HP StoreVirtual	HITACHI CONTENT PLATFORM
Платформа	Підтримувана архітектура	Стандартний сервер IBM на базі x 86-64	Стандартний сервер HP на базі x 86-64	Стандартний сервер Hitachi Data Systems на базі x 86-64
Захист даних	Віддалена асинхронна реплікація	Так	Так	так
	Миттєві знімки	Читання/сапис	Читання/сапис	Читання/сапис
	Шифрування даних	Немає	Немає	Так
	Гарантоване видалення даних	Немає	Немає	Так
	Захист від видалення даних. Режим WORM	Немає	Немає	Так

Продовження таблиці 2.4

Характеристики		IBM SVC	HP StoreVirtual	HITACHI CONTENT PLATFORM
Продуктивність	Автоматичне балансування навантаження між вузлами	Так	Немає	Міграція даних "у слід за навантаженням" у гео-кластері
	Обмеження по масштабуванню ємності	32 ПБ. 8 вузлів	32 вузла	80ПБ, 80 вузлів
	Лінійне масштабування продуктивності	Так	Так	Так
	QoS	Так	Немає	Так
Функціонал	Інтеграція із традиційними СЗД	Так, IBM Storwzie	Немає	Інтеграція зі СЗД Hitachi HUS, VSP, HNAS.
	Дедуплікація	Немає	Немає	так
	Стиск	Так	Немає	так
	Багаторівневе зберігання (Tiering)	Так	Так	Так
	Гео-розподілені рішення	Так	Так	Так

Продовження таблиці 2.4

Характеристики		IBM SVC	HP StoreVirtual	HITACHI CONTENT PLATFORM
Інтерфейс	Об'єктний доступ	Немає	Немає	S3, Swift
	Блоковий доступ	iSCSI, FC, FCo	FC, iSCSI	Немає
	Файловий доступ	Немає	CIFS, NFS, HTTP, FTP	HTTP, SMB, NFS, WebDAV

Сприйняття, інформованість і затребуваність програмно-визначаємих середовищ

Для замовників основними передумовами до переходу на SDS є:

– висока вартість апаратних СЗД у порівнянні з вартістю широко розповсюджених серверів;

– централізація основних функцій в обмеженому числі контролерів СЗД і обмеження масштабування систем.

Раніше вважалося, що паралельні файлові системи призначені для інтернет-компаній або наукових організацій, а зараз подібне відношення переноситься на SDS. Тим часом, деякі з них мають більшого функціонала, що сполучається із проробленими механізмами розгортання, і зручним користувальницьким інтерфейсом. Тому SDS – гарний вибір не тільки для компаній, чий бізнес заснований на IT, але для корпорацій, які є споживачами традиційних СЗД.

Основною перевагою рішень SDS замовники насамперед уважають низьку вартість володіння, що може непередбачено збільшитися за рахунок росту витрат

на підтримку роботи системи у випадку унікальних рішень і непередбаченої надійності апаратних ресурсів.

Замовникам в основному знайомі рішення від відомих виробників, що пропонують SDS разом з яким-небудь добре зарекомендували себе продуктом. Так, наприклад, HP StoreVirtual VSA пропонується разом із широко відомої й добре зарекомендувала себе апаратною версією HP LeftHand p4000, a DataONTAP Edge – для організації резервування в центрі даних з територіально віддалених офісів.

Деякі виробники заздалегідь вбудовують рішення SDS у свої успішні продукти, рятуючи замовника й інтегратора від необхідності проектування й розробки власних рішень. Як приклад можна привести VMware VSAN, що пропонується разом з vSphere для зберігання віртуальних машин. Закінчене рішення з віртуалізації являє собою Nutanix, що включає власний модуль зберігання – NDFS. Компанія RedHat включила у свій портфель продуктів opensource розробку GlusterFS під ім'ям RedHat Storage, забезпечивши комерційною підтримкою.

Завдання

Рішення SDS сьогодні затребувані, насамперед, там, де необхідно знизити до мінімуму ціну ємності за рахунок використання або існуючого парку застарілих серверів, або за рахунок придбання дешевого встаткування. Основні завдання, ефективно розв'язувані за допомогою SDS:

– зберігання резервних копій або архівів;

– середовища розробки й тестування;

– хмарні обчислення й хостинг.

При цьому деякі SDS уже можуть застосовуватися для рішення більше широкого спектра завдань, таких як:

– створення програмно-визначаємих центрів обробки даних;

– об'єднання існуючих апаратних ресурсів у єдину систему зберігання для підвищення гнучкості й відказостійкості;

– створення модульних обчислювальних середовищ на базі ідентичних багатофункціональних вузлів;

– розширення функціонала існуючих систем зберігання даних. Об'єднання декількох систем зберігання даних у єдиний пул ресурсів;

– створення унікальних рішень із високим рівнем відказостійкості, продуктивності й ефективності.

Очікування й перспективи

Впроваджуючи SDS, замовники очікують від інфраструктури:

– зниження вартості – економія в сформованій складній ситуації на ринку;

– підвищення продуктивності – побудова високопродуктивних систем, без покупки СЗД класу Hi-End;

– розширення функціонала – одержання функціонала традиційних СЗД, для рішень, у яких недоцільне застосування реальних СЗД.

При цьому більшість очікує, що програмно-визначаємі системи зберігання даних будуть мати наступні характеристики:

Гнучкість (flexibility) – завдяки підтримці більшості розповсюджених апаратних платформ, системи SDS дозволять практично миттєво додавати або заміняти вузли зберігання без прив'язки до виробника встаткування, форм-фактору. Строки розробки нового функціонала в програмно-визначаємих системах зберігання помітно нижче, ніж для створення нової апаратної платформи, що підтримує необхідного функціонала.

Віртуалізація ресурсів (Resource Virtualization) – функціонал системи не прив'язаний до апаратних компонентів. Це забезпечить заміну й обслуговування компонентів без зупинки системи, дозволить знизити витрати на керування розрізненою інфраструктурою. Можливість сполучення програмно-визначаємих підсистем зберігання з підсистемами віртуалізації обчислення й мережі дозволяють підвищити утилізацію ресурсів.

Інтерфейс програмування (APIs) – програмно-визначаємі системи зберігання нададуть широкий функціонал і великий набір інтерфейсів взаємодії із

суміжними системами, дозволяючи створювати закінчені рішення, автоматизувати рутинні процеси знизивши ризик помилки й скоротивши витрати.

Зручність керування (Ease of Management) – SDS представить користувачеві єдину точку входу для керування всією підсистемою зберігання, крім необхідності налаштування окремих компонентів.

Заміна компонентів (Component Replacement) – програмно-визначаємі середовища дозволять абстрагуватися від апаратних ресурсів і робити заміну, міграцію й обслуговування компонентів практично без впливу на продуктивність і без припинення сервісу.

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 користувальницьким 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 взнають синтаксис із фігурними дужками, характерний для мови Visual C#. Розроблювачі, що знають кожну 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` – точку входу застосунка – інкапсулюється у визначення класів. Клас може 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 підтримуюче спадкування.

На застосунок 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 застосунок 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.

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, називаному 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.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи Software Defined Storage.

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

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

По суті, SDS – це програмний рівень, який відокремлює фізичні ресурси зберігання від базових пристроїв зберігання даних. Замість того, щоб прив’язувати дані безпосередньо до власного обладнання, SDS використовує віртуалізацію сховища для створення гнучкої та економічно ефективної інфраструктури зберігання даних.

На відміну від традиційних систем зберігання даних, таких як NAS та SAN, де зміни ємності або продуктивності часто вимагають заміни всіх фізичних ресурсів зберігання, система SDS робить масштабування майже безперешкодним. Ви можете додавати більше дисків, вузлів або навіть хмарну ємність без переписування програм або зміни робочих процесів. Однак віртуалізація має недоліки, тому деякі організації неохоче впроваджують технологію SDS.

Коротше кажучи, програмно-визначені рішення для зберігання даних дозволяють керувати складними операціями зберігання даних з єдиного інтерфейсу, підвищуючи гнучкість та знижуючи витрати. Хіба це не та перевага, яка робить SDS кращим підходом до зберігання даних у 2025 році?

Які переваги та недоліки сучасної системи SDS?

Впровадження програмно-визначених сховищ різко зросло з однієї причини: це вирішує проблеми, які не дають системним адміністраторам та DevOps спати вночі. Вигоди переважують головний біль, особливо якщо ви перестали платити данину завищеним постачальникам обладнання.

Ось що насправді дає вам SDS:

1. Гнучкість у виборі обладнання

За допомогою SDS ви можете запускати сховище даних на виділених серверах або пристроях, які легше оновити або замінити. Це зменшує залежність

від масивів певних постачальників і робить фізичні ресурси сховища більш адаптивними.

2. Масштабованість у всіх напрямках

Завдяки віртуалізації сховища, SDS масштабується як вертикально (додаючи більше потужності до існуючих вузлів), так і горизонтально (додаючи більше вузлів або дисків), що робить керування ресурсами сховища приємним процесом.

3. Економічно ефективність

Оскільки SDS абстрагує програмний рівень від фізичних ресурсів зберігання, вам не потрібно інвестувати в дорогі традиційні рішення для зберігання даних. Натомість ви можете перепрофілювати стандартне обладнання та розширювати його лише за потреби.

4. Централізоване управління сховищами

Єдина панель інструментів на базі програмного забезпечення для керування та автоматизації дозволяє адміністраторам керувати реплікацією, резервним копіюванням та моніторингом.

5. Висока доступність та аварійне відновлення

SDS дозволяє реплікувати дані між кількома вузлами або навіть географічними регіонами. У разі збою обладнання або форс-мажору відновлення відбувається швидше та надійніше.

6. Підтримка багатотипного зберігання даних

Добре розроблене програмно-визначене рішення для зберігання даних може поєднувати блочне, файлове та об'єктне сховище в одній уніфікованій системі, оптимізуючи інфраструктуру зберігання даних у різних відділах.

7. Виділений сервер

Виділений хостинг для тих, кому потрібна більша потужність, контроль та справжня стабільність.

Але користувачі технології програмно-визначеного сховища вказують на деякі недоліки:

Оскільки SDS повністю залежить від програмного рівня, продуктивність та надійність залежать від правильного налаштування та моніторингу. Неправильна конфігурація або недостатній моніторинг призводять до погіршення продуктивності.

Розгортання та обслуговування системи SDS вимагає навчання з віртуалізації та автоматизації сховищ. Брак досвіду може уповільнити впровадження.

Абстрагуючи сховище від апаратного забезпечення, SDS вводить нові міркування безпеки. Неправильно налаштоване програмне забезпечення для керування та автоматизації або неконтрольовані пристрої зберігання даних можуть стати векторами атаки.

Як керувати та оптимізувати програмно-визначену систему зберігання даних?

Розгортання програмно-визначеного сховища даних – це не фінішна пряма, а саме початок справжньої роботи. Ставлення до нього як до рішення за принципом «налаштував і забув» – це найшвидший спосіб виявити, що ваш рівень сховища даних перетворився на найслабшу ланку вашої інфраструктури.

Перша проблема – це планування ємності сховища. Занадто висока оцінка – і ви витрачаєте бюджет на обладнання, яке простоює. Занадто низька оцінка – і ви будете боротися з вузькими місцями у найневідповідніший момент. Баланс досягається шляхом моніторингу реальних робочих навантажень, прогнозування зростання за допомогою чітких показників та масштабування навмисно, а не реактивно.

Програмне забезпечення для управління та автоматизації є основою SDS, але воно надійне настільки, наскільки надійне його налаштування. Панелі інструментів та сповіщення повинні зменшувати кількість людських помилок, а не замінювати оперативну обізнаність. Відстеження IOPS, затримки та

пропускної здатності в режимі реального часу – це спосіб виявити проблеми, перш ніж вони переростуть у збої.

Стратегії резервного копіювання та відновлення також змінюються в рамках SDS. Знімки та реплікація між вузлами або регіонами значно спрощують відновлення після збоїв, але лише за умови дотримання та тестування політик. Занадто багато команд вважають, що реплікація відбувається тому, що вона була налаштована один раз; на практиці кластер без моніторингу може швидко вийти з синхронізації. Практики безпеки для захисту даних також повинні залишатися суворими.

Зрештою, **підтримка постачальників або спільноти** – це тиха, але критично важлива частина оптимізації SDS. Стек, який сьогодні виглядає надійним, може стати крихким, якщо оновлення перестануть надходити або спільнота проекту втратить імпульс. Перевірка стану екосистеми так само важлива, як і перевірка стану власного кластера.

3.2 Розробка структурної схеми

Програмно-визначене сховище зазвичай буває двох видів: конвергентна інфраструктура та гіперконвергентна інфраструктура.

Конвергентна інфраструктура (CI):

– Структура: Обчислення, мережа та сховище даних – це окремі апаратні блоки, об'єднані між собою спеціальною інтерфейсною структурою.

– Продуктивність: Висока пропускна здатність і передбачувана затримка – добре, якщо ваші робочі навантаження пов’язані з великим обсягом операцій вводу/виводу.

– Відповідність варіантам використання: застарілі програми, транзакційні бази даних або середовища, де спеціалізація обладнання все ще має значення.

– Компроміси: масштабування незграбне, залежність від постачальника залишається, а гнучкість обмежена.

Гіперконвергентна інфраструктура (HCI):

– Структура: Обчислення, мережа та сховище даних об'єднані в один рівень, керований програмним забезпеченням.

– Масштабованість: додавання вузлів, масштабування. Просто. Без модернізації навантажувача.

– Відповідність варіантам використання: хмарні стеки, оркестрація контейнерів, конвеєри з інтенсивним використанням DevOps та робочі навантаження штучного інтелекту/машинного навчання.

– Компроміси: значною мірою залежить від експертизи програмного забезпечення – неправильно налаштуйте його, і ви щойно підвищите свій кластер до аномалії класу Keter.

Що обрати у 2025 році?

– Оберіть неперервну інтеграцію (CI), якщо ви використовуєте застарілі системи або вам потрібні гарантії продуктивності без будь-яких пошкоджень.

– Оберіть HCI, якщо бажаєте використовувати його за замовчуванням для сучасних програмно-визначених середовищ зберігання даних. Якщо відповідність вимогам або застарілі обмеження не змушують вас використовувати CI, HCI позбавить вас головного болю.

Коли варто використовувати програмно-визначену систему зберігання даних

Не кожній організації потрібне програмно-визначене сховище з першого дня. Перевірте ці випадки, щоб визначитися.

– SDS спрощує налаштування та керування віртуальними машинами, робочими столами та програмами в гібридних або багатохмарних системах.

– Якщо ви аналізуєте поведінку клієнтів, ринкові тенденції або бізнес-рушії, SDS забезпечує єдиний доступ до великих обсягів сховища даних. Його централізований програмний рівень робить аналітику швидшою та легшою для автоматизації.

– Реплікація на кількох пристроях зберігання даних та в різних регіонах

забезпечує швидке відновлення під час збоїв, що робить SDS кращим вибором для організацій, де простої дорівнюють фінансовим втратам.

– Деякі підприємства використовують програмно-визначені рішення для зберігання даних на периферії для локальної обробки даних, зберігаючи при цьому синхронізацію з централізованим обладнанням для зберігання даних. Це забезпечує продуктивність та стійкість навіть у розподілених середовищах.

Як захистити програмно-визначену систему зберігання даних?

Безпека – це те, де SDS або витримує тиск, або стає дуже дорогим звітом про інциденти. Оскільки SDS переносить контроль на рівень програмного забезпечення, неправильна конфігурація не просто можлива – вона неминуча без дисципліни.

1. Обмежте права root/адміністратора до найменшої можливої групи. Кожна дія має бути зареєстрована та відстежена, а не закопана на сервері журналів, який ніхто не перевіряє до завершення збою.

2. Дані в стані спокою, під час передачі, між вузлами – шифруйте все. Будь-що менше – це відкрите запрошення до витоку даних. Базовий рівень: TLS для трафіку, AES для зберігання.

3. Автоматизуйте сканування на вразливості та не покладайтеся на електронні листи постачальників, щоб дізнатися про випадки зараження.

4. Перевіряйте кожен запит, ізолюйте робочі навантаження та сегментуйте ресурси сховища. Якщо один вузол скомпрометовано, радіус вибуху слід вимірювати в гігабайтах, а не петабайтах.

5. Відкритий код SDS забезпечує прозорість, але також робить вразливості публічними. Відстежуйте частоту виправлень спільноти.

Яка різниця між SDS, NAS та SAN для зберігання даних?

Коли люди оцінюють сховище даних, вони завжди використовують одні й ті ж три абревіатури. Ось їх простий розбив:

Програмно-визначене сховище (SDS)

SDS абстрагує сховище на програмно-керований рівень на стандартному

обладнанні. Підтримує блоки, файли та об'єкти в одній системі.

– Сильні сторони: Висока масштабованість, незалежність від апаратного забезпечення, автоматизація. Ідеально підходить для гібридної хмари, конвеєрів штучного інтелекту/машинного навчання або робочих процесів DevOps.

– Слабкі сторони: Складніший у розгортанні, вимагає фактичного досвіду та додає поверхню для атаки у разі неправильного налаштування.

– Перевірка реальності: якщо ви хочете гнучкості та втомилися від прив'язки до постачальника, то саме в цьому напрямку рухається галузь – це не «новітній розвиток», це стандартний процес.

Мережеве сховище даних (NAS)

Мережеве сховище даних (NAS) – це архітектура сховища даних на основі файлів, яка дозволяє користувачам отримувати доступ до даних через мережу, зазвичай за допомогою протоколів NFS або SMB.

– Сильні сторони: Просто, дешево, працює одразу після розпакування.

– Слабкі сторони: залежність від одного обладнання для зберігання даних, незграбне масштабування та обмежена продуктивність.

– Перевірка реальності: Чудово підходить для невеликої команди або лабораторного середовища. Завантажте його на виробниче навантаження, і ви зрозумієте, чому системні адміністратори ненавидять фразу «просто поставте його на NAS».

Мережа зберігання даних (SAN)

SAN, або мережа зберігання даних, – це блочна архітектура сховища, яка з'єднує сервери зі сховищем через виділену мережу за допомогою протоколів Fibre Channel або iSCSI.

– Сильні сторони: Низька затримка, висока пропускна здатність, надійна робота з критично важливими для продуктивності корпоративними робочими навантаженнями.

– Слабкі сторони: Власницька розробка, дорога, масштабування – це джерело доходу постачальника, а не ваш вибір.

– Перевірка реальності: SAN – це «безпечний варіант», який досі просувають фінансові та традиційні IT-фахівці. Він працює – доки вам не знадобиться масштабування або оплата рахунків.

Який вам слід обрати?

SDS: найкраще підходить для організацій, що переходять на гібридну хмару, робочі навантаження зі штучним інтелектом/машинним навчанням або автоматизовані середовища зберігання даних.

NAS: підходить для невеликих команд, яким потрібен простий обмін файлами з мінімальними накладними витратами.

SAN: все ще цінна для критично важливих для продуктивності робочих навантажень, але менш адаптивна, ніж сучасні програмно-визначені рішення для зберігання даних.

Програмно-визначене сховище даних (SDS) – це базовий варіант для тих, хто хоче масштабовану інфраструктуру без необхідності платити постачальникам обладнання за цю перевагу. Абстрагуючи програмний рівень від фізичних пристроїв, SDS надає вам гнучкість для зростання, відновлення та адаптації до ваших робочих навантажень.

Впровадження SDS означає планування потужності, моніторинг IOPS/затримки, забезпечення належних політик резервного копіювання та реплікації, виконання регулярних оновлень та перевірку справності стеку. Натомість ви отримуєте гнучкість обладнання, передбачувану масштабованість та зменшену залежність від пристроїв різних постачальників.

Коротко кажучи:

– Що робить SDS: він абстрагує сховище за допомогою програмного рівня, усуваючи залежність від конкретного обладнання та знижуючи сукупну вартість володіння (TCO) завдяки стандартним серверам та еластичному зростанню.

– Що змінюється на практиці: стало легше масштабувати горизонтально та вертикально, швидше відновлюватися після збоїв та стандартизувати

управління в гібридних та багатохмарних середовищах.

– Хто отримує найбільшу вигоду: команди, чиї робочі навантаження не зростають за календарним графіком – конвеєри штучного інтелекту/машинного навчання, аналітика, веб-проекти з високим трафіком та ландшафти DevOps.

NAS та SAN все ще мають свої ніші, але якщо ви створюєте системи, які повинні витримувати реальний трафік та реальні збої, SDS – це практичний вибір.

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Програмно-визначаємі системи зберігання (SDS) – це один з основних компонентів програмно-визначаємого ЦОД. SDS принципово відрізняються від

класичних і блокових сховищ. Головний принцип таких систем – повний перенос організації зберігання з апаратних контролерів на програмний рівень.

Основні переваги:

– Економічність. Програмно-визначаємі сховища дають можливість відмовитися як від пропрієтарних рішень на користь стандартизованого устаткування, так і від традиційної мережі зберігання даних (SAN) Fibre Channel на користь більше універсального стандарту Ethernet.

– Гнучкість. Програмно-визначаємі сховища можуть бути побудовані на устаткуванні будь-якого виробника або навіть різних виробників у рамках однієї системи. SDS легко масштабуються, оскільки програмний рівень організації надає дуже широкі можливості по керуванню ресурсами.

– Відказостійкість. При виникненні проблем на основний SDS за допомогою засобів реплікації дані легко й бесшовно переносяться на резервну систему.

– Зручність експлуатації. IT-служба розширює SDS без оглядки на виробника кожної конкретної одиниці встаткування, легко перерозподіляє дані між різними рівнями зберігання для найбільш раціональних використань ресурсів.

– Продуктивність. Програмно-визначаємі сховища підтримують підключення PCIe NVMe-карт, що істотно збільшує можливості систем.

На рисунку 3.2 зображена функціональна схема системи. З неї бачимо, що розроблена система підвищення стійкості до уражень інформації яка записана на дисках систем Software Defined Storage для зберігання даних складається з наступних основних функціональних блоків:

– сам носій інформації – система Software Defined Storage для зберігання даних;

– кодер Ріда-Соломона, який використовується, коли відбувається запис інформації на диск системи Software Defined Storage для зберігання даних, при цьому необхідно враховувати, що об’єм інформації, яка записується на диск системи Software Defined Storage для зберігання даних, повинна бути меншою,

ніж об’єм диску системи Software Defined Storage для зберігання даних, 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 с відповідного диску системи Software Defined Storage для зберігання даних.

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

– дані, які потрібно захистити;

– поліном для кодування;

– відмовостійки коди.

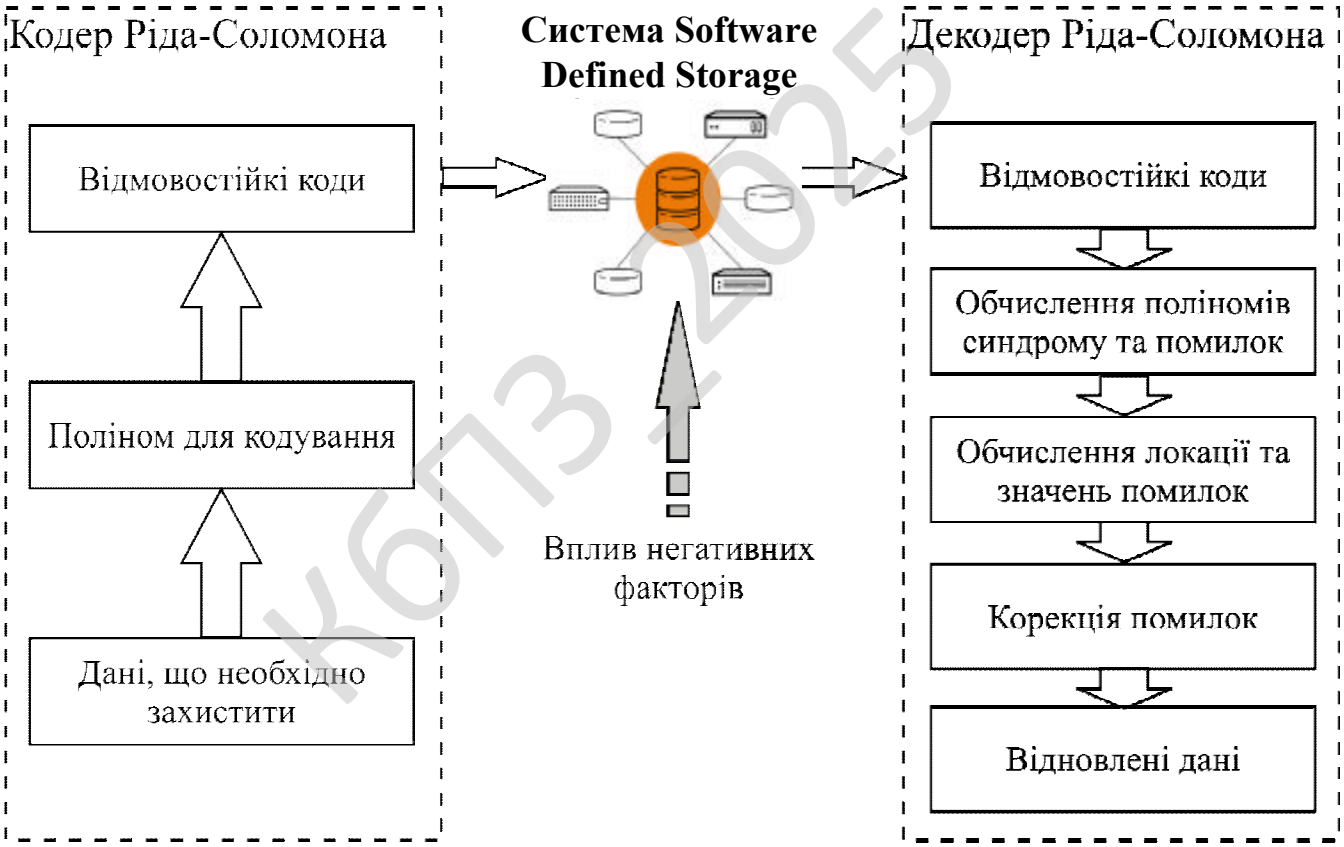

Рисунок 3.2 – Функціональна схема системи

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.

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:

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

а кодове слово формується за допомогою операції:

$$c(x) = g(x).i(x),$$

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.
– Si – Синдроми.
– L(x) – Поліном локації помилок.
– Xi – Положення помилок.
– Yi – 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 символьних 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 символьних 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 (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 Массачуссетского 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 Месси.

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-Хоквингхема (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) = m(x)g(x), де ступінь m(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 перевірочних символів r, число інформаційних символів k = n – r.

Матриця, що породжує

Поліноми $g(x), xg(x), x^2g(x), \ldots, x^{k-1}g(x)$ 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:

$$\overline{m}G = (m_0, m_1, ..., m_{k-1}) \begin{bmatrix} g(x) \\ xg(x) \\ \ldots \\ x^{k-1}g(x) \end{bmatrix} = m(x)g(x), \qquad (3.1)$$

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

Матрицю G можна записати в символьній формі:

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

Перевірочна матриця

Для кожного кодового слова циклічного коду справедливо $c(x) = 0 \mod g(x)$. Тому перевірочну матрицю можна записати як $H = \begin{bmatrix} 1 & x & x^2 & \ldots & x^{n-2} & x^{n-1} \end{bmatrix} \mod g(x)$.

Тоді:

$$\overline{c}H^T = \sum_{i=0}^{n-1} c_i x^i \bmod g(x). \qquad (3.2)$$

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), корінням якого є d – 1 підряд, що йдуть ступенів, $\alpha^{l_0}, \alpha^{l_0+1}, \ldots, \alpha^{l_0+d-1}$ елемента α, є поліномом, що породжує, коду над полем GF(q) $g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1}) \ldots (x - \alpha^{l_0+d-1})$:, 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) ≤ 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.

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

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 поле Галуа, те самі операції кодування/декодування вимагають багато ресурсів і часу. Швидкий алгоритм декодування, заснований на швидкому перетворенні Фур'є, виконується за час порядку lnn^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 перетворяться 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 інший доцільно використовувати таблицю істинності розміром 256 байт, що становить ємність одного EAB (Embedded Array Block – блок зосередженої пам'яті). Для реалізації кодеру потрібно 16 таких перемножувачів, при цьому те саме число множиться на різні константи, що дозволяє використовувати для його перекладу в індексну форму один EAB.

Для перекладу результатів у поліноміальну форму потрібно вже 16 таких таблиць, що вимагає застосування IMC 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 засувок 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 IMC EPF10K10. Після оптимізації розміщення схеми на кристалі FPGA швидкодія схеми досяглася 11,57 Мгц (частота надходження байт даних, далі – байтова частота).

При використанні IMC 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-Месси, 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-Месси. 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 $a, b, r_1 > r_2 > r_3 > r_4 > \cdots > r_n$ визначена тим, що кожне r_k це остача від ділення перед-попереднього числа на попереднє, а передостаннє ділиться на останнє націло, тобто

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

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

Існування таких $r_1, r_2, \ldots$, 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 випливає 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 (ECC), запису на система Software Defined Storage для зберігання даних. 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.

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

3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників.

Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

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.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграми потоків даних містять чотири типи елементів:

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

– Сховища даних (репозиторії).

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 ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо реалізацію магістерської дипломної роботи. Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень. Блок-схеми показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. 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 високого рівня декомпозиції задач на класи.

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 підсистеми 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.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий

стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

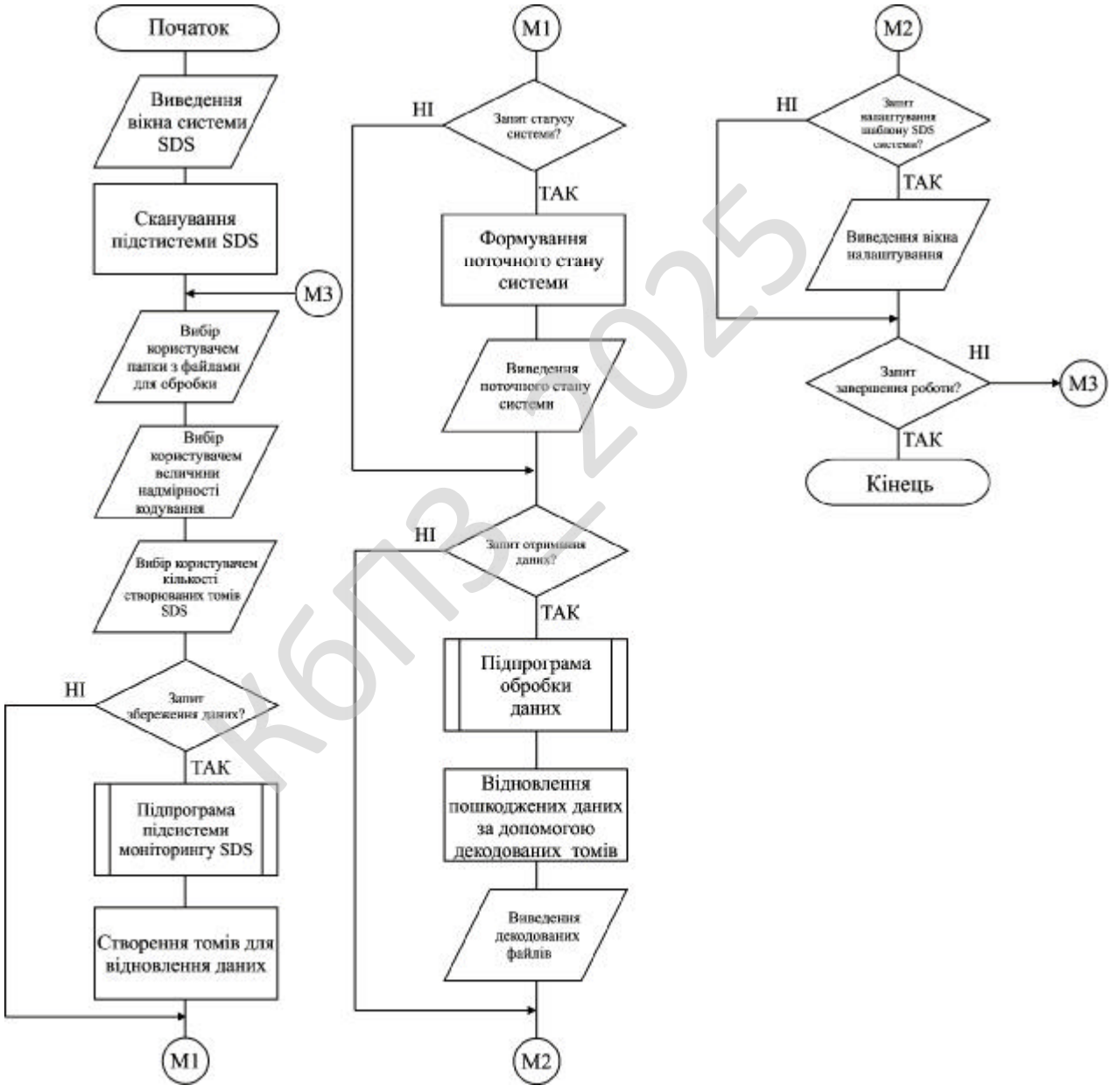

Рисунок 4.1 – Блок-схема основної програми

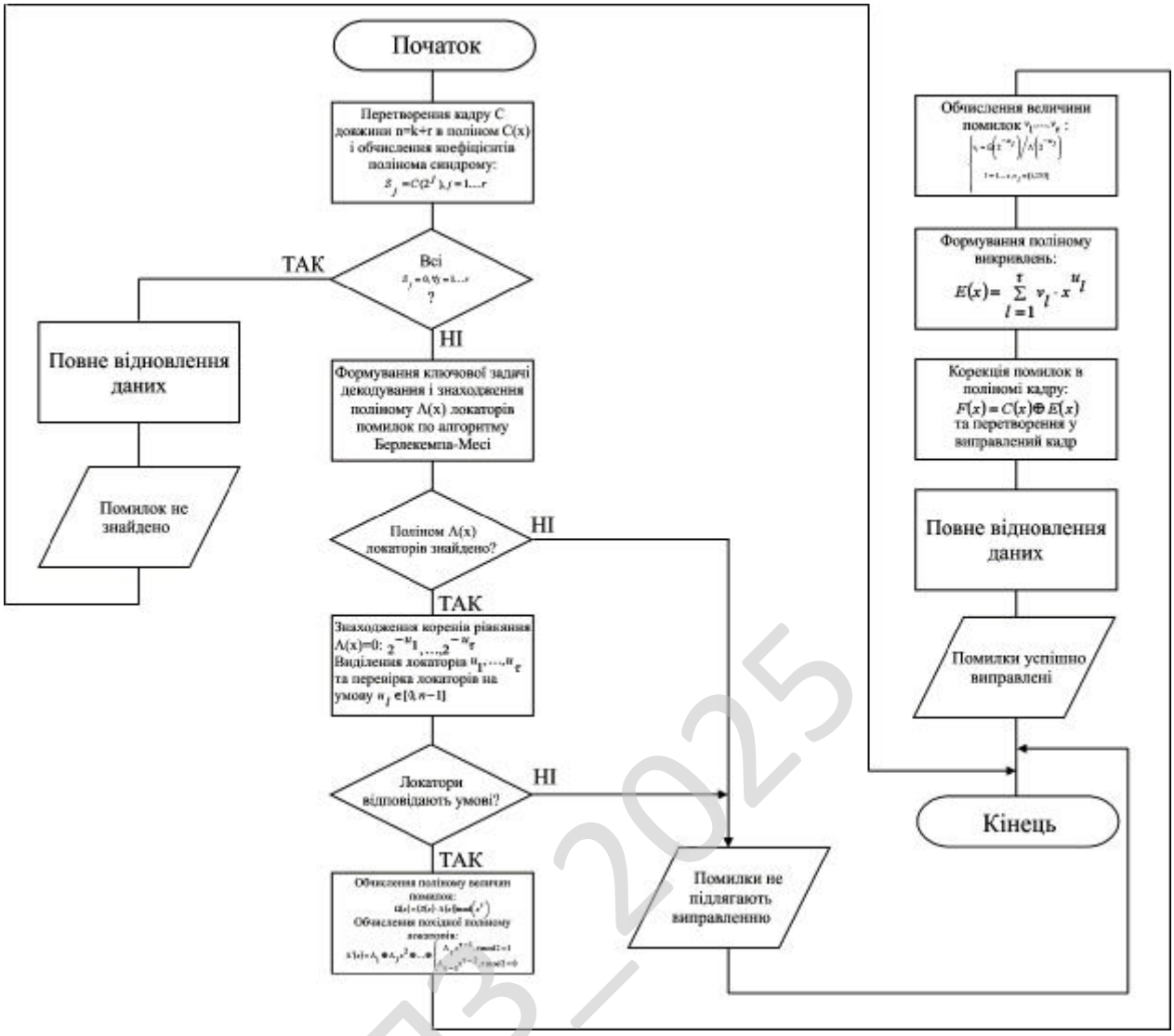

Рисунок 4.2 – Блок-схема роботи підпрограми

Було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів; Діаграма компонент.

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.

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.

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: проектована 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.

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 посилатися 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 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.

Діаграма компонент в UML це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

Модуль програмного забезпечення може бути представлено в якості компоненти. Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми.

Діаграма компонент відображає лише структурні характеристики, для відображення окремих екземплярів компонент слід використовувати діаграму розгортування.

Компоненти об'єднуються разом використовуючи структурні зв'язки (assembly connector) щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

Структурна взаємодія – «зв'язок двох компонент, який передбачає, що один з них надає послуги, потрібні іншому компоненту».

При використанні діаграми компонент щоб показати внутрішню структуру компонента, клієнтські та серверні інтерфейси можуть утворювати пряме з'єднання з внутрішніми. Таке з'єднання називається з'єднанням делегації.

Розглянемо формат що використовується – **JSON** (JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється джейсон) – це текстовий формат обміну даними між комп'ютерами.

JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). Розробив і популяризував формат Дуглас Крокфорд.

JSON знайшов своє головне призначення у написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти.

JSON з'явився через необхідність обміну даними з сервером у реальному часі без використання плагінів для браузерів, flash-додатків або Java-аплетів, які використовувались скрізь на початку 2000-х років. Дуглас Крокфорд був тим, хто активно просував новий на той час формат. Він з колегами хотів створити технологію, яка використовувала б можливості звичайного браузера давала б веб-розробникам можливість створювати веб-додатки із постійним двостороннім зв'язком із веб-сервером.

JSON вперше був використаний в проекті в Communities.com для Cartoon Network, він дозволяв обмінюватись повідомленнями і одночасно маніпулювати DHTML-елементами.

Веб-сайт JSON.org було запущено 2002 року. У грудні 2005-го року Yahoo! почав переводити деякі зі своїх веб-сервісів на роботу з JSON. Google взялася до роботи з технологією у своєму веб-протоколі GData у грудні 2006-го року.

Використання

За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур.

Якщо говорити про веб-застосунки, в такому ключі він доречний в задачах обміну даними як між браузером і сервером (AJAX), так і між самими серверами (програмні HTTP-інтерфейси).

Формат JSON так само добре підходить для зберігання складних динамічних структур в реляційних базах даних або файловому кеші.

Приклад використання JSON (JavaScript)

```
const ajaxData = '{"name": "wiki", "fname": "pedia", "rates": [1, 4, 5, 6]}'
const ajaxObj = JSON.parse(ajaxData)
console.log(`${ajaxObj.name}_${ajaxObj.rates[2]}`) /* Виведе "wiki_5"*/
```

Синтаксис

JSON будується на двох структурах:

1. Набір пар назва/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом.

2. Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах.

У JSON використовуються такі їхні форми:

1. Об'єкт – це послідовність пар назва/значення. Об'єкт починається з символу { і закінчується символом } . Кожне значення слідує за : і пари назва/значення відділяються комами.

2. Масив – це послідовність значень. Масив починається символом [і закінчується символом]. Значення відділяються комами.

3. Значення може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну.

4. Рядок – це послідовність з нуля або більше символів юнікода, обмежена подвійними лапками, з використанням escape-послідовностей, що починаються зі зворотної косої риски \. Символи представляються простим рядком. Тип Рядок (String) дуже схожий на String в мовах C і Java. Число теж дуже схоже на C- або Java-число, за винятком того, що вісімкові та шістнадцяткові формати не використовуються. Пропуски можуть бути вставлені між будь-якими двома лексемами.

Наступний приклад показує JSON представлення об'єкта, що описує людину. У об'єкті є рядкові поля імені і прізвища, об'єкт, що описує адресу, і масив, що містить список телефонів.

```
{
   "firstName": "Іван",
   "lastName": "Коваленко",
   "address": {
       "streetAddress": "вул. Гру 14, кв.101",
       "city": "Київ",
       "postalCode": 21000
   },
   "phoneNumbers": [
       "044 123-1234",
       "050 123-4567"
   ]
}
```

Використання JSON в AJAX

Наступний фрагмент коду JavaScript показує, як клієнт може використати XMLHttpRequest для запиту об'єктів в форматі JSON з серверу. (Серверна частина коду пропущена, вона просто повертає на запит URL рядок у JSON форматі).

Треба відзначити, що тут використання XMLHttpRequest не є кросс-браузерним (за деталями звертайтеся на сторінку XMLHttpRequest), і код зазнаватиме незначних модифікацій в різних версіях оглядачів Internet Explorer, Opera, Safari або Mozilla. Використання запиту XMLHttpRequest обмежено правилом одного джерела (same origin policy): URL, що відповідає на запит, має посилатися на той же сайт, що обслуговує сторінку, що ініціювала запит.

Оглядачі можуть також використовувати тег <iframe> для асинхронного запиту JSON-даних в кросс-браузерному варіанті, або використати просте перенаправлення <form action="url_to_cgi_script" target="name_of_hidden_iframe">. Такий підхід був поширений до приходу популярного нині запиту XMLHttpRequest.

Динамічний тег <script> також можна використати для підвантаження JSON-даних. Ця техніка можлива, щоб обійти надто суворе правило одного джерела, але вона не є безпечною. Запит JSONRequest пропонується, як безпечніша альтернатива.

Питання безпеки

Хоча JSON призначений для передачі даних в серіалізованому вигляді, його синтаксис відповідає синтаксису JavaScript і це створює низку проблем безпеки. Часто для обробки даних, отриманих від зовнішнього джерела у форматі JSON, до них застосовується функція eval() без якої-небудь попередньої перевірки.

JavaScript eval()

Оскільки JSON представляється синтаксично правильним фрагментом коду JavaScript, природним способом розбору JSON-даних в JavaScript-програмі є використання вбудованої в JavaScript функції eval(), яка призначена для

обчислення JavaScript-виразів. При цьому підході відпадає необхідність у використанні додаткових парсерів.

Техніка використання eval() робить систему вразливою, якщо джерело JSON-даних, що використовуються, не відноситься до надійних. Такими даними може виступати шкідливий JavaScript-код для атак за допомогою ін'єкції коду. За допомогою цієї вразливості можливо здійснювати крадіжку даних, підробку автентифікації. Проте, вразливість можна усунути за рахунок використання додаткових засобів перевірки даних на коректність. Наприклад, до виконання eval() отримані від зовнішнього джерела дані можуть перевірятися за допомогою регулярних виразів. У RFC, що визначає JSON пропонується використовувати такий код для перевірки його відповідності формату JSON

```
const my_JSON_object = !(/[^,:{}\[\]0-9.\-+Eaeflnr-u \n\r\t]/.test(
text.replace(/"(\\.|[^"\\])*"/g, ''))) &&
eval('(' + testedData + ')');
```

Як безпечніша альтернатива eval() була запропонована нова функція parseJSON(), здатна обробляти тільки JSON-дані.

Вбудований JSON

Останні версії веб-браузерів мають вбудовану підтримку JSON і здатні його обробляти без виклику функції eval(), що призводить до описаної проблеми. Обробка JSON у такому разі зазвичай здійснюється швидше. Так у червні 2009 року вбудовану підтримку JSON мали такі браузери: Mozilla Firefox 3.5+; SeaMonkey 2; Thunderbird 3; Microsoft Internet Explorer 8; Opera 10.5; Браузери, засновані на WebKit (наприклад, Google Chrome, Apple Safari).

Принаймні дві популярні бібліотеки JavaScript використовують вбудований JSON у разі його доступності: jQuery; Dojo.

Підробка крос-доменного запиту

Непродумане використання JSON робить сайти вразливими до підробки міжсайтових запитів (CSRF або XSRF).

Оскільки тег <script> допускає використання джерела, що не належить до того ж домену, що і використовуваний ресурс, це дозволяє виконувати код даних,

представлених у форматі JSON, в контексті довільної сторінки, що робить можливою компрометацію паролів або іншої конфіденційної інформації користувачів, що пройшли авторизацію на іншому сайті.

Це є проблемою тільки у разі вмісту в JSON-даних конфіденційної інформації, яка може бути компрометована третьою стороною і якщо сервер розраховує на політику одного джерела, блокуючи доступ до даних при виявленні зовнішнього запиту.

Це не є проблемою, якщо сервер визначає допустимість запиту, надаючи дані тільки у разі його коректності. HTTP cookie не можна використовувати для визначення цього. Виключне використання HTTP cookie використовується підробкою міжсайтових запитів.

Розширення, JSONP

JSONP або «JSON з підкладкою» є розширенням JSON, коли назва функції зворотного виклику вказується як вхідний аргумент. Спочатку ідея була запропонована в блозі MacPython в 2005 році, і в наш час використовується багатьма Web 2.0 застосунками, такими, як Dojo Toolkit Applications, Google Toolkit Applications і zanox Web Services.

Подальші розширення цього протоколу були запропоновані з урахуванням введення додаткових аргументів, як, наприклад, у разі JSONPP за підтримки S3DB вебсервісів.

Оскільки JSONP використовує скрипт-теги, виклики по суті відкриті світові. З цієї причини JSONP може бути недоречними для зберігання конфіденційних даних.

Включення скриптових тегів від віддалених сайтів дозволяє їм передати будь-який контент на сайті.

Якщо віддалений сайт має вразливості, які дозволяють виконати ін'єкції JavaScript, то початковий сайт також може бути ними зачеплений.

Розширення, BSON

BSON – це бінарна форма представлення простих структур даних і асоціативних масивів (які називають об'єктами або документами). Назва «BSON» заснована на визначенні JSON і неофіційно означає «Binary JSON» (бінарний JSON).

JSON Reference

Стандарт JSON не описує посилання на інші об'єкти або частини, але існує чернетка стандарту IETF для посилань на об'єкти на основі JSON. Посилання дозволяють здійснювати трансклюзію – вставляти одні документи в інші.

JSON Reference – це JSON-об'єкт з ключем "$ref" (всі інші ключі ігноруються) і значенням стрічкового типу що містить URI, наприклад:

```
{ "$ref": "http://example.com/example.json#/foo/bar" }
```

Якщо URI містить ідентифікатор фрагмента (в прикладі вище "/foo/bar"), він інтерпретується як JSON Pointer.

Модуль dojox.json.ref в Dojo toolkit, забезпечує підтримку декількох форм JSON Reference.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Threefish – в криптографії симетричний блоковий криптоалгоритм, розроблений автором Blowfish та Twofish, американським криптографом Брюсом Шнайером 2008 року для використання в геш-функції Skein і як універсальну заміну наявним блоковим шифрам. Основними принципами розробки шифру були: мінімальне використання пам'яті, необхідна для використання в геш-функції стійкість до атак, простота реалізації та оптимізація під 64-розрядні процесори.

Структура алгоритму

Threefish має дуже просту структуру і може бути використаний для заміни алгоритмів блочного шифрування, бувши швидким і гнучким шифром, що

працюють в довільному режимі шифрування. Threefish S-блоки не використовує, заснований на комбінації інструкцій виключаючого або, складання і циклічного зсуву.

Як і AES, шифр реалізований у вигляді підстановочно-перестановочної мережі на оборотних операціях, не будучи шифром мережі Фейстел.

Алгоритм передбачає використання tweak-значення, свого роду вектора ініціалізації, дозволяючи змінювати таким чином значення виходу, без зміни ключа, що має позитивний ефект як для реалізації нових режимів шифрування, так і на криптостойкости алгоритму.

Як результат думки авторів, що кілька складних раундів часто гірше ніж застосування великого числа простих раундів, алгоритм має нетрадиційно велику кількість раундів – 72 або 80 при ключі 1024 біт, проте, за заявою творців, його швидкісні характеристики випереджають AES приблизно вдвічі. Варто зауважити, що через 64-бітної структури шифру, дана заява має місцево лише на 64-розрядної архітектури. Тому, Threefish, як і Skein [1], заснований на ньому, на 32-розрядних процесорах показує значно гірші результати ніж на «рідному» обладнанні.

Ядром шифру є проста функція «MIX», перетворювальна два 64-бітових беззнакових числа, в процесі якої відбувається складання, циклічний зсув (ROL / ROR), і додавання по модулю 2 (XOR) .

Нижче представлений код MIX-функції для Threefish-1024 [2]:

```
<syntaxhighlight lang="C">
// Константи для циклічного зсуву
int R16 [8] [8] = {
  {55, 43, 37, 40, 16, 22, 38, 12},
  {25, 25, 46, 13, 14, 13, 52, 57},
  {33, 8, 18, 57, 21, 12, 32, 54},
  {34, 43, 25, 60, 44, 9, 59, 34},
  {28, 7, 47, 48, 51, 9, 35, 41},
  {17, 6, 18, 25, 43, 42, 40, 15},
  {58, 7, 32, 45, 19, 18, 2, 56},
  {47, 49, 27, 58, 37, 48, 53, 56},
};
```

```
// D – раунд, j – індекс в таблиці циклічного зсуву
void mix (int j, int d) {
  usigned long long rotl;
  y [0] = x [0] + x [1];
  rotl = R16 [d% 8] [j];
  y [1] = (x [1] << rotl) | (x [1] >> (64 – rotl));
  y [1] ^ = y [0];
}
</Source>
```

Процедура розшифрування обернена процедурі зашифрування і містить зворотну функцію DEMIX.

Кожен з 72 раундів Threefish-256 і Threefish-512 має чотири MIX перетворення, Threefish-1024 – вісім звернень до MIX функції.

Безпека

За заявою авторів, алгоритм має більш високий рівень безпеки, ніж AES. Існує атака на 25 з 72 раундів Threefish, в той час як для AES – на 6 з 10. Threefish має показник фактора безпеки 2.9, в свою чергу, AES всього 1.7 [3]

Для досягнення повної дифузії, шифру Threefish-256 досить 9 раундів, Threefish-512 – 10 раундів і Threefish-1024 – 11 раундів. Виходячи з цього, 72 і 80 раундів відповідно в середньому, забезпечать кращі результати, ніж існуючі шифри. [4]

У той же час, алгоритм має набагато простішу структуру і функцію перетворення, проте виконання 72-80 раундів, на думку дослідників, забезпечує необхідну стійкість. Вживаний розмір ключа від 256 до 1024 біт зводить нанівець можливість повного перебору паролів при так званій атаці грубою силою (brute force attack) на сучасному обладнанні.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у магістерської дипломної роботі програмне забезпечення системи Software Defined Storage. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ.

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

– Розділу обрання групи.

– Розділу виведення результату роботи системи.

– Функції представлені у графічному вигляді (іконки).

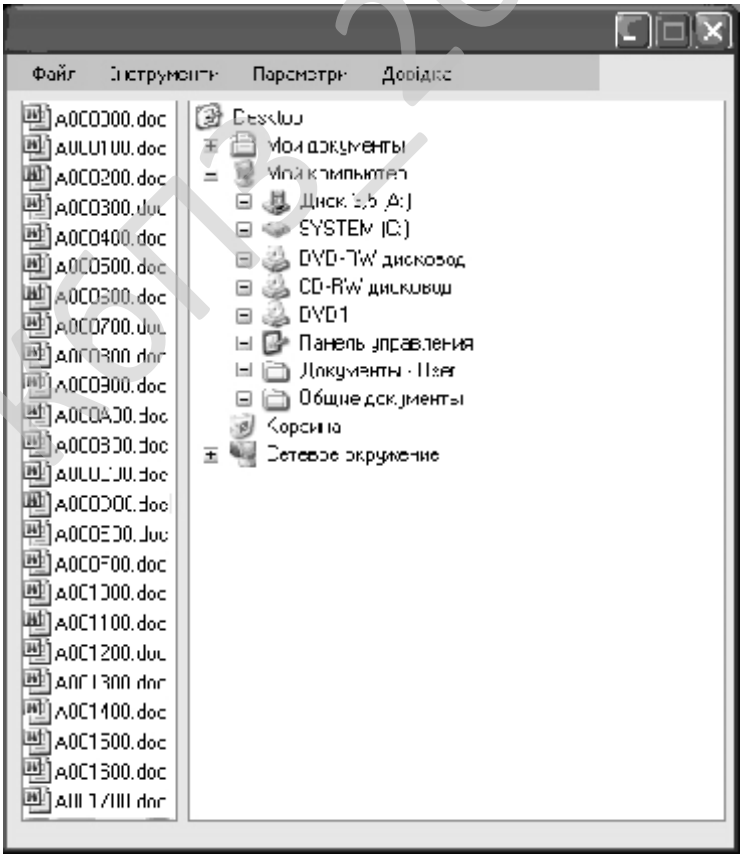

Рисунок 5.1 – Головне вікно ПЗ

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.

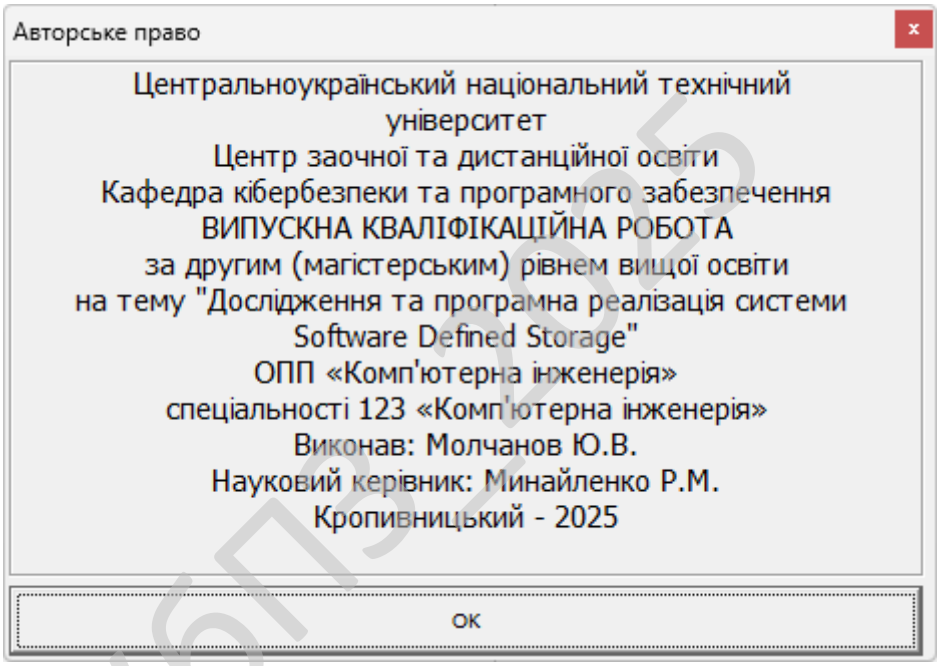

Рисунок 5.2 – Авторське право

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 IT рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене IT 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 IT 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.

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.

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», чию 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 ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

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.

– Помилок інтерфейсу.

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

– Помилок характеристик (необхідна ємність пам'яті і т.д.).

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – commercial software. Програмне забезпечення, створене комерційною організацією з метою отримання прибутку від його використання іншими, наприклад, шляхом продажу копій.

Найважливішою особливістю комерційний програмних продуктів є підтримка великих компаній, прямо зацікавлених у поширенні програм. Багато організацій надають виключно платну підтримку своїх продуктів, такий підхід, як правило, використовують організації надають відкриті вихідні коди. Для продуктів, що розповсюджуються на комерційній основі діють зазвичай безкоштовні служби підтримки, покликані збільшити рівень довіри у клієнтів і потенційних покупців.

Далеко не завжди, але як правило терміни критично важливих змін в комерційних продуктах значно менше, ніж у некомерційних проектів. Це пов'язано з тим, що над комерційним продуктом працюють цілі групи розробників і ця робота є їх основним заняттям. Розробникам-початківцям як правило доводиться шукати додаткові способи заробітку, і це збільшує час, що витрачається на доповнення і зміни програм. Так як основним рушійним фактором створення комерційного ПЗ є одержання прибутку, то комерційні програмні продукти першими заповнюють вільні ніші та пропонують варіанти вирішення завдань відразу по мірі виявлення вакууму в будь-якому секторі ринку.

Окремий вид комерційних програм, коли їх розробка оплачується безпосередньо замовником. Такі програми найчастіше позбавлені всіх переваг комерційних продуктів, оскільки мають обмежений бюджет, але більш адаптовані до вимог замовника, ніж аналоги.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи Software Defined Storage.

Метою розробки є дослідження та програмна реалізація системи Software Defined Storage.

Об'єктом дослідження є процес Software Defined Storage.

Предметом дослідження є методи Software Defined Storage.

Методи дослідження базуються на методах теорії побудови комп'ютерних мереж, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод Software Defined Storage.

– Розроблено вітчизняний продукт Software Defined Storage, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та розробки системи Software Defined Storage можуть бути насамперед корисними для підприємств, які мають розгалужену ІТ-інфраструктуру й використовують сервери, мережеве обладнання та корпоративні сервіси для підтримки своєї діяльності. Для таких компаній стабільність і безперебійність роботи інформаційних систем є критично важливими, тому можливість своєчасного виявлення несправностей або перевантажень стає суттєвою конкурентною перевагою. Саме система моніторингу допомагає контролювати роботу мережевих пристроїв у режимі реального часу, виявляючи проблеми ще до того, як вони вплинуть на користувачів.

Особливий інтерес до таких систем можуть проявити ІТ-компанії, які займаються наданням послуг хостингу, розробкою програмного забезпечення або підтримкою клієнтів. Для них швидкість реагування на інциденти та якість технічного обслуговування є показниками репутації, а отже, від роботи системи моніторингу залежить рівень довіри клієнтів і лояльність користувачів. Такі підприємства часто працюють у середовищі, де навіть хвилинна затримка чи зупинка сервера призводить до фінансових збитків, тому автоматизація контролю за станом мережі – це не розкіш, а необхідність.

Крім комерційних компаній, результати дослідження будуть актуальними для державних структур, освітніх установ і організацій, які мають внутрішні мережі та зберігають великі обсяги інформації. У таких установах впровадження системи моніторингу підвищує ефективність роботи ІТ-відділів, зменшує ризик втрати даних і допомагає раціонально використовувати наявні ресурси.

Не менш важливим є значення цієї розробки для навчальних і наукових закладів. Вони можуть використовувати систему як навчальну платформу для підготовки фахівців у сфері інформаційних технологій. Студенти отримають можливість не лише спостерігати за реальною роботою системи моніторингу, а й аналізувати дані, моделювати різні ситуації та вчитися реагувати на інциденти. Таким чином, результати дослідження мають універсальний характер і можуть бути впроваджені як у бізнесі, так і в освіті.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості програмного продукту було проведено експертне опитування серед фахівців у галузі IT-інфраструктури, адміністраторів систем і представників компаній, що мають досвід використання схожих рішень. Експертам було запропоновано оцінити систему за основними критеріями – функціональні можливості, надійність, простота впровадження, масштабованість, вартість експлуатації та потенційна економічна ефективність.

Більшість експертів високо оцінили саме інтелектуальну частину системи – можливість автоматичного сповіщення про інциденти, генерацію аналітичних звітів і прогнозування потенційних відмов обладнання. Особливо було відзначено, що система працює стабільно навіть при великому навантаженні й може адаптуватися до різних типів мережевої інфраструктури, що робить її універсальною.

За результатами оцінки середній рівень привабливості продукту склав 8,7 бала з 10 можливих. Експерти зазначили, що така система може мати великий попит серед середніх і великих підприємств, особливо якщо її вартість залишатиметься конкурентною. Також було підкреслено, що простота інтерфейсу та можливість кастомізації під конкретного користувача є суттєвими перевагами, які підвищують комерційний потенціал рішення.

Таким чином, метод експертних оцінок показав, що система має високу ринкову привабливість, відповідає актуальним потребам бізнесу та може стати успішним продуктом за умови належного маркетингового просування та підтримки користувачів.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості розробки системи Software Defined Storage доцільно використовувати витратний метод. Він передбачає визначення всіх фактичних витрат, які були понесені під час створення програмного продукту, включаючи оплату праці розробників, витрати на апаратне забезпечення, ліцензії, тестування та впровадження. Такий підхід дозволяє точно визначити базову собівартість проєкту, що є особливо важливим для невеликих команд і стартапів.

Однак, у випадку комерційного впровадження, доцільно поєднати цей підхід із дохідним методом. Дохідний метод дає змогу оцінити майбутні вигоди, які підприємство отримає після впровадження системи. Наприклад, скорочення простоїв серверів, підвищення ефективності роботи персоналу та зменшення витрат на ручну діагностику мережі є прямими джерелами економічної вигоди.

Такий комбінований підхід дозволяє не лише визначити початкову вартість розробки, а й обґрунтувати економічну доцільність проєкту. Він допомагає потенційним інвесторам побачити не просто витрати, а реальні фінансові перспективи, які відкриває впровадження системи.

У результаті використання комбінованої моделі оцінки можна отримати повну картину вартості та окупності проєкту, що стане основою для прийняття управлінських рішень щодо його реалізації чи масштабування.

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

Компанія має розгалужену IT-інфраструктуру, яка включає сервери, мережеве обладнання, робочі станції, системи зберігання даних і корпоративні сервіси. До впровадження системи моніторингу контроль за станом мережі здійснювався вручну: адміністратори виявляли проблеми лише після звернень користувачів або повного виходу сервісів із ладу. Це призводило до простоїв, затримок у роботі та фінансових втрат. Основна мета впровадження системи мережевого моніторингу – забезпечити цілодобове автоматичне відстеження стану обладнання, серверів і додатків, оперативне реагування на інциденти, зниження кількості простоїв і запобігання критичним збоям у роботі IT-інфраструктури. Вхідні дані зафіксовано в таблиці 7.1.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	До впровадження	Після впровадження	Економічний ефект
Кількість простоїв серверів на рік	20 випадків	5 випадків	–15
Середня тривалість простою одного сервера	4 години	1 година	–3 години
Середні втрати підприємства за 1 годину простою	25 000 грн	5 000 грн	–20 000 грн
Витрати на ручну діагностику й усунення збоїв	300 000 грн/рік	150 000 грн/рік	–150 000 грн
Вартість впровадження системи моніторингу	—	—	450 000 грн
Річні витрати на підтримку системи	—	—	100 000 грн

Розрахунок економічного ефекту демонструє наступне: зменшення збитків від простоїв – 1 975 000 грн/рік, економія на технічному обслуговуванні – 150 000 грн/рік, сукупний річний ефект – 2 125 000 грн/рік, чистий ефект – 2 025 000 грн/рік, термін окупності (Payback Period) – 0,22 року (~2,5 місяці), коефіцієнт ефективності (ROI) – 450%.

Додаткові (немонетарні) переваги: підвищення стабільності IT-інфраструктури завдяки ранньому виявленню збоїв, зменшення навантаження на IT-персонал через автоматизацію моніторингу, покращення SLA (Service Level Agreement) і задоволеності користувачів, прогнозування потенційних проблем через аналітику та звітність у реальному часі, зростання репутації підприємства, адже мінімізуються ризики затримок у наданні послуг або збою критичних бізнес-процесів.

Таким чином, моніторинг стає не лише технічним інструментом, а й важливою складовою операційної надійності та конкурентоспроможності підприємства.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Просування системи моніторингу має будуватися на поетапному підході, що включає як технічну демонстрацію, так і інформаційне просування. На першому етапі варто створити пілотний проєкт і запропонувати його впровадження у невеликій кількості підприємств для збору відгуків і реальних кейсів. Це дозволить перевірити ефективність системи в реальних умовах і створити довіру до продукту.

Далі важливо забезпечити інформаційну присутність продукту – через участь у галузевих конференціях, IT-форумах, онлайн-презентаціях і спеціалізованих публікаціях. Саме через публічну експертну комунікацію формується репутація розробника та усвідомлення цінності рішення на ринку.

Наступним етапом є розширення партнерських зв’язків. Доцільно співпрацювати з ІТ-компаніями, які займаються інтеграцією корпоративних систем, адже вони можуть пропонувати продукт своїм клієнтам як частину комплексного рішення. Водночас слід розробити гнучку цінову політику – наприклад, ліцензування за кількістю пристроїв або модель передплати, що зробить продукт доступнішим для малого та середнього бізнесу.

Просування має супроводжуватися технічною підтримкою користувачів, оновленнями та навчанням персоналу. Це створює позитивний досвід використання продукту та сприяє формуванню довгострокових відносин із клієнтами. У підсумку правильна стратегія просування допоможе не лише збільшити продажі, а й побудувати впізнаваний бренд на ринку ІТ-рішень.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту варто поєднати прямі продажі з цифровими платформами розповсюдження програмного забезпечення. Власний сайт компанії може стати не лише вітриною продукту, а й каналом комунікації з клієнтами, де вони зможуть отримати демо-версію, консультацію або підтримку. Це сприятиме зниженню витрат на маркетинг і збільшенню довіри.

Додатково ефективним буде впровадження партнерської програми для системних інтеграторів і реселерів, які вже мають доступ до корпоративних клієнтів. Така модель дозволяє розширити охоплення ринку без суттєвих додаткових інвестицій. Також можна запропонувати гібридну форму реалізації: ліцензування для великих компаній і модель SaaS (Software as a Service) для малого бізнесу. Це підвищить доступність системи та дозволить гнучко реагувати на потреби різних сегментів ринку.

Ключовим напрямом оптимізації збуту є створення якісного сервісу після продажу – технічна підтримка, регулярні оновлення, аналітичні звіти. Усе це забезпечує стабільність роботи клієнта й стимулює його до подальшої співпраці.

7.7 Визначення ключових факторів успіху конкретного проєкту

Основним фактором успіху є стабільність і надійність системи. Якщо система моніторингу працює без збоїв і забезпечує реальну користь, вона швидко здобуває довіру користувачів. Технологічна якість продукту, його здатність масштабуватися й інтегруватися з іншими IT-рішеннями відіграють ключову роль у його життєздатності.

Другим важливим чинником є професійна команда розробників і технічної підтримки. Клієнти цінують не лише продукт, а й можливість отримати швидку допомогу у випадку проблем або питань. Від рівня компетенції фахівців залежить не лише якість обслуговування, а й довгострокові відносини з партнерами.

Не менш значущим є гнучкість системи – можливість адаптувати її під специфіку кожного клієнта. Різні компанії мають різну інфраструктуру, тому універсальне, але налаштовуване рішення стає перевагою.

І, нарешті, успіх будь-якого IT-проєкту визначається здатністю постійно вдосконалюватися. Регулярні оновлення, впровадження нових технологій і зворотний зв’язок із користувачами формують довіру й підтримують актуальність продукту на ринку. Саме ці чинники разом створюють основу для стабільного розвитку та комерційного успіху системи моніторингу.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Згідно закону України “Про охорону праці” [3] кожна компанія впроваджує заходи з охорони праці. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

– Інструкцій з охорони праці по кожній професії і загальні.

– Положення про охорону праці.

– Накази з охорони праці.

– Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Законом України “Про охорону праці” [3] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми

роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робітників галузі інформаційних технологій (ІТ), фахівці відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [5], та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп’ютера на організм людини визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року.

Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 8.2.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

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

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину
Об'єм до 20 $м^3$ на людину	Не менше 30
20... 40 $м^3$ на людину	Не менше 20
Більше 40 $м^3$на людину	Може біти використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно впливає на продуктивність праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів –

малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [4]

8.3 Пропозиції щодо підвищення працездатності ІТ-фахівців

Поява та впровадження нових інформаційно-комунікаційних технологій зумовлює необхідність подальшого вдосконалення охорони праці фахівців ІТ-індустрії. Все це потребує розробки нових нормативно-правових актів з регламентації праці та відпочинку фахівців ІТ-індустрії і стандартів підприємств, центрів комп'ютерної техніки, центрів інформаційних технологій, сучасних комп'ютерних класів.

Для підвищення розумової працездатності то зорової роботи повинна здійснюватися ергономічна оптимізація в рамках системи «оператор-термінал», яка сприятиме результативній фізичній та інтелектуальний працездатності і відновленню психосоматичного здоров'я фахівців ІТ-індустрії. Всі наведені заходи щодо вдосконалення охорони праці фахівців ІТ-індустрії повинні

контролюватися службою охорони праці та комісією з охорони праці підприємства.

Особливе значення у соціальному захисті цієї категорії працівників належить прийняття комплексного договору, який може забезпечити фахівців додатковими пільгами та компенсаціями.

Пропозиції щодо підвищення працездатності ІТ-фахівців, розіб'ємо на декілька категорій:

1. Середовище і розпорядок праці.

Для мінімізації негативних ефектів, що пов'язані з перевтомленням ІТ-фахівців, потрібно чітко прописати і реалізувати графік періодів праці-відпочинку, щоб фахівець міг можливість переключити увагу, дати можливість відпочити очам, мозку, елементарно, встати розім'яти ноги.

Також потрібно зробити максимально комфортними умови мікроклімату у офісному приміщенні, де працюють ІТ-фахівці. Мається на увазі встановлення і експлуатація, коли виникає необхідність, кондиціонерів, опалення, та системи вентиляції, задля попередження перегрівання, переохолодження ІТ-фахівців, і подальшої неможливості ними виконувати свої функції.

Також, за можливості, нами пропонується введення практики віддаленої праці ІТ-фахівцями, якщо роботодавець не може забезпечити оптимальні і безпечні умови в офісному приміщенні, або якщо фахівця вони не влаштовують із певних причин.

2. Фізичні і психоемоційні чинники.

Першим і найважливішим чинником, що впливає на працездатність ІТ-фахівців є робоче місце, і саме тому, роботодавець має забезпечити максимальний його комфорт і безпеку.

Гарантією цих факторів може слугувати сертифікація меблів, що використовуються на підприємстві ІТ-галузі. Тому нами пропонується закупівля тільки меблів, які пройшли сертифікацію на відповідність. Під психоемоційними чинниками ми розуміємо гарне самопочуття фахівців, позитивний настій, гарний

психологічний клімат у колективі, тощо. Задля того, щоб психоемоційні чинники мали максимально позитивний ефект, керівництву слід поводити заходи, які сприятимуть укріпленню і покращенню міжособистісних стосунків у колективі, таких як психологічні тренінги, таймбілдінг, спортивні змагання і естафети.

Також, сюди можна віднести розробку і впровадження системи мотивації працівників, як фінансової, так моральної і адміністративної.

8.4 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток 50·50·5 мм., довжиною L=3 м., та горизонтальний електрод – металева полоса з перетином 50·5 мм. Напруга – 220/380 В. Розрахункова схема розташування заземлюючих електродів – по контуру (прямокутником).

Розрахунок проведемо за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару ґрунту – чорнозем, нижнього шару ґрунту – глина (питомий опір ρ_2 = 40 Ом·м). Умовна товщина верхнього шару ґрунту: H=0,6 м. Відстань між вертикальними заземлювачами (електродами) A=3 м. Глибина закладення горизонтального контура заземлення t=0,75 м. Опір заземлювача, який нормується: $R_{ЗН}$ = 4 Ом. Необхідно визначити необхідну кількість вертикальних заземлювачів та довжину полоси (горизонтального заземлювача) (рис. 8.1).

Виконуємо розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T = t + L/2 = 0,75 + 3/2 = 2,25 \text{ м.}$$

Розрахунковий питомий опір ґрунту (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі ґрунту):

$$\rho = \psi \rho = 1,36 \cdot 40 = 54,5 \text{ Ом·м.}$$

де ψ = 1,36 – табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому ґрунті [11];

ρ_2= 40 Ом·м – табличне значення питомого опору нижнього шару ґрунту (глина) [11].

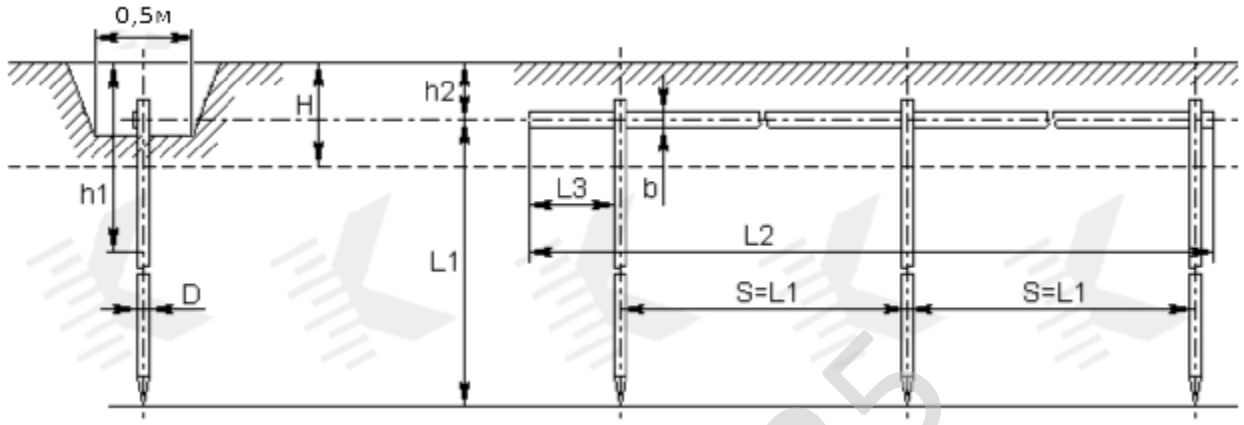

Рисунок 8.1 – Схема штучного заземлення

Еквівалентний діаметр вертикального електрода (кутка) [11]:

$$D_в=0,95\cdot К= 0,95\cdot 50 = 47,5 \text{ мм.} =0,0475 \text{ м.}$$

де К= 50 мм – розмір металевого кутка (задано).

Відношення A/L=3/3=1.

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [11]:

$$R_O=0,366\cdot(\rho/L)\cdot[\lg(2L/D_в)+(1/2)\cdot\lg((4T+L)/(4T-L))]=$$

$$=0,366\cdot(54,5/3)\cdot[(\lg(2\cdot3/0,0475)+(1/2)\cdot\lg((4\cdot2,25+3)/(4\cdot2,25-3))]=14,9 \text{ Ом.}$$

Визначаємо коефіцієнт екранування вертикальних електродів Кев=0,53 при орієнтовній кількості вертикальних електродів, яке дорівнює 5 [11].

Визначаємо необхідну кількість вертикальних електродів заземлювача (без врахування горизонтального заземлювача), при $R_{ЗН}$ = 4 Ом:

$$N=R_O /(К_{ев} R_{ЗН})= 14,9 / (0,53\cdot4)= 7,05 \approx 7 \text{ шт.}$$

Визначаємо довжину з’єднуючої полоси:

$$L_П = 1{,}05 \cdot A \cdot N = 1{,}05 \cdot 3 \cdot 7 = 18{,}5 \approx 18 \text{ м.}$$

Опір розтіканню електричного струму з’єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору ґрунту $K_П$ [11]:

$$R_П = 0{,}366(\rho \cdot K_П / L_П) \lg(2 \cdot L_П^2 / (B \cdot t)) =$$

$$= 0{,}366(40 \cdot 5/18) \cdot \lg((2 \cdot 18^2)/(0{,}05 \cdot 0{,}75)) = 20{,}1 \text{ Ом.}$$

де $K_П$=5 – табличне значення кліматичного коефіцієнта питомого опору ґрунту для відповідної кліматичної зони для з’єднуючої полоси [11]:

B = 50 мм = 0,05 м.- ширина з’єднуючої полоси (задана).

Загальний опір розтіканню електричного струму заземлювача [11]:

$$R = (R_O \cdot R_П)/(R_O \cdot \eta_П + N \cdot R_П \cdot Кев) =$$

$$= (14{,}9 \cdot 20{,}1)/(14{,}9 \cdot 0{,}55 + 7 \cdot 20{,}1 \cdot 0{,}53) = 3{,}56 \text{ Ом.}$$

де $\eta_П$ = 0,55 – табличне значення коефіцієнта екранування з’єднуючої полоси [11].

Умова $R \leq R_{ЗН}$ виконується (3,56 ≤ 4).

Остаточно кількість вертикальних електродів дорівнює 7.

За потреби можна зменшити кількість електродів заземлювача, зменшивши загальний опір розтіканню електричного струму заземлювача методом зменшення питомого опору ґрунту, домішуючи у ґрунт безпосередньо навколо електродів заземлювача розчини солей NaCl, CaCl, сажу, соду, шлак або спеціальні суміші.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров’я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи.

Тільки повна усвідомленість працівника про можливі небезпеки, що можуть підстерігати його на робочому місці та дотримання вимог нормативних актів з питань охорони праці та відповідних рекомендацій фахівців, дозволять значною мірою знизити негативний вплив шкідливих та небезпечних факторів при роботі з комп’ютером на організм людини.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи Software Defined Storage.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів Software Defined Storage.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем Software Defined Storage.

– Досліджена система Software Defined Storage.

– На основі отриманих результатів досліджень створена програмна реалізація системи Software Defined Storage.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання Software Defined Storage.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Threefish.

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

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Молчанов Ю.В. Дослідження та програмна реалізація системи Software Defined Storage // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

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

3. 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.

4. 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.

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

6. 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.

7. 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.

8. 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.

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

10. 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.

11. 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.

12. 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.

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

14. 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.

15. 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.

16. 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.

17. 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.

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

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». *International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019*; Odessa; Ukraine; 9-13 September 2019. P.22-28.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

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

35. 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

36. 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.

37. 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.

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

39. 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.

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

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

42. 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).

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

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

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

46. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-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.

47. 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.

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

49. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв 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.

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

51. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв 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.

52. 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.

53. 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.

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

55. Смірнов О.А., Кавун С.В., Коваленко О.В., Доренський О.П., Дрєєв О.М., Вялкова В.І. Комп'ютерні мережі. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 233 с.