МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Вєтров Андрій Олексійович

Програмне забезпечення системи SDS побудованої з використанням RDDA

Спеціальність: 123 «Комп'ютерна інженерія»

Освітній ступінь: бакалавр

Науковий керівник:

Коваленко Олександр Володимирович_____

(підпис) (дата)

доктор технічних наук, доцент

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« 11 » січня 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Вєтрову Андрію Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи SDS побудованої з використанням RDDA*

керівник роботи *Коваленко Олександр Володимирович, докт. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 204-02 від 28.12.2020 року

2. Строк подання студентом роботи до захисту *22.05.2021 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи SDS побудованої з використанням RDDA*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « 11 » січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2021 р.	
3.	Розробка моделі компонента	20.03.2021 р.	
4.	Розробка структур даних	25.03.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2021 р.	
6.	Програмування алгоритмів	10.04.2021 р.	
7.	Оформлення ПЗ	17.04.2021 р.	
8.	Попередній захист роботи	14.05.2021 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Вєтров А.О. Програмне забезпечення системи SDS побудованої з використанням RDDA. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній бакалаврській розроблено програмне забезпечення, яке призначено для системи SDS побудованої з використанням RDDA.

Метою розробки є програмне забезпечення системи SDS побудованої з використанням RDDA.

Результат роботи – програмна реалізація системи SDS побудованої з використанням RDDA.

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 архітектури IBM PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено в середовищі Visual C#.

Ключові слова: комп'ютерна інженерія, SDS, RDDA

ABSTRACT

Vietrov A.O. SDS system software built using RDDA. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this bachelor's qualification the software which is intended for the SDS system constructed with use of RDDA is developed.

The purpose of the development is the software of the SDS system built using RDDA.

The result is a software implementation of the SDS system built using RDDA.

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.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in Visual C # environment.

Keywords: computer engineering, SDS, RDDA

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ОС	–	Операційна система
ПЗ	–	Програмне забезпечення
СЗД	–	системи зберігання даних
BIOS	–	Basic Input-Output System
FcoE	–	Fibre Channel over Ethernet
LINQ	–	Language Integrated Query
SDS	–	програмно-обумовлені сховища, Software-Defined Storage
RAID	–	Redundant Array of Inexpensive Disks
RDDA	–	Метод віддаленого прямого доступу до дисків, Remote Direct Drive Access

ВСТУП

Актуальність теми. Потреби обробки й зберігання величезних масивів даних привели до істотної зміни ринкового й технологічного ландшафтів, а хвиля нововведень, внесених у системи зберігання, стала причиною змін у суміжних сегментах. Незважаючи на всі часом революційні перетворення, головні ще спереду.

Двома найбільш помітними тенденціями останніх років в області зберігання є перехід на SSD і програмно-обумовлені сховища (Software-Defined Storage, SDS). Незважаючи на схожість абревіатур, це зовсім різні поняття, хоча друга технологія, програмно-обумовлені сховища, стала можливою багато в чому завдяки першій – флеш-накопичувачам.

Однак уся різноманітність потреб неможливо задовольнити за допомогою тільки двох нових технологій, тому традиційні системи й рішення продовжують розвиватися, не залучаючи, втім, такої ж ажіотажної уваги. Більше того, вони часом одержують друге життя, як це трапилося з оптичними дисками: носії останнього покоління здатні зберігати дані протягом 100 років, а в досвідчених перспективних зразків прогнозований термін служби становить 600 років!

Хвиля змін у системах зберігання породжує зміни й у суміжних сегментах. Насамперед це стосується забезпечення необхідних характеристик при доступі до даних по мережі. Незважаючи на невдачу технології Fibre Channel over Ethernet (FcoE), з появою протоколу NVMe і його мережного варіанта NVMe over Fabric прихильники перекладу мережі на єдиний протокол – таким, звичайно ж, повинен бути Ethernet – знову воспряли духом. Правда, найбільш обережні з них погоджуються з тим, що для підключення систем зберігання краще використовувати окрему мережу.

Незважаючи на всі ці часом революційні перетворення, кардинальні зміни ще спереду. І зв'язані вони з тим, що ефективний рішення завдань і оптимальний

використання ресурсів можливі лише в тому випадку, якщо обчислювальні потужності, ємності зберігання й мережні ресурси будуть об'єднані. Усвідомлення цього факту привело до сплеску популярності гіперконвергентних рішень. Однак на підході вже наступна архітектура – дезагрегована, або компонуєма, інфраструктура.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи SDS побудованої з використанням RDDA.

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

– Огляд існуючих систем SDS побудованої з використанням RDDA.

– Дослідження системи SDS побудованої з використанням RDDA.

– Програмна реалізація системи SDS побудованої з використанням RDDA.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі SDS побудованої з використанням RDDA.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи SDS побудованої з використанням RDDA, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

У цій роботі розглянемо програмно-обумовлені сховища (Software-Defined Storage, SDS) і про можливості їх застосування, які вони дають при побудові IT-інфраструктури.

У великих організаціях системи зберігання даних займають значну частку вартості IT-інфраструктури (по оцінках фахівців – до 25%). Ця цифра може суттєво вирости. Причини – ріст обсягу даних і збільшення потреби в ємностях систем зберігання даних (СЗД), у тому числі через закони, які зобов'язують ці дані зберігати. У той же час компанії активно намагаються заощаджувати IT-Бюджети, що змушує їх перебувати в постійному пошуку найбільш вигідних технологічних рішень, які б дозволили скоротити ці витрати не на шкоду якості сервісу. Це ж ставиться до зберігання й обробці даних.

Вимоги замовників до зниження вартості володіння IT-інфраструктурою змушують постачальників інвестувати в розробки й пропонувати нові технології. Одна з них – програмно-обумовлені системи зберігання даних (Software-Defined Storage, SDS). Компанії починають замислюватися про впровадження SDS, коли процедури роботи з даними стають неефективними і їх пошук забирає багато часу.

Концепція SDS дозволяє одержати такі переваги, як:

- абстрагування від нижнього рівня (апаратній платформи);
- масштабованість;
- спрощена інфраструктура зберігання;
- низька вартість рішень.

Завдяки технологіям SDS можна значно знизити вартість СЗД і їх адміністрування. За прогнозами Gartner, до 2022 року 70-80% неструктурованих

даних будуть зберігатися на недорогих системах, керованих за допомогою SDS, а вже до 2023 року 70% існуючих масивів зберігання стануть доступні в повністю програмній версії.

1.2 Область застосування

Коли й навіщо потрібна SDS

ПЗ керування СЗД повинне забезпечувати гнучку організацію зберігання даних, а також:

- дедупликацію;
- реплікацію даних;
- динамічне виділення ємності;
- знімки даних;
- дотримання політик зберігання.

SDS визначають в Storage Networking Industry Association (SNIA, Асоціація виробників і споживачів систем зберігання) як віртуалізоване середовище зберігання даних з інтерфейсом керування сервісами, яка повинна містити в собі:

- автоматизацію – спрощене керування, що знижує витрати на обслуговування інфраструктури зберігання даних;
- стандартні інтерфейси – API для керування, виділення й звільнення ресурсів, обслуговування сервісів і пристроїв зберігання;
- віртуалізацію шляхів доступу до даних – блоковий, об'єктний і файловий доступ відповідно до інтерфейсів застосунків;
- масштабованість – зміна інфраструктури зберігання без зниження необхідного рівня доступності або продуктивності;
- прозорість – моніторинг споживаних ресурсів зберігання, керування ними й контроль їх вартості.

Відзначу, що для SDS потрібний стандартизований інтерфейс керування – такий, як SNIA Storage Management Initiative Specification (SMI-S). Він є складовою частиною концепції програмно-обумовлених дата-центрів (SDDC). Ця програмна логіка хмарної інфраструктури зберігання й хмарних апаратних платформ може бути елементом і традиційних ЦОД. Сервіси зберігання й обробки даних можуть виконуватися на серверах, спеціалізованих пристроях зберігання (storage appliance) або на обох цих платформах, усуваючи традиційні границі.

Технологія SDS почала розвиватися ще на початку 2000-х, але поки не змогла замінити класичні СЗД із цілого ряду причин – зараз ми їх обговорювати не будемо. Але виробники активно займаються розвитком своїх продуктів і інтерес до технологій SDS росте. За нашими оцінками, найближчим часом вони стануть тем інструментом, який дозволить скорочувати вартість IT-інфраструктури при росту потреби в збільшенні ємності СЗД.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи SDS побудованої з використанням RDDA, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Порівнюємо SDS-рішення

Software-Defined Storage пропонують багато вендорів:

- Dell EMC (рішення Dell Nexenta, EMC Scaleio);
- HPE (рішення Storevirtual VSA);
- IBM (рішення Spectrum Storage);
- Netapp (рішення ONTAP Select);
- Vmware (рішення vsan);
- Red Hat (рішення Red Hat Storage);
- Stonefly (рішення SCVM, SDUS);
- Datacore (рішення SANsymphony);
- Swiftstack;
- Pivot3 і ін.

Уточню, що рішення Redhat Storage представлено двома продуктами: Redhat Ceph Storage і Redhat Gluster Storage (RH Storage Server). Тут вони маються на увазі обоє, але в наведеному нижче порівнянні вони не брали участь, тому що значно відрізняються від інших згаданих рішень.

Ceph – не зовсім коробковий продукт. Його використання без штату розроблювачів досить важко, що зробило його нецікавим для нашої компанії. Тому цього рішення немає в порівняльній таблиці.

Умовно всі SDS-рішення можна розділити на три категорії:

- класичні (CEPH, Red Hat Storage Server, EMC Scaleio),
- на основі традиційних систем зберігання (Netapp ONTAP Select, HPE Storevirtual VSA),

– у складі обчислювальних комплексів (Vmware vsan).

Деякі виробники пропонують як комплексні рішення, так і програмну частину (Huawei, Dell EMC). Це дозволяє гнучко підходити уводити, увести до ладу добору продуктів і використовувати успадковане «обчислювальне» устаткування для рішення менш ресурсномістких завдань зберігання даних. Ще однією заслугою SDS стала можливість застосування в деяких класичних СЗД віртуалізації дискових масивів.

Рішення архітектурно будуються по двом принципам:

– слабко зв'язані;

– розподілені (без загальних елементів).

У першому випадку відказостійкість забезпечується за рахунок розподілених копій даних, але через надмірність комунікацій між вузлами (нодами) знижується швидкість запису. Критичним місцем є мережу передачі даних, тому такі рішення звичайно реалізовані на основі Infiniband. По такому принципу побудовані рішення Vmware vsan, HPE Storevirtual VSA, Dell EMC Scaleio.

У системах без загальних елементів дані записуються на один вузол, а потім із заданою періодичністю копіюються на інші для забезпечення відказостійкості. При цьому записи не є транзакційними. Такий підхід найбільш дешевий. Найчастіше в якості інтерконекту в ньому використовується Ethernet. Дана архітектура зручна з погляду масштабованості. Яскравий її представник – CEPH.

Зараз багато компаній займаються розробкою як програмної SDS (наприклад, Atlantis Computing, Maxta, Starwind, Datacore Software, Sanbolic, Nexenta, Cloudbyte), так і випуском комплексних рішень (Dell EMC, IBM) або спеціалізованих пристроїв (Tintri, Nimble, Solidfire).

Компонуєма інфраструктура від Western Digital

Western Digital пропонує власний підхід до реалізації дезагрегованої, або, як її називають слідом за HPE, компонуємої, архітектури.

Дезагрегована архітектура уможливлює створення необхідної апаратної конфігурації з доступного пулу різних ресурсів: процесорів, пам'яті, жорстких дисків, мережних карт і т.д. Зміст дезагрегації в підвищенні ступеня утилізації ресурсів і гнучкості їх надання. У компонуємої архітектурі можна буде використовувати ресурси не на 40-45%, як зараз, а на 80-90%.

В Western Digital роблять ставку на відкритий підхід (зокрема, компанія планує протягом декількох років перевести всю свою продукцію на сервери RISC-V). Крім відкритості, при створенні компонуємої архітектури Openflex розроблювачі прагнули забезпечити масштабованість (різні види ресурсів повинні нарощуватися незалежно), дезагрегацію (незалежність усіх компонентів на апаратному рівні) і універсальність (підтримка всіх можливих типів процесорів і накопичувачів).

Openflex базується на концепції мережних пристроїв з підтримкою NVMe-oF. Для реалізації компонуємої архітектури, їх необхідно доповнити обчислювальними потужностями, підключивши окремий пул процесорів з пам'яттю й мережними картами, але без яких-небудь дисків SSD/HDD системи SDS (насправді така архітектура є лише частково дезагрегованою, тому що повна дезагрегація припускає незалежність процесорів і пам'яті).

Необхідна пропорція ресурсів – процесорів, флеш-накопичувачів і жорстких дисків SSD/HDD системи SDS – підбирається відповідно до поставлених завдань. Інакше кажучи, кожний віртуальний сервер збирається з наявного пулу ресурсів (при цьому компонування здійснюється на апаратному рівні – без використання гіпервізора).

Western Digital випустила дві системи зберігання для реалізації такої архітектури, причому в них підтримуються не тільки флеш-накопичувачі, але й жорсткі диски. Це шасі для фабрики даних Openflex E3000 Fabric Enclosure і масив жорстких дисків SSD/HDD системи SDS Openflex D3000 Fabric Device. У шасі висотою 3U може бути встановлено десять модулів флеш-накопичувачів E3000 ємністю 61 Тбайт із двома інтерфейсами по 50 Гбіт/с кожний з підтримкою

RoCE для реалізації NVMe-oF (див. рис. 2.1). Масив D3000 – досить незвичайний рішення, фактично це дискова полиця на 13 SSD/HDD з підтримкою двох NVMe-oF-сумісних інтерфейсів на 25 Гбіт/с. Користувачі сприймають систему як один NVMe-Диск ємністю 168 Тбайт.

Рисунок 2.1 – СЗД Aeon S200-RX M2 компанії Netberg працює під керуванням ПЗ Raidix 4.4. У цій системі, що має відказостійку апаратну архітектуру, застосовуються патентовані алгоритми RAID

Для керування Openflex компанія пропонує відкритий Kingfish API, що одержав свою назву за аналогією з API Redfish для керування фізичними серверами й Swordfish для керування логічними сховищами. Створити широку екосистему навколо Kingfish поки не вдалося, але ідею підтримала компанія HPE, що активно займається просуванням компонуємої інфраструктури. WD працює над оркестратором, за допомогою якого з дезагрегованих ресурсів можна буде збирати сервери на вимогу.

У підсумку ми прагнемо прийти до того, щоб усередині ЦОД можна було одержати у два кличі фізичну маши ну для конкретних завдань – точно так само, як замовляються віртуальні машини із хмари.

Фабрика зберігання від Mellanox

Цього року компанія Mellanox була вперше включена аналітиками Gartner у магічний квадрант постачальників рішень для мереж центрів обробки даних.

Одним з аргументів на користь такого вибору стала запропонована нею концепція мереж зберігання Ethernet (Ethernet Storage Fabric, ESF).

Fibre Channel – несучасний рішення. Це досить складна мережа, експлуатація якої вимагає спеціальних компетенцій. Крім того, перехід з одного покоління Fibre Channel на інше з метою підвищення швидкості обходиться дуже дорого. Тим часом Ethernet уже сьогодні забезпечує підтримку 100 Гбіт/с проти 32 Гбіт/с в Fibre Channel.

Концепція ECF припускає можливість застосування ряду технологій для оптимізації використання Ethernet у якості виділеної мережі зберігання – зокрема, NVMe-oF, коли мережа Ethernet виконує роль фабрики. Для цього в мережних картах реалізуються функціональність RDMA поверх конвергентного Ethernet (RDMA over Converged Ethernet, RoCE) і розвантаження цільових пристроїв NVMe-oF і операцій кодування. Для спрощення конфігурування мережі з підтримкою RoCE компанія пропонує безкоштовне програмне забезпечення NEO.

Рисунок 2.2 – Фабрика зберігання Ethernet, у якій використовуються комутатори Mellanox Spectrum, здатна забезпечити передбачувану продуктивність і мінімальні, 1 мкс, затримки в мережі

Сучасним мережам зберігання пропускної здатності 10 Гбіт/с явно недостатньо – один накопичувач NVMe може видавати дані зі швидкістю 20–25 Гбіт/с. Відповідно, чотири накопичувачі в сервері можуть зайняти смугу в 100 Гбіт/с. Мережне встаткування Ethernet компанії Mellanox дозволяє будувати мережі із пропускною здатністю 100 Гбіт/с, при цьому така мережа обійдеться дешевше, чим мережа Fibre Channel на 32 Гбіт/с. В I-II кварталах наступного року цей виробник збирається випустити комутатори з підтримкою 200 і 400 Гбіт/с. Сучасні ЦОД усе частіше відмовляються від модульних шасі на користь побудови мережних фабрик з використанням стійкових комутаторів форм-фактора 1U.

На відміну від багатьох інших вендорів, у своїх комутаторах Spectrum і мережних платах Connectx компанія використовує мікросхеми власної розробки. Завдяки цьому встаткування Mellanox забезпечує на порядок меншу затримку (як затверджується, у фабриці із трьома транзитними вузлами затримка становить усього 1 мкс (див. рис. 2.2)). Крім того, втрата пакетів виключається навіть при максимальному навантаженні.

Для комутаторів Mellanox передбачена можливість вибору операційної системи. Крім власної «закритої» ОС, вони можуть працювати під керуванням Cumulus Linux, а також будь-якої стандартної версії Linux з ядром четвертої галузей. Зокрема, можна встановити дистрибутив Alt Linux компанії «Базальт».

Panasonic

Panasonic займається технологіями оптичних дисків SSD/HDD системи SDS і приводів протягом останніх 30 років, причому компанія розробляє всі необхідні мікросхеми й механічні елементи. У співробітництві з Facebook створена система архівації даних на оптичних дисках – freezeray. У ній використовується новий формат архівних дисків SSD/HDD системи SDS замість BluRay, який був розроблений Panasonic разом з Sony. У цей час на один диск міститься 300 Гбайт, незабаром повинні з'явитися накопичувачі ємністю 500 Гбайт, у планах – ємність 1 Тбайт. Розрахунковий термін служби Archival Disc

становить 100 років. Для підвищення надійності зберігання, 12 дисків SSD/HDD системи SDS у касеті об'єднані в масив RAID.

Компанія випустила програмне забезпечення Raidix Archival Solution for Panasonic (RASP) для віртуалізації ресурсів оптичної бібліотеки. ПАК на основі freeze-ray і RASP дозволяє забезпечити зберігання «гарячих» і «холодних» даних в одній гібридній інфраструктурі. Для кешування даних з архіву пропонується файлова система Raspfs.

Учені з Австралії й Китаю продемонстрували прототип оптичного диска з потенційною ємністю 10 Тбайт і терміном служби 600 років. Стекло – зносостійкий матеріал, який може зберігатися більш 1000 років, але ємність зберігання такого диска обмежена. Для її збільшення був використаний нетрадиційний матеріал – гібридний скляний композит (комбінація скла з органікою) з інкорпорованими золотими наностержнями.

Netberg

Компанія Netberg з 2015 року розвиває власні лінійки серверів, систем зберігання й мережного встаткування, при цьому розробка й виробництво ведуться на Тайвані.

Обчислювальна платформа Netberg реалізована відповідно до концепції «кластер у коробці» (Cluster in Box, Cib): в одному шасі розміщаються два сервери із загальною об'єднавчою панеллю. На базі цієї платформи можуть бути розгорнуті як серверні кластери, так і системи зберігання даних. Проводиться аналогія з популярними гіперконвергентними рішеньи, коли в одній архітектурі поєднуються й сервери, і системи зберігання.

Відповідно до концепції Cib реалізована, наприклад, нова система зберігання Netberg Aeon S200-RX M2, яка працює під керуванням програмного забезпечення Raidix 4.4 (базується на серверній платформі Demos R420 M2). Залежно від конфігурації в неї може бути встановлено 24 диска 2,5" або 12 дисків SSD/HDD системи SDS 3,5". При необхідності систему можна розширити до 560

дисків SSD/HDD системи SDS за допомогою дискових полиць серії Aeon RX (див. рис. 2.3).

Рисунок 2.3 – Шасі Openflex E3000 Fabric Enclosure розроблялося компанією Western Digital розраховуючи на використання в рамках компонуємої архітектури

Дана компактна система (висота 2U) дозволяє надати можливості програмно-обумовленого сховища корпоративного класу невеликим і середнім компаніям. Серед таких можливостей можна виділити модуль розпізнавання застосунків Qosmic, за допомогою якого критичним додаткам гарантується необхідний рівень продуктивності при роботі зі СЗД. Для керування системою передбачений порт IPMI, що не так часто зустрічається в розв'язках даного класу.

Торік Netberg випустила дискову полицю Netberg Aeon J380 NVMe висотою 3U, яка являє собою набір флеш -накопичувачів (Just Bunch of Flash, JBOF) – 80 дисків SSD/HDD системи SDS NVMe з інтерфейсом U.2. Усі вони можуть бути підключені до одному серверу, або їх можна розділити між п'ятьма серверами за допомогою п'яти комутаційних модулів Pcie, оснащених контролерами PMC. Дана система призначена для проектів, де пред'являються високі вимоги до продуктивності дискової системи.

Fibre Channel – віджила технологія, хоча компанія й підтримує її у своїх серверах і СЗД. Крім більш високої номінальної пропускної здатності, що досягається завдяки таким технологіям, як Data Center Bridging (DCB), сучасні комутатори Ethernet здатні надати багато можливостей, властиві FC, зокрема передачу трафіка без втрат. При цьому вони коштують набагато дешевше. Так, комутатор Netberg 620 с 48 портами 25Gbe і 6 портами для каскадування продається за ціною 9000 доларів (комутатор FC з таким же кількістю портів на 16 Гбіт/с обійдеться в три рази дорожче).

Із самого початку в розробці мережного встаткування була зроблена ставка на підтримку високих швидкостей: перший 100-гигабитний комутатор Aurora 720 був випущений в 2015 році. Ці комутатори використовуються в Інституті Макса Планка для передачі даних з радіотелескопів. Основний попит на таке встаткування поки зосереджений за рубежем. Якийсь час назад компанії навіть довівся відновити виробництво моделі комутатора 10/40, оскільки ринок не був готовий до швидкостей 25/100 Гбіт/с (хоча за ціною встаткування мале відрізнялося).

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 – крапку входу застосунка – інкапсулюється у визначення класів. Клас може успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які п еревизначають 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 додаток до основних описаних об'єктно-орієнтованих принципів, мова Visual C# спрощує розробку компонентів програмного забезпечення завдяки декільком інноваційним конструкціям мови, у число яких входять наступні:

– Інкапсульовані підписи методів, називані делегатами, які підтримують строго-типізовані повідомлення про події.

– Властивості, що виступають у ролі методів доступу для закритих

змінних-членів.

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

– LINQ (Language-Integrated Query), що пропонує вбудовані можливості запитів у різних джерелах даних.

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

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

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. Код, виконуваний середовищем CLR, іноді називають "керованим кодом" у протиставлення "некерованому коду", що компілюється в машинний код, призначений для певної системи. Далі показані відносини під час компіляції й час виконання між файлами з вихідним кодом Visual C#, бібліотеками класів .NET Framework, складаннями й середовищем CLR.

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

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи SDS

побудованої з використанням RDDA.

В процесі розробки кваліфікаційної бакалаврської роботи 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

Сьогодні багато компаній шукають шляхи для заміни колишніх систем зберігання даних (СЗД) на нові програмно-обумовлені рішення (Software-Defined Storage, SDS). Головна причина цього процесу – бажання позбутися залежності, пов'язаної з використанням устаткування одного вендору, гнучко нарощувати (або скорочувати) застосовувані ресурси, вирішувати нові функціональні завдання, а в підсумку добитися скорочення експлуатаційних витрат при росту ефективності використання обчислювальних потужностей, опираючись на можливості корпоративного ЦОДа й публічної хмари.

Незважаючи на очікуваний ріст ринку ще не сформувалася остаточна думка, у якому виді будуть існувати SDS-системи, яке встаткування з нині використовуваного все-таки прийде заміняти в першу чергу.

Для SDS характерно те, що їх програмна частина відділена від апаратної начинки. В IDC уважають, що це приведе в підсумку до формування абсолютно автономного програмного стека, який буде повністю відповідати за рішення будь-яких функціональних завдань СЗД на вже наявному встаткуванні.

Спробу сформувати список функцій, якими повинні мати SDS-системи, недавно здійснила галузева некомерційна асоціація SNIA (Storage Networking Industry Association). На її думку, у ньому повинні бути відбиті необхідні засоби автоматизації, стандартні інтерфейси, передбачені засоби для масштабування й забезпечення прозорості відносно виконуваних операцій. Останнє має особливе значення для користувачів, тому що завдяки цим засобам вони зможуть здійснювати моніторинг і керування сховищем, враховуючи його апаратні можливості й контролюючи майбутні витрати.

Вендори SDS-рішень, однак, бачать, що відбувається в іншому світлі. Вони вибирають між реалізацією блокового, об'єктного й файлового доступу до даних, вивчають можливість побудови гіперконвергентних рішень, що забезпечують завдяки модульності гнучкі можливості по вибудовуванню ефективних конфігурацій устаткування, простоту й легкість його масштабування.

Традиційні системи зберігання

Серед традиційних корпоративних систем зберігання даних найпоширеніші рішення двох типів – мережні пристрої зберігання (Network Attached Storage, NAS) і мережі зберігання даних (Storage Area Networks, SANs).

NAS вибудовуються з окремих пристроїв, що працюють із файлами. Передача даних здійснюється по локальній мережі з використанням Ethernet.

SAN працює інакше. Це – зв'язна мережна інфраструктура, вибудувана із пристроїв зберігання даних, які працюють як дискові масиви й реалізує метод блокового доступу до даних, що зберігаються. Для передачі даних між такими пристроями часто застосовується високопродуктивний волоконно-оптичний канал зв'язку.

Вибудовування традиційних систем зберігання завжди носило чітко виражений прикладний характер. Було потрібно забезпечити певний набір функцій і продуктивність, інших застосувань, крім закладених на етапі проектування таких систем, не передбачалося.

Це цілком улаштовувало замовників доти, поки не з'явилися хмарні обчислення й програмно-обумовлені рішення. Разом з ними прийшли нові вимоги при роботі з даними, з якими колишні традиційні системи справлялися вже не так ефективно, як хотілося замовникам.

У нових умовах важливий вплив на розвиток СЗД стали виявляти два фактори: можливість нарощування обсягів хмарних обчислень і реалізація об'єктної моделі зберігання даних. Це стало причиною для росту популярності SDS.

Як приклад ефективної реалізації нового підходу часто називають хмарний сервіс Amazon Web Services (AWS). Він являє собою платформу, яка поєднує публічну систему хмарних обчислень із об'єктним сховищем даних Amazon Simple Storage Service (Amazon S3). компанія запропонувала недорогий і збалансований рішення, що дозволяє витягати переваги з об'єктного принципу зберігання, часто навіть не усвідомлюючи, як працює ця технологія.

Об'єктне зберігання даних

На відміну від традиційних систем зберігання, де збережена інформація розглядається у вигляді набору файлів або блоків даних, об'єктний підхід припускає керування даними як самостійними сутностями (об'єктами). Цей спосіб адресації застосовується при роботі із хмарою.

Гідність об'єктного підходу взаємодії з даними полягає в тому, що він не вимагає наявності заздалегідь створеної й постійно підтримуваної в актуальному стані системи впорядкованої ієрархії для роботи даними. Немає необхідності вишиковувати деревоподібні інформаційні структури, як при роботі з файловими сховищами. Процес добування даних стає більш ефективним завдяки відсутності попередньої роботи зі складними за структурою системами каталогів.

Об'єктна система зберігання розглядається сьогодні як ідеальної варіант для роботи з неструктурованою інформацією (наприклад, медіа-файлами або Web-Контентом).

AWS – це не єдиний великий представник об'єктної моделі зберігання інформації. Власні версії об'єктного зберігання в публічній хмарі пропонують сьогодні, наприклад, Microsoft Azure і Google.

Багато вендорів, відомі своїми продуктами традиційного типу, також додали опції для побудови об'єктних сховищ.

Перший приклад – це IBM, яка в 2015 р. придбала стартап Cleversafe з його моделлю об'єктного сховища. У результаті зовсім скоро на ринку з'явився рішення IBM Cloud Object Storage, вибудуваної на об'єктній технології зберігання даних.

Наступний вендор, пішовший цим шляхом – Hitachi Data Systems. Свою присутність компанія застолбила випуском серії продуктів на платформі Hitachi Content Platform.

Власний об'єктний рішення, що допускає масштабування в хмарі, недавно запропонувала й Dell EMC. Мова йде про платформу Elastic Cloud Storage (ECS), у якій недавно з'явилася опція створення гібридної моделі зберігання. Компанія також оголосила про роботу над новим проектом Nautilus, який передбачає створення SDS-сховищ, що підтримують розвитий набір засобів аналітики. Передбачається, що такий рішення буде затребуваний для роботи з більшими масивами потокових даних, які з'являться в результаті бурхливого впровадження Інтернету речей (IoT).

На ринку SDS-рішень зараз зіштовхнулися представники різних груп. З одного боку, це вендори існуючих традиційних рішень (такі, як Dell EMC і IBM). З іншого боку, тут активно працюють компанії, у яких раніше не було подібних рішень (наприклад, Vmware і Red Hat). По оцінках IDC, у реальну трійку лідерів SDS-рішень зараз входять Dell EMC, IBM і Vmware.

Аналітики також уважають, на ринку SDS-рішень будуть розвиватися всі три напрямки типів зберігання даних – файловий, об'єктний і блоковий. Темпи їх середньорічного росту на період з 2014 по 2019 рр. складуть 10,5, 16,2 і 7,5% відповідно. Однак найбільш значний ріст очікується в категорії гіперконвергентних SDS-рішень – 59,7%.

Гіперконвергентні SDS-рішення

Рішення на базі гіперконвергентної інфрастуктури (Hyperconverged infrastructure, HCI) – це високоінтегровані системи, які мають повний комплекс ресурсів – обчислювальних, мережних і зберігання даних. Вони легко масштабуються в широкому діапазоні характеристик. Їхня гнучкість забезпечується насамперед за рахунок використання програмної начинки.

HCI-продукти цікаві споживачам насамперед тому, уважають в IDC, що вони дозволяють «підвищити продуктивність за рахунок використання флеш-

масивів, гібридної моделі зберігання, поліпшеної інтеграції їх керування із системами оркестрації хмарних платформ». Найбільшою популярністю на ринку сьогодні користуються HCI-рішення від Nutanix, Simplivity (недавно увійшло до складу HPE), Scaleio від компанії Dell EMC.

Своє гіперконверговане рішення має Vmware. Це – структурний блок vsan, що є частиною універсальної програмної платформи для будівництва ЦОДа. Вона представлено трьома компонентами: vsphere для віртуалізації серверів; vsan для створення високопродуктивної гіперконвергованої СЗД для віртуальних машин на флеш-масивах і vcenter для керування середовищами vsphere.

Netapp і Cisco недавно випустили власний спільний рішення для дата-центрів. Це – система Flexpod, яка дозволяє вишиковувати гіперконвергентні СЗД на встаткуванні Cisco і флеш-масивах Netapp Solidfire.

Інтерес замовників до HCI-рішень продиктований сьогодні, на думку Gartner, бажанням скоротити витрати на побудовування СЗД, домагаючись наміченої мети за рахунок використання вже встановленого встаткування, зниження витрат на його відновлення й технічний супровід.

Однак у такому підході є елемент ризику. Він полягає в наступному. Здобуваючи традиційні СЗД замовник завжди одержував устаткування, уже протестоване на сумісність із, що запускається ПЗ. У програмно-обумовлених системах програмна й апаратна частини розділені. Тому з'являється потреба в додатковій перевірці їх на сумісність і більш ретельному супроводі.

Враховуючи, що на ринку поки не так багато компаній, які мають достатній досвід і кадрами, щоб проводити такі перевірки, багатьом замовникам буде потрібно залучати фахівців з боку. Це – змушені витрати, які допоможуть уникнути непередбачених проблем, якщо вони все-таки існують.

3.2 Розробка структурної схеми

SDS в епоху NVMe

Ще два роки тому замовникам не потрібна була продуктивність в 1 млн IOPS на один вузол. При плануванні розробок у компанії виходили з того, що це потрібно для рішення тільки одиничних нішевих завдань. Однак життя, як звичайно, внесла свої корективи: у цілому ряді проектів уже незабаром знадобилося забезпечити підтримку декількох мільйонів операцій уведення-виводу в секунду.

Крім того, на ринку з'явилося нове обладнання; наприклад, продуктивність серверів зберігання Serv24 висотою 2U від Western Digital становить 6 млн IOPS, а SSD компанії Intel у вигляді лінійки (ruler) відкривають шлях до досягнення ємності 1 Пбайт на 1U. Останнє нововведення робить застарілими колишні підходи до зберігання метаданих, оскільки при долі в 5–8% їх обсяг перевищить терабайтний рівень. Побудувати вузол з оперативною пам'яттю на трохи терабайтів – недешеве завдання.

Поява нових рішень і вимог змусила фахівців переглянути весь стек зберігання й створити спеціальну технологію, покликану забезпечити підтримку протоколів NVMe і NVMe over Fabric і продуктивність 5-10 млн IOPS на вузол із затримкою не більш 2 мс, причому при будь -яких умовах, у тому числі під час реконструкції даних після відмови дисків. Крім того, додатково ставилося завдання підтримки різних сценаріїв розгортання: класичного програмного RAID, підключення зовнішніх систем зберігання по NVMe-oF і розподіленого горизонтально масштабованого блокового сховища.

У тестах відкриті програмні реалізації RAID (RAID-Z і MD-RAID) демонструють низькі результати при роботі з NVMe: через протяжний і повільний шлях передачі даних (data path) вони не здатні повністю задіяти пропускну здатність накопичувачів NVMe. Тому при переробці стека ПЗ в постаралися скоротити маршрут і прискорити проходження даних.

У основ ERA, програмн RAID для накопичувач NVMe, були покладені чотири принцип: швидкі обчислення, що неблокируюча (lockless) архітектура, паралелізація и оптимізація черг. Бібліотека для завадостійкого кодування, яка використовується в ПЗ Raidix 4.x і інших продуктах, є найшвидшою у світі, забезпечуючи високу швидкість обчислень навіть при роботі з NVMe.

Архітектура без блокувань припускає організацію окремого шляху для прискореного проходження даних. Оптимізація черг дозволяє уникнути непотрібних затримок. Для досягнення максимальної продуктивності здійснюється розпаралелювані операцій між ядрами й потоками. У результаті вдалося добитися високої продуктивності: 4 млн IOPS для масиву RAID 6 з 10 дисків SSD/HDD системи SDS при затримці менш 0,5 мс. Цей показник близький до загальної теоретичної продуктивності дисків, яка становить 4,5 млн IOPS.

У свою розподілену систему зберігання даних RAIN уважають більш кращим вибором, чому Ceph (правда, RAIN є блоковим сховищем, а Ceph – об'єктним). З одного боку, це зручна багатофункціональна система, оскільки за останні роки в ній з'явилося більш 500 нових параметрів, а з іншого, навіть фахівцеві важко розібратися у всіх можливостях Ceph. Серед переваг рішення виділімо виконання реконструкції із пріоритизацією у фоновому режимі, використання завадостійкого кодування замість реплікації й більш високу продуктивність.

Програмно-обумовлене сховище для NVMe

Програмне забезпечення дозволяє об'єднати мережні NVMe-накопичувачі в розподілене блокове сховище, поверх якого може бути розгорнута будь-яка локальна або розподілена файлова система. Поєднувані ресурси розміщають як на зовнішній системі зберігання, так і безпосередньо в серверах. Як затверджується, додаткова внесена затримка (у порівнянні із затримками при локальному доступі) не перевищує при цьому 5 мкс.

Для доступу до ресурсів пулу на сервері повинен бути встановлений програмний клієнт, а на цільовому пристрої – NVMesh Target Module. Їхня

віддалена взаємодія забезпечить мережна версія протоколу NVMe – NVMe поверх фабрики (NVMe over Fabric, NVMe-oF). Для моніторингу й конфігурування розподіленого зберігання в мережі потрібний модуль керування NVMesh.

NVMesh підтримує гіперконвергентну модель, коли кожний сервер додатка є одночасно сервером зберігання. Одним з недоліків такої архітектури вважаються потенційні проблеми із продуктивністю застосунків у результаті виконання операцій зберігання – так звана проблема гучних сусідів. Щоб виключити її виникнення, використовуємо метод віддаленого прямого доступу до дисків SSD/HDD системи SDS (Remote Direct Drive Access, RDDA).

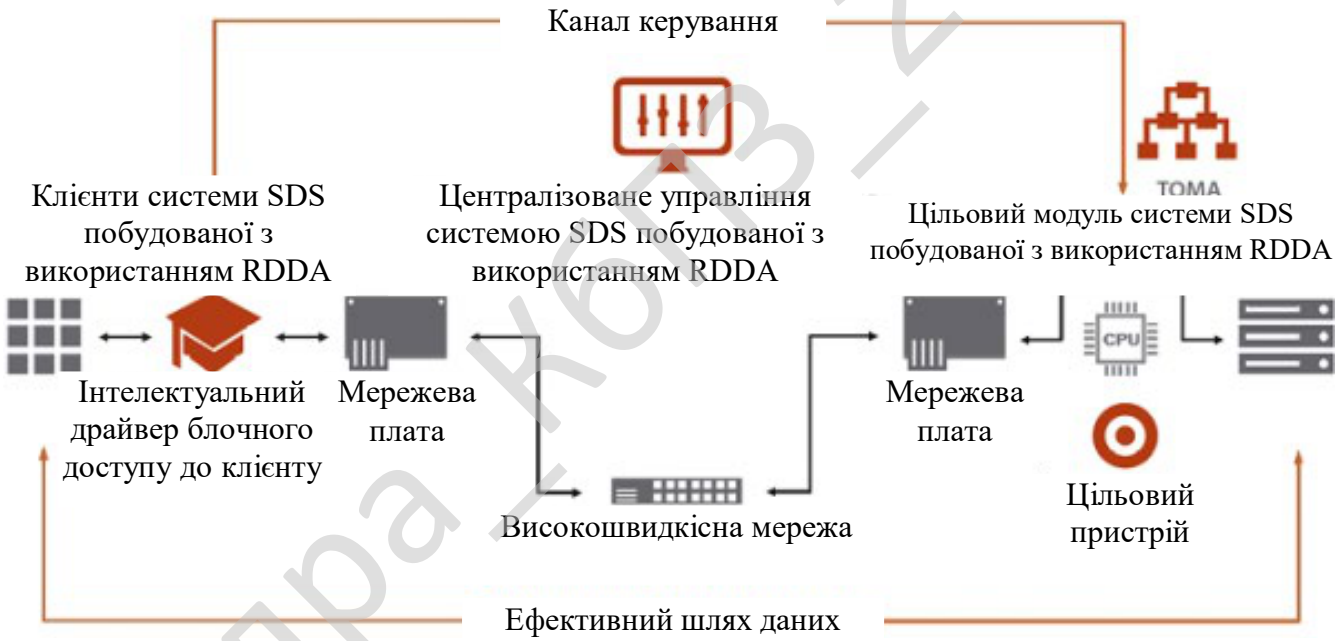

Рисунок 3.1 – Структурна схема системи

Метод віддаленого прямого доступу до дисків SSD/HDD системи SDS (Remote Direct Drive Access, RDDA) дозволяє звертатися до віддалених дисків, не задіючи ресурси процесора

Як видно з назви, RDDA відтворює метод віддаленого прямого доступу до пам'яті (Remote Direct Memory Access, RDMA): клієнти одержують дистанційний

доступ до дисків SSD/HDD системи SDS NVMe, а процесорні ресурси на віддаленій стороні не задіються. Переклад підтримки сервісів зберігання із цільового пристрою на клієнта дозволяє, як затверджують у компанії, забезпечити детерміновану продуктивність для застосунків і лінійне масштабування розподіленого сховища.

Пропонований рішення є повністю програмним, однак на всіх клієнтах (у якості яких виступають сервери застосунків) і серверах зберігання повинні бути мережні плати з підтримкою RDMA (R-NIC) (точніше, RoCE v2). Це обмежує потенційну застосовність NVMesh у гіпермасштабованих інсталяціях, на які воно орієнтоване, тому що припускає заміну великої кількості встаткування. (Не випадково Facebook разом з іншими учасниками ринку стимулювала розробку NVMe поверх TCP.)

У новій версії NVMesh додамо підтримку Fibre Channel і TCP/IP. Ціною цього нововведення стало підвищення затримки, зате замовники тепер можуть використовувати наявні в них мережі. Як затверджується, затримка становить усього чверть від типової для флеш -масивів, а от продуктивність уведення-виводу може досягати десятків і сотень тисяч IOPS.

Магнітна стрічка або оптичні диски?

Найчастіше дані необхідно зберігати протягом дуже тривалого часу. Це стосується, наприклад, цифрових копій музейних експонатів, кінофільмів, документації на інфраструктурні об'єкти, величезних масивів даних користувачів у соціальних мережах і т.п. Для такої інформації потрібно забезпечити економічне й довгострокове зберігання.

Втрата даних – найстрашніше, що може відбутися з ними. Але при довгостроковому зберіганні протягом 25-50 років до всіх надзвичайних ситуацій підготуватися неможливо». Так, наприклад, у скандинавському відділенні біржі NASDAQ у результаті неправильної тривоги спрацювала система пожежогасіння. Голосний пронизливий звук газу, що виходить із некаліброваних редукторів,

привів до поломки жорстких дисків SSD/HDD системи SDS у третині всіх серверів. У результаті біржа не працювала кілька годин.

Класичним підходом до забезпечення схоронності даних є стратегія «3-2-1»: наявність трьох копій даних, використання двох різних типів носіїв, зберігання однієї з копій в іншому місці. Яку ж технологію вибрати для запису копій? При довгостроковому зберіганні завжди виникає питання можливості їх зчитування згодом. Стрічкові приводи забезпечують сумісність тільки із двома попередніми поколіннями стрічок. Це означає, що поточний формат LTO-8 приводи нового, 11-го покоління, коли вони з'являться (а це відбудеться приблизно через 10 років), читати вже не зможуть. Тим часом навіть найперші оптичні диски, які з'явилися 36 років тому, можуть бути прочитані сучасними приводами BluRay, оскільки оптичні технології підтримують усі попередні формати.

3.3 Розробка функціональної схеми

Функціонально програмне забезпечення системи SDS побудованої з використанням RDDA складається з використання технології RAID. На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Система складається з наступних блоків.

Блок вибору кількості дисків SSD/HDD системи SDS – призначений для визначення кількості дисків SSD/HDD системи SDS з яких буд е складатися RAID-масив. Якщо дисків SSD/HDD системи SDS буде 2 то буде використовуватися технологія RAID 0 та RAID 1. Якщо дисків SSD/HDD системи SDS буде більше 2 то буде використовуватися технологія RAID 0 та RAID 6.

Блок формування віртуальних дисків SSD/HDD системи SDS використовується у разі коли диск тільки один. Тоді на цьому диску формуються декілька віртуальних дисків, з якими відбуваються дії аналогічні як ті, що

відбуваються над фізичними дисками. Тобто RAID-масив формується з віртуальних дисків.

Блок розподілу інформації за критерієм критичності/важливості визначає який саме вид RAID-масиву необхідно буде використовувати. Якщо інформація не є критичною то буде використовуватися RAID 0. Якщо інформація є важливою, то в залежності від ступеня важливості, та вимог до швидкодії, буде використовуватися або RAID 1, який є більш швидким, та менш надійним, або RAID 6, який є дуже надійним але менш швидким.

Блок вибору типу RAID масиву дозволяє автоматично, в залежності від заданих параметрів у попередніх блоках, визначити, який саме тип RAID-масиву потрібен для зберігання тієї, або іншої інформації.

Блоки RAID 0, RAID 1 та RAID 6 реалізують ту, або іншу технологію.

У блоці RAID 6 використовується кодек Ріда-Соломона. Нижче надамо опис цього кодеку.

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.

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.

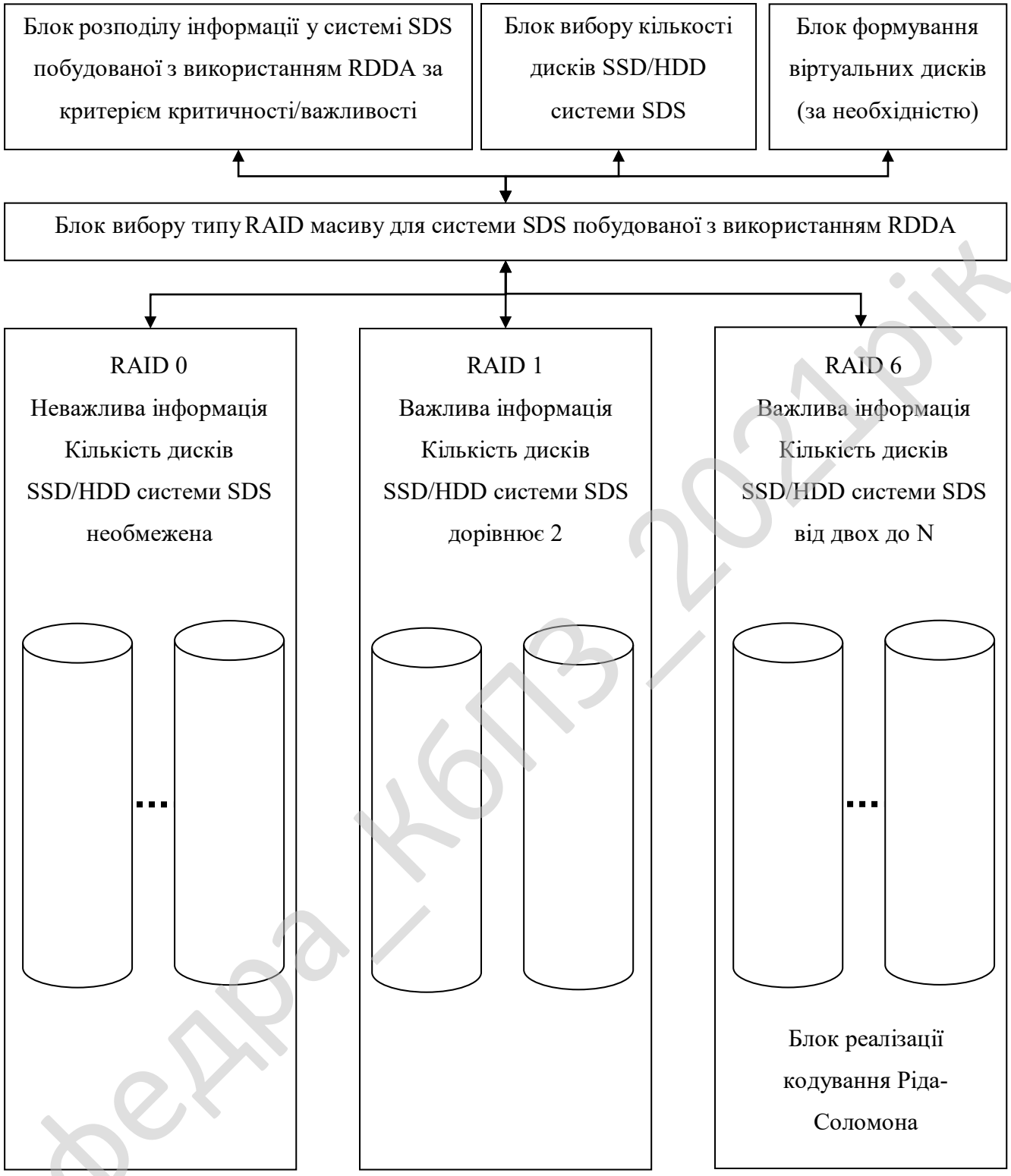

Рисунок 3.2 – Функціональна схема системи

Поліном, що породжує

Визначення поліномом, що породжує, циклічного (*n*,*k*) *коду* С 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:

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

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 $H = \begin{bmatrix} 1 & x & x^2 & \dots & x^{n-2} & x^{n-1} \end{bmatrix} \mod g(x)$.

Тоді:

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

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, $\alpha^{l_0}, \alpha^{l_0+1}, \dots, \alpha^{l_0+d-1}$ елемента α, є поліномом, що породжує, коду над *полем* GF(q) $g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1}) \dots (x - \alpha^{l_0+d-1})$:, де l_0 – деяке ціле число (у тому числі 0 і 1), за допомогою якого іноді вдається спростити кодер. Звичайно покладається $l_0 = 1$. Ступінь багаточлена $g(x)$ дорівнює $d - 1$.

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

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

Властивості

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

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 2*t*, 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 нацело, 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,..., 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 корня

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 Форни 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.

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

3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3.

Розроблена діаграма взаємодії процесів використовується для представлення та візуалізації процесів обробки даних тобто структурного проектування бакалаврської роботи.

Основні складові елементи діаграми взаємодії процесів це потоки даних:

– Репозиторії, поток сховища даних.

– Потоки зовнішні по відношенню до системи сутності.

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

– Потоки даних гібридні між елементами трьох попередніх типів.

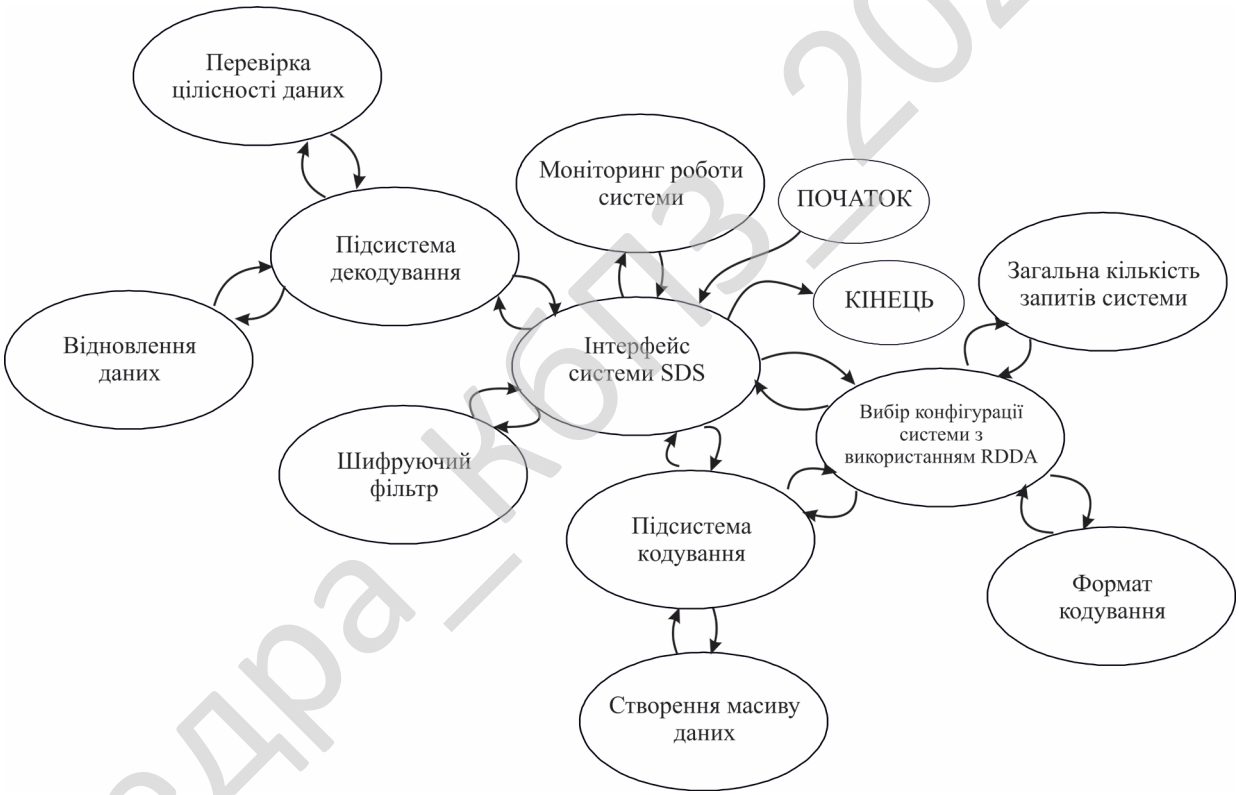

Рисунок 3.3 – Діаграма взаємодії процесів

Відповідно до документації 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.

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Таким чином у результаті після розгляду, вищеописаної системи, схеми структурної, функціональної, діаграми взаємодії процесів перейдемо до опису та розгляду блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Відповідно до обраної теми бакалаврської роботи – «Програмне забезпечення системи SDS побудованої з використанням RDDA», розглянемо блок схему розробленого програмного забезпечення яку можна побачити на рисунку 4.1. Відповідно роботу підпрограми зображено на рисунку 4.2.

Зі схеми видно що спочатку відбувається виведення головного вікна ПЗ з обранням користувача основного диску та величини надмірності кодування з файлом обробки. У подальшому йде робота основного функціонала з запитами та викликами підпрограми. Основний функціонал реалізовано з використанням алгоритму Ріда-Соломона (шифрування/дешифрування). Основні запити системи це запит створення рейд масиву, скидання дисків, перевірка цілісності, перегляду довідки та створення шифрую чого фільтру роботи с системи.

Спочатку розглянемо які програмні рішення було використано для реалізації системи. Впершу чергу було використано Redmine це вільне серверне ПЗ для управління проектами та відстежування помилок. До системи входить календар-планувальник та діаграми Ганта для візуального представлення ходу робіт за проектом та строків виконання. Redmine написано на мові Ruby і є ПЗ розробленим з використанням відомого веб-фреймворку Ruby on Rails, що означає легкість в розгортанні системи та її адаптації під конкретні вимоги. Для кожного проекту можна вести свої вікі та форуми.

Функціональні можливості:

– Ведення декількох проектів;

– Гнучка система доступу з використанням ролей;

– Система відстеження помилок;

– Діаграми Ганта та календар;

– Ведення новин проекту, документів та управління файлами;

– Сповіщення про зміни за допомогою RSS-потоків та електронної пошти;

– Власна Wiki для кожного проекту;

– Форуми для кожного проекту;

– Облік часових витрат;

– Налаштування власних (custom) полів для задач, затрат часу, проектів та користувачів;

– Легка інтеграція із системами керування версіями (SVN, CVS, Git, Mercurial, Bazaar и Darcs).

– Створення записів про помилки на основі отриманих листів

– Підтримка LDAP автентифікації;

– Можливість самореєстрації нових користувачів;

– Багатомовний інтерфейс (у тому числі українська мова);

– Підтримка СКБД: MySQL, PostgreSQL, SQLite.

Діаграма Ганта (*Gantt chart*, також стрічкова діаграма, графік Ганта) – це популярний тип діаграм, який використовується для ілюстрації плану, графіка робіт за будь-яким проектом. Є одним з методів планування та управління проектами.

Діаграма Ганта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі. Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і тривалості завдання. На деяких діаграмах Ганта також показується залежність між завданнями.

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню,

заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні».

Часто діаграма Ганта використовується спільно з таблицею зі списком робіт, рядки якої відповідають окремо взятій задачі, зображеній на діаграмі, а стовпці містять додаткову інформацію про задачу.

Система відстеження помилок Багтрекер – прикладна програма для допомоги розробникам програмного забезпечення (програмістам, тестувальникам тощо) враховувати і контролювати помилки, знайдені у програмах, питання щодо функціональності, рішення та оновлення, побажання користувачів, а також стежити за процесом їх виконання.

Кожному, хто розробляв програмні продукти, добре знайоме співвідношення «20/80» – останні 20 % роботи тривають 80 % часу.

Як це не парадоксально, але нічого дивного в цій пропорції немає, адже саме на завершальній стадії починається тестування проекту, коли виявляються помилки, і що більший проект, то більше буде знайдено помилок.

Водночас досить часто виявляється, що більшість цих помилок були відомі та могли бути виправлені з меншими витратами на попередніх стадіях роботи, але не були вчасно описані, а потім загубилися серед інших важливих завдань.

Отже, система відстеження помилок у найпростішому варіанті – це процес, що включає в себе виявлення помилки, її опис, виправлення і перевірку цього виправлення, тобто процес «стеження» за багом протягом всього як його життєвого циклу, так і життєвого циклу розробки в цілому.

Сукупність інформації про дефект. Головний компонент такої системи – база даних, що містить відомості про виявлені дефекти. Ці відомості можуть включати в себе:

– номер (ідентифікатор) дефекту;

– хто повідомив про дефект;

– дата і час виявлення дефекту;

– версія продукту, в якій виявлено дефект;

– серйозність (критичність) дефекту та пріоритет рішення;

– опис кроків для відтворення дефекту (неправильної поведінки програми);

– відповідальний за усунення дефекту;

– обговорення можливих рішень та їх наслідків;

– поточний стан виправлення дефекту;

– версії продукту, в якій дефект виправлений.

Крім того, розвинені системи надають можливість прикріплювати файли, які допомагають описати проблему, наприклад, дамп пам'яті або скріншот.

Використання. Основна перевага систем відстеження помилок полягає в забезпеченні чітких централізованих оглядів, запитів на розробку (включаючи помилки і виправлення) та їх стан. У корпоративному середовищі, системи відстеження помилок можуть бути використані для генерації звітів по продуктивності програмістів виправлення помилок. Однак, це може іноді приводити до неточних результатів, тому що різні помилки можуть мати різні ступені пріоритету та серйозності, що пов'язано з складністю їх фіксації.

Життєвий цикл дефекту. Як правило, система відстеження помилок використовує той чи інший варіант «життєвого циклу» помилки, стадія якого визначається поточним станом помилки.

Типовий життєвий цикл дефекту:

1. Новий – дефект зареєстрований тестувальником.

2. Призначений – призначений відповідальний за виправлення дефекту.

3. Дозволений – дефект переходить назад у сферу відповідальності тестувальника. Як правило, супроводжується резолюцією, наприклад:

– Виправлено (виправлення включені у версію таку-то).

– Дубль (повторює дефект, що вже знаходиться в роботі).

– Не виправлено (працює відповідно до специфікації, має занадто низький пріоритет, виправлення відкладено до наступної версії тощо).

– «В мене все працює» (запит додаткової інформації про умови, в яких дефект проявляється).

4. Далі тестувальник проводить перевірку виправлення, залежно від чого дефект або знову переходить у стан «Призначений» (якщо він описаний як виправлений, але не виправлений), або у стан «Закрито».

5. Відкрито повторно – дефект знайдено знову в іншій версії.

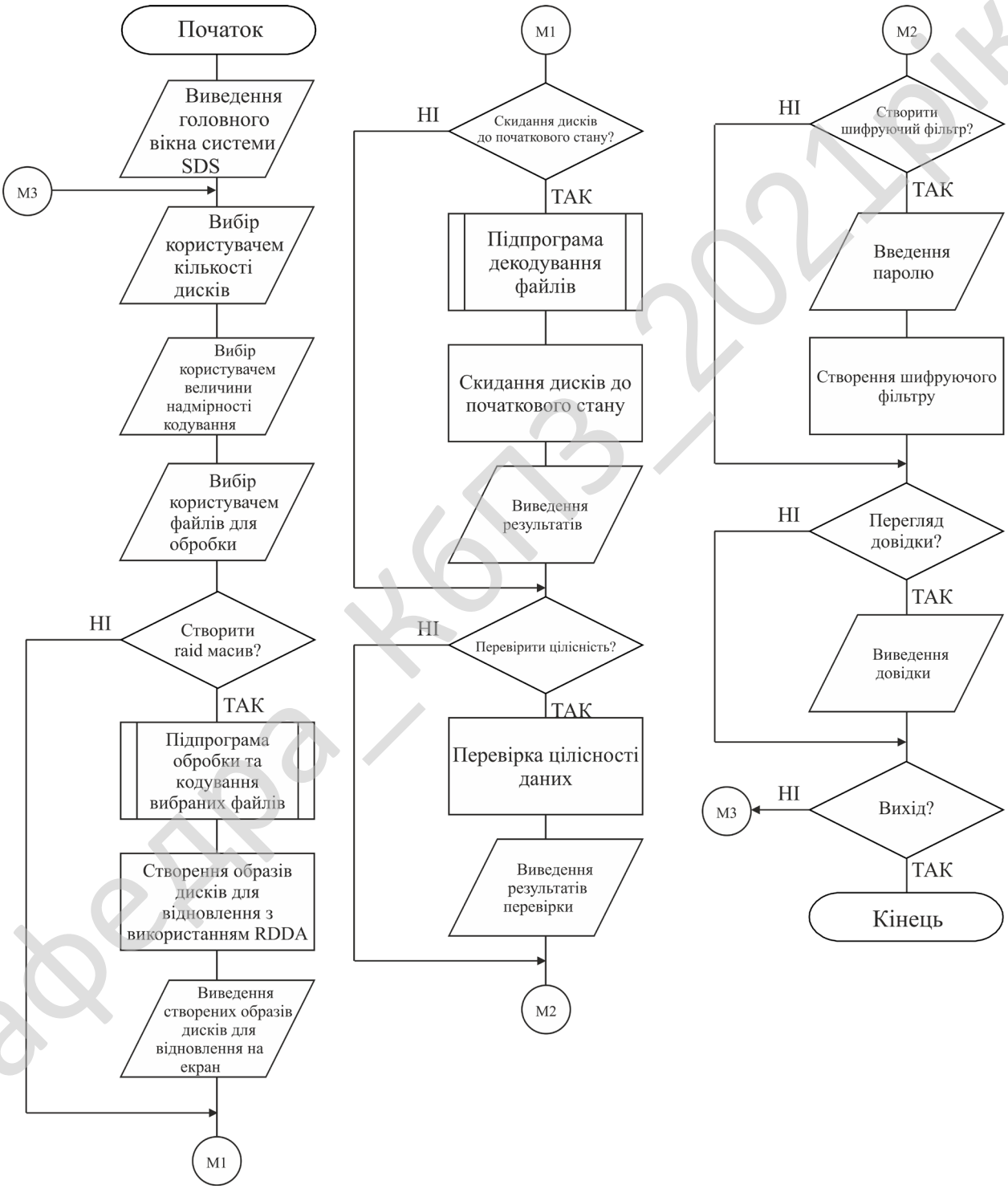

Рисунок 4.1 – Блок-схема роботи основної програми

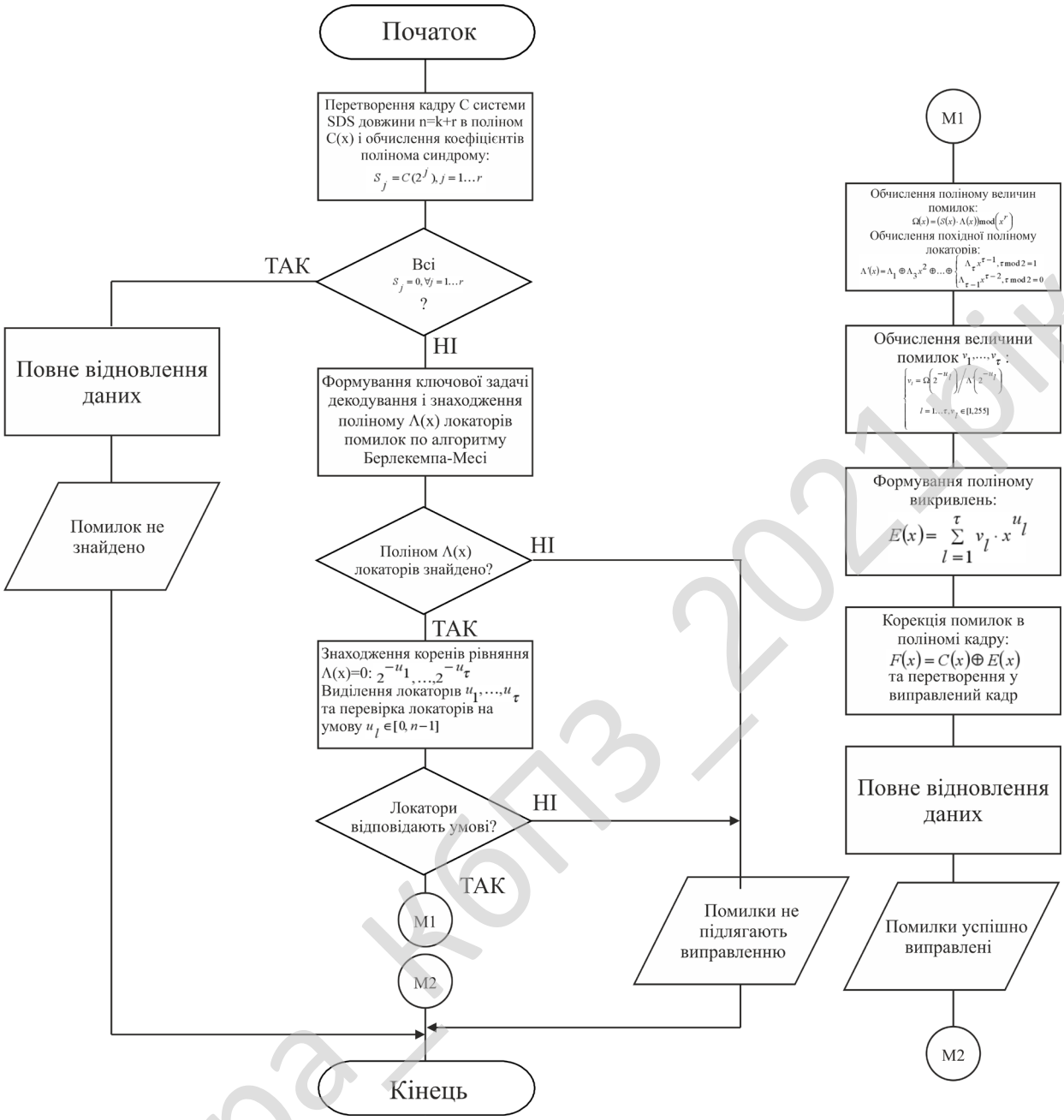

Рисунок 4.2 – Блок-схема роботи підпрограми

Система може надавати адміністраторові можливість налаштування користувачі, які можуть переглядати і редагувати помилки залежно від їх стану, переводити їх в інший стан або видаляти.

У корпоративному середовищі, система відстеження помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає

достатньо точних результатів через те, що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми прямо не стосується складності її усунення.

Розглянемо як саме було застосовано кодер Ріда-Соломона.

```
ERS ()
{
        int i, j; int feedback;
        // ініціалізація поля біт парності нулями
        for (i = 0; i < n - k; i++) b[i] = 0;
        // обробляємо всі символи
        // вихідних даних праворуч ліворуч
        for (i = k - 1; i >= 0; i-i--)
        {
// готовимо (data[i] + b[n - k - 1]) до множення на g[i]
              feedback = index_of[data[i] ^ b[n - k - 1]];
 // є ще символи для обробки?
              if (feedback != -1)
              {
 // здійснюємо зрушення ланцюга bx-регістрів
                    for (j = n - k - 1; j > 0; j-j--)
  // якщо поточний коефіцієнт g - це дійсний
 // (тобто ненульовий коефіцієнт, те
  // множимо feedback на відповідний g-коефіцієнт
  // і складаємо його з наступних елементів ланцюжка
     if (g[j] != -1) b[j] = b[j - 1] ^ alpha_to[(g[j] + feedback) % n];
                    else
                          b[j] = b[j - 1];
   // закільцьовуємо вихідний символ у крайній лівий b 0-регістр
                    b[0] = alpha_to[(g[0] + feedback) % n];
              }
              else
              {
  // розподіл завершений,
  // здійснюємо останнє зрушення регістра,
  // на виході регістра буде частка, що губиться,
  // а в самому регістрі - шуканий залишок
         for (j = n - k - 1; j > 0; j-j--) b[j] = b[j - 1]; b[0] = 0;
}}}
```

Розглянувши алгоритм та програмну реалізацію процесе кодування перейдемо до процесе декодування закодованого тексту.

Типова схема декодування, що одержала назву авторегрессионого спектрального методу декодування. Перейдемо до розгляду блок-схеми процедури декодування. Спершу відбувається перетворення кадру в поліном та обчислюються коефіцієнти полінома синдрому.

Якщо усі коефіцієнти дорівнюють нулю, то відбувається повне відновлення даних й видається повідомлення, що помилок не знайдено.

У іншому випадку формується ключова задача декодування й знаходження поліному локаторів помилок по алгоритму Берлекемпа-Мессі.

Якщо поліном локаторів не знайдено, то помилки не підлягають виправленю, про що виводиться відповідне повідомлення.

У іншому випадку знаходяться корні рівняння локаторів, виділяються локатори та відбувається перевірка локаторів на умову.

Якщо локатори не відповідають умові то помилки не підлягають виправленю, про що виводиться відповідне повідомлення.

У іншому випадку відбувається обчислення поліному величини помилок, обчислюється його похідна, та обчислюються величини помилок. На основі цих даних формується поліном викривлень та відбувається корекція помилок у кадрі, у результаті чого дані відновлюються, про видається відповідне повідомлення.

Нижче наведемо вихідний текст підпрограми декодера Ріда-Соломона.

```
DRS()
{
        int i, j, u, q;
        int s[n - k + 1]; // поліном синдрому помилки
        int elp[n - k + 2][n - k]; // поліном локатора помилки лямда
        int d[n - k + 2];
        int l[n - k + 2];
        int u_lu[n - k + 2],
        int count = 0, syn_error = 0, root[t], loc[t], z[t + 1], err[n],
              reg[t + 1];
        // переводимо отримане кодове слово в індексну форму
        // для спрощення обчислень
        for (i = 0; i < n; i++) recd[i] = index_of[recd[i]];
        // обчислюємо синдром
        //---------------------------------
```

```
        for (i = 1; i <= n - k; i++)
        {
                s[i] = 0;         // ініціалізація s-регістра
                                  // на його вхід за замовчуванням надходить нуль
for (j = 0; j < n; j++) if (recd[j] != -1) s[i] ^=
alpha_to[(recd[j] + i * j) % n];
                if (s[i] != 0) syn_error = 1;
// якщо синдром не дорівнює нулю, зводимо прапор помилки
// перетворимо синдром з поліноміальної форми в індексну
                s[i] = index_of[s[i]];
        }
        // корекція помилок
        //------------------------
        if (syn_error)
// якщо є помилки, намагаємося їх скорегувати
        {
 // обчислення полінома локатора ламбла
//--------------------------
                // ініціалізація елементи таблиці
                d[0] = 0; // індексна форма
                d[1] = s[1]; // індексна форма
                elp[0][0] = 0; // індексна форма
                elp[1][0] = 1; // поліноміальна форма
                for (i = 1; i < n - k; i++)
                {
                        elp[0][i] = -1; // індексна форма
                        elp[1][i] = 0; // поліноміальна форма
                }
                l[0] = 0; l[1] = 0; u_lu[0] = -1; u_lu[1] = 0; u = 0;
                do
                {
                        u++;
                        if (d[u] == -1)
                        {
                                l[u + 1] = l[u];
                                for (i = 0; i <= l[u]; i++)
                                {
                                        elp[u + 1][i] = elp[u][i];
                                        elp[u][i] = index_of[elp[u][i]];
                                }
                        }
                                else
```

```
                    {
                        // пошук слів з найбільшим u_lu[q], таких що d[q] != 0
                        q = u - 1;
                        while ((d[q] == -1) && (q > 0)) q-q--;
                        // знайдений перший ненульовий d[q]
                        if (q > 0)
                        {
                            j = q;
                            do {
                            j-j-- ;
                            if ((d[j] != -1) && (u_lu[q] < u_lu[j]))
                            q = j;
                        } while (j > 0);
                    };
if (l[u] > l[q] + u - q) l[u + 1] = l[u]; else l[u + 1] = l[q] + u - q;
for (i = 0; i < n - k; i++) elp[u + 1][i] = 0;
for (i = 0; i <= l[q]; i++)
if (elp[q][i] != -1)
elp[u + 1][i + u - q] = alpha_to[(d[u] + n - d[q] + elp[q][i]) % n];
                    for (i = 0; i <= l[u]; i++)
                    {
                        elp[u + 1][i] ^= elp[u][i];
                        // перетворимо старий elp
                        // в індексну форму
                        elp[u][i] = index_of[elp[u][i]];
                    }
            }
            u_lu[u + 1] = u - l[u + 1];
//---------------------------
            if (u < n - k)          // на останній ітерації розбіжність
            {                       // не було виявлено
if (s[u + 1] != -1) d[u + 1] = alpha_to[s[u + 1]]; else d[u + 1] = 0;
for (i = 1; i <= l[u + 1]; i++)
if ((s[u + 1 - i] != -1) && (elp[u + 1][i] != 0))
d[u + 1] ^= alpha_to[(s[u + 1 - i] + index_of[elp[u + 1][i]]) % n];
                    // переводимо d[u + 1] в індексну форму
                    d[u + 1] = index_of[d[u + 1]];
            }
        } while ((u < n - k) && (l[u + 1] <= t));
// розрахунок локатора завершений
        //-------------------------
        u++ ;
```

```
        if (l[u] <= t)
        {
// корекція помилок можлива
// переводимо elp в індексну форму
for (i = 0; i <= l[u]; i++) elp[u][i] = index_of[elp[u][i]];
// знаходження корінь полінома локатора помилки
//---------------------------
for (i = 1; i <= l[u]; i++) reg[i] = elp[u][i]; count = 0;
               for (i = 1; i <= n; i++)
               {
                       q = 1 ;
                       for (j = 1; j <= l[u]; j++)
                       if (reg[j] != -1)
                       {
                              reg[j] = (reg[j] + j) % n;
                              q ^= alpha_to[reg[j]];
                       }
                       if (!q)
{
root[count] = i;
loc[count] = n - i;
count++;
}
} if (count == l[u])
{ // немає корінь - ступінь elp < t помилок
// формуємо поліном z(x)
for (i = 1; i <= l[u]; i++) // Z[0] завжди дорівнює 1
{
if ((s[i] != -1) && (elp[u][i] != -1))
z[i] = alpha_to[s[i]] ^ alpha_to[elp[u][i]];
else
if ((s[i]! =- 1) && (elp[u][i] == -1)) z[i] = alpha_to[s[i]];
else
if ((s[i]==-1) && (elp[u][i]!=-1)) z[i] = alpha_to[elp[u][i]];
else
z[i] = 0 ;
for (j = 1; j < i; j++)
if ((s[j] != -1) && (elp[u][i - j] != -1))
z[i] ^= alpha_to[(elp[u][i - j] + s[j]) % n];
// переводимо z[i] в індексну форму
z[i] = index_of[z[i]];
}
```

```
for (i = 0; i < n; i++)
{
err[i] = 0;
if (recd[i] != -1) recd[i] = alpha_to[recd[i]]; else recd[i] = 0;
}
// спочатку обчислюємо чисельник помилки
for (i = 0; i < l[u]; i++)
{
err[loc[i]] = 1;
for (j = 1; j <= l[u]; j++)
    if (z[j] != -1)
err[loc[i]] ^= alpha_to[(z[j] + j * root[i]) % n];
                    if (err[loc[i]] != 0)
                    {
                  err[loc[i]] = index_of[err[loc[i]]];
                  q = 0 ; // формуємо знаменник коефіцієнта помилки
                  for (j = 0; j < l[u]; j++)
                  if (j != i)
q += index_of[1 ^ alpha_to[(loc[j] + root[i]) % n]];
q = q % n; err[loc[i]] = alpha_to[(err[loc[i]] - q + n) % n];
                           recd[loc[i]] ^= err[loc[i]];
} } }
else // немає корінь,
// рішення системи рівнянь неможливо, тому що ступінь elp >= t
{
// переводимо recd[] у поліноміальну форму
for (i = 0; i < n; i++)
if (recd[i] != -1) recd[i] = alpha_to[recd[i]];
else
recd[i] = 0; // виводимо інформаційне слово як є
}
else // ступінь elp > t, рішення неможливо
{
// переводимо recd[] у поліноміальну форму
for (i = 0; i < n; i++)
if (recd[i] != -1)
recd[i] = alpha_to[recd[i]];
else
recd[i] = 0 ; // виводимо інформаційне слово як є
       }
els   for (i = 0; i < n; i++) if (recd[i] != -1) recd[i] = alpha_to[recd[i]]; else
recd[i] = 0;}
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Khufu. Khufu – це 64-бітовий блоковий шифр. 64-бітовий відкритий тест спочатку розщеплюється на дві 32-бітові половини, *L* і *R*. Над обома половинами й певними частинами ключа виконується операція XOR. Потім, аналогічно DES, р езультати проходять деяку послідовність раундів. У кожному раунді молодший значущий байт *L* використовується як вхід S-блоку. У кожного S-блоку 8 вхідних біт і 32 вихідних біта. Далі обраний в S -блоці 32-бітовий елемент піддається операції XOR з *R*. Потім *L* циклічно зрушується на число, кратним восьми биткам, *L* і *R* міняються місцями, і раунд завершується. Сам S-блок не статичний, він міняється кожні вісім раундів. Нарешті, по закінченні останнього раунду, над *L* і *R* виконується операція XOR з іншими частинами ключа, і половини поєднуються, утворюючи блок шифртексту.

Хоча частини ключа використовуються для операції XOR із блоком шифрування на початку й кінці виконання алгоритму, головне призначення ключа – генерація S-блоків. Ці S-блоки секретні, по суті, це частина ключа. Повний розмір ключа алгоритму Khufu дорівнює 512 біт (64 байт), алгоритм надає спосіб генерації S-блоків по ключу.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. З рисунку головного вікна можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Незважаючи на те що розроблене ПЗ працює у полу автоматичному режимі у інтерфейсі є функціональні кнопки: Пауза; Закрити.

– Навігаційного меню.

– Розділи: Контролю цілісності; Логування роботи ПЗ; Лічильнику; Пріоритету; Інформації результату аналізу.

– Розділу виведення результату роботи системи.

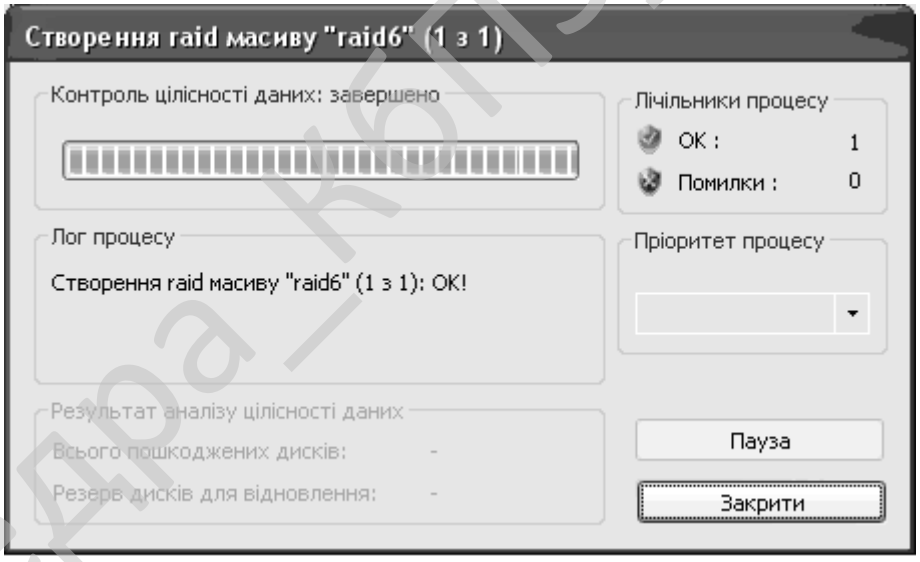

Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення. 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.

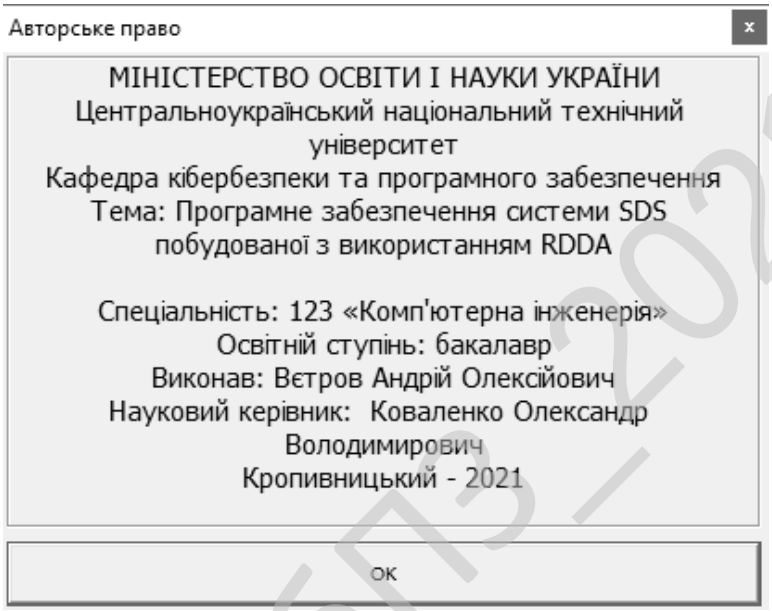

Рисунок 5.2 – Авторське право

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.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної бакалаврської роботи, призначено для системи SDS побудованої з використанням RDDA.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем SDS побудованої з використанням RDDA.

– Досліджена система SDS побудованої з використанням RDDA.

– На основі отриманих результатів досліджень створена програмна реалізація системи SDS побудованої з використанням RDDA.

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання SDS побудованої з використанням RDDA.

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 призначені для системи SDS побудованої з використанням RDDA. 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.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Khufu.

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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Якшин М.М Inquisitor свободная платформа тестирования и мониторинга программного обеспечения // На Протве. 5-я конференция разработчиков свободных программ. Обнинск, июль, 21–23. 2008. Тез. докл. М., Институт Логики: 2008. С. 17–20

2. D. Patterson G. Garth, R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)" University of Califomia, Berkely, Report No. UC8/CSD/87/391, December 1987.

3. Pete McLean "An Introduction to RAID Redundant Arrays of Inexpensive Disks. Digital Equipment Corporation, April 1991.

4. Azer Bestaves "IDA-based Redandant Arrays of Inexpensive Disks." Proceed. of I-st International Conference on Parallel and Distributed Information Systems. Dec 4-6, 1991. IEEE Comp. Society Pkss.

5. Lee, Edward Kiehyen; Chen, Peter Ming-Chien, and others "RAID-II A Scalable Storage Architecture for High-Bandwith Network File Service" University of California, Berkely, Report No. UCB/CSD-92-672, October 1992.

6. Lee, Edward Kiehyen. "Performance Modeling and Analisis of Disk Arrays" University of Califonia, Berkely, Report No. UCB/CSD-93-770, September 1993.

7. Кормен Т. Алгоритмы: построение и анализ / Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. – М.:"Вильямс", 2005. –1296 с.

8. Конахович Г.Ф. Сети передачи пакетных данных / Г.Ф. Конахович, В.М.Чуприн. – К.:МК-Пресс, 2006. – 272 с.

9. Королев А.В. Адаптивная маршрутизация в корпоративных сетях / А.В. Королев, Г.А. Кучук, А.А. Пашнев. – Х.: ХВУ, 2003. – 224 с.

10. Кучерявый Е.А. Управление трафиком и качество обслуживания в сети Интернет / Евгений Андреевич Кучерявый. – СПб.: Наука и техника, 2004. – 336 с.

11. Кучук Г.А. Управление ресурсами инфотелекоммуникаций / Г.А. Кучук, Р.П. Гахов, А.А. Пашнев. – М.: Физматлит, 2006. – 220 с.

12. Лагутин В.С., Степанов С.Н. Телетрафик мультисервисных сетей связи / В.С. Лагутин, С.Н. Степанов. – М.: Радио и связь, 2000. – 320 с.

13. Майника Э. Алгоритмы оптимизации на сетях и графах: пер. с англ. / Э. Майника; под ред. Е.К. Масловского. – М.: Мир, 1981. – 321 с.

14. Мохамад Гани Абу Таам Разработка математической GERT-модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / А.А.Смирнов, Мохамад Гани Абу Таам // Информационные системы в управлении, образовании, промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2014. – 498 с.

15. Мохамад Гани Абу Таам Метод управления доступом в интеллектуальных узлах коммутации / Мохамад Гани Абу Таам, А.А.Смирнов // Информационные технологии и защита информации в информационно-коммуникационных системах: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2015. – 486 с.

16. Мохамад Гани Абу Таам Математическая GERT-модель технологии передачи метаданных в облачные антивирусные системы / В.В.Босько, А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Збірник наукових праць "Системи обробки інформації". – Випуск 1(117). – Х.: ХУПС – 2014. – С. 137-141.

17. Мохамад Гани Абу Таам Структурно-логическая GERT-модель технологии распространения компьютерных вирусов / А.А.Смирнов, И.А.Березюк, Мохамад Гани Абу Таам // Системи управління, навігації та зв’язку. – Випуск 1(29). – П.: ПНТУ. – 2014. – С. 120-125.

18. Мохамад Гани Абу Таам Сравнительные исследования математических моделей технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, А.В. Коваленко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 9(125). – Х.: ХУПС – 2014. – С. 105-110.

19. Мохамад Гани Абу Таам Математическая модель интеллектуального узла коммутации с обслуживанием информационных пакетов различного приоритета / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 4 (41). – Харків: ХУПС. – 2014. – С. 48-52.

20. Мохамад Гани Абу Таам Исследование показателей качества функционирования интеллектуальных узлов коммутации в телекоммуникационных системах и сетях / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 4(17). – Харків: ХУПС. – 2014. – С.90-95.

21. Мохамад Гани Абу Таам Усовершенствованный алгоритм управления доступом к «облачным» телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, Н.С. Якименко, С.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 1(126). – Х.: ХУПС – 2015. – С. 150-153.

22. Мохамад Гани Абу Таам Анализ и исследование методов управления сетевыми ресурсами для обеспечения антивирусной защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Системи озброєння і військова техніка. – Випуск 3(43) – Х.: ХУПС – 2015. – С. 100-107.

23. Мохамад Гани Абу Таам Исследование эффективности метода управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 3(19). – Х.: ХУПС. – 2015. – С. 134-141.

24. Mohamad Abou Taam Method of controlling access to intellectual switching nodes of telecommunication networks and systems / A.A. Smirnov, Mohamad Abou Taam, S.A. Smirnov // International Journal of Computational Engineering Research (IJCER). – Volume 5, Issue 5. – India. Delhi. – 2015. – P. 1-7.

25. Мохамад Гани Абу Таам GERT-модель технологии передачи данных в облачные антивирусные системы / А.А. Смирнов, В.В. Босько, Мохамад Гани Абу Таам // Збірник тез доповідей науково-практичної конференції «Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку». м. Харків. 12-13 березня 2014 р. – Харків. АВВ МВС. – 2014. – С. 18-19.

26. Мохамад Гани Абу Таам Математическое моделирование технологии передачи сигнатур в облачные антивирусные системы / Мохамад Гани Абу Таам, А.А. Смирнов // Збірник тез VI міжнародної науково-практичної конференції "Проблеми і перспективи розвитку ІТ-індустрії". м. Харків. 17-18 квітня 2014 р. – Харків: ХНЕУ. – 2014. – С. 260.

27. Мохамад Гани Абу Таам Анализ требований к качеству обслуживания в информационно-телекоммуникационных системах / А.А. Смирнов, Мохамад Гани Абу Таам // Збірник тез XVI міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 11-12 квітня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 124-126.

28. Мохамад Гані Абу Таам Дослідження та реалізація GERT-моделі технології розповсюдження комп'ютерних вірусів для захисту телекомунікаційних систем / Мохамад Гані Абу Таам, С.А. Смирнов // Збірник тез науково-практичної конференції «Інформаційні технології та комп'ютерна інженерія». м. Кіровоград. 4 грудня 2014 р. – Кіровоград: КНТУ. – 2014. – С. 168.

29. Мохамад Гани Абу Таам Исследование математических моделей технологии распространения компьютерных вирусов / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник наукових праць міжнародної науково-

практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 25-28 лютого 2015 р. – Київ: Європейський університет. – 2015. – С. 90-91.

30. Мохамад Гани Абу Таам Метод управления доступом к «облачным» ресурсам для защиты телекоммуникационных систем / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез всеукраїнської науково-практичної конференції «Інформаційна безпека держави, суспільства та особистості». м. Кіровоград. 16 квітня 2015. – Кіровоград: КНТУ. – 2015. – С. 50-52.

31. Мохамад Гани Абу Таам Разработка метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам, С.А. Смирнов // Збірник тез VII міжнародної науково-практичної конференції "Проблеми і перспективи розвитку ІТ-індустрії". м. Харків. 17-18 квітня 2015 р. – Харків: ХНЕУ. – 2015. – С. 14.

32. Мохамад Гани Абу Таам Реализация метода управления доступом в интеллектуальных узлах коммутации / А.А. Смирнов, Мохамад Гани Абу Таам // Збірник тез XVII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 17-18 квітня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 91-92.

33. Мохамад Гани Абу Таам Реализация математической модели интеллектуального узла коммутации для обеспечения защищенности телекоммуникационной сети / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез II Міжнародної науково-практичної Інтернет-конференції «Інформаційна та економічна безпека» (INFECO-2015)». м. Харків. 21-22 травня 2015 р. – Харків: ХІБС УБС НБУ. – 2015. – С. 20-24.

34. Мохамад Гани Абу Таам Разработка математической модели технологии распространения компьютерных вирусов в информационно-телекоммуникационных сетях / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Сборник тезисов XI международной конференции "Стратегия

качества в промышленности и образовании". г. Варна. Болгария. 01 – 06 июня 2015 г – Варна. ТУВ. – 2015. – С. 488-491

35. Мохамад Гани Абу Таам Метод управления доступом к облачным телекоммуникационным ресурсам для обеспечения защиты данных / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Комп'ютерні технології та інформаційна безпека». м. Кіровоград. 2-3 липня 2015 р. – Кіровоград: КНТУ. – 2015. – С. 4-5.

36. Мохамад Гани Абу Таам Имитационная модель системы управления доступом к облачным антивирусным телекоммуникационным ресурсам / Мохамад Гани Абу Таам, А.А. Смирнов, С.А. Смирнов // Збірник тез першої всеукраїнської науково-практичної конференції «Перспективні напрями захисту інформації». м. Затока. 7-9 вересня 2015 р. – Одеса: ОНАЗ. – 2015. – С. 90-94.

37. МСЭ-Т Рекомендация G.101. Международные телефонные соединения и цепи – Общие определения //11/2003. [Електронний ресурс]. – Режим доступу до ресурсу: http://www. telecom61.ru›SharedFiles/Download.aspx? …pageid=106

38. Одом Ш. Коммутаторы CISCO / Ш. Одом, Х. Ноттингем – М.:"Кудиц-Образ", 2003. – 528 с.

39. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов / В.Г. Олифер, Н.А. Олифер. –2-е изд. – СПб.: Питер, 2007. – 958 с.

40. Руководство по технологиям объединенных сетей. 4-е изд. / пер.с англ. и ред. А.Н. Крикуна – М.: Изд. дом «Вильямс», 2005. – 1040 с.

41. Свами М.Н., Тхуласираман К. Графы, сети и алгоритмы: пер. с англ. / М.Н. Свами, К. Тхуласираман; под ред. В.А. Горбатова. – М.: Мир, 1984. – 454 с.

42. Семенов С.Г. Анализ методов прогнозирования в телекоммуникационных сетях автоматизированных систем

управления / С.Г.Семенов // Збірник наукових праць «Системи управління, навігації та зв'язку», – К.:ЦНДІ навігації і управління, – 2008.-Вип. 2(6) .- С.134-137

43. Семенов С.Г. Математическая модель процесса доставки информационных пакетов в компьютерной сети системы критического применения / С.Г.Семенов, И.В.Ильина // Науково-технічний журнал «Радіоелектронні і комп'ютерні системи» Х.:ХАІ, – 2008.-Вип. 1(28) – С.162-165

44. Семенов С.Г. Оптимизация трафика на основе сбалансированной загрузки информационно-телекоммуникационной сети // Системи обробки інформації. – Х.: ХВУ, 2004. – № 8(36). – С.206-210

45. Семенов С.Г. Математическая модель мультисервисного канала связи на основе экспоненциальной GERT-сети / С.Г. Семенов, Є.В. Мелешко, Я.В. Ілюшко // Системи озброєння і військова техніка. – Х.:ХУ ПС. – 2011. –Вип. 3(27). – С. 64-67.

46. Семенов С.Г. Математична модель системи криптографічного захисту електронних повідомлень на основі GERT-мережі / С.Г. Семенов, О.О. Сур // Системи управління, навігації та зв'язку. – К.:ЦНДІ навігації і управління. – 2012. – Том 1. Вип. 1(21). – С. 131-137

47. Семенов С.Г. Исследования вероятностно-временных характеристик мультисервисного канала связи с использованием математического аппарата GERT-сети / С.Г. Семенов, В.В. Босько, І.А. Березюк // Системи обробки інформації. – Х.: ХУ ПС. – 2012. – Том 1. Вип. 3(101). – С. 139-142.

48. Семенов С.Г. Моделирование защищенного канала связи с использованием экспоненциальной GERT-сети / С.Г. Семенов, А.А. Можаев // Информатика, математическое моделирование, экономика. – Смоленськ.: Смоленский филиал АНО ВПО ЦС РФ "Российский университет кооперации". – 2012. – Том.1. – С. 152-160.

49. Семенов С.Г. Методика математического моделирования защищенной ИТС на основе многослойной GERT-сети / С.Г. Семенов // Вісник

Національного технічного університету «Харківський політехнічний інститут». – Х.:НТУ «ХПІ». – 2012. –№62 (968). – С 173-181.

50. Семенов С.Г. Защита данных в компьютеризированных управляющих системах / С.Г. Семенов, В.В. Давыдов, С.Ю. Гавриленко. – LAP Lambert Academic Publishing GmbH & Co. KG (Саарбрюккен, Германия), 2014. – 236 с.

51. Смирнов А.А. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / А.А.Смирнов, В.В.Босько, Е.В.Мелешко // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

52. Смирнов А.А. Усовершенствование метода управления очередями в многопротокольных узлах телекоммуникационной сети / А.А.Смирнов, Е.В.Мелешко // Збірник тез та доповідей другої всеукраїнської науково-практичної конференції «Системний аналіз. Інформатика. Управління». Запоріжжя. Тези доповідей. Запоріжжя: КПУ, 2011.

53. Современные телекоммуникации. Технологии и экономика / [В.Л. Банкет, О.В. Бондаренко, П.П. Воробьенко и др.]; под ред. С.А. Довгого. – М.: Эко-Трендз, 2003. – 320 с.

54. Столлингс В. Современные компьютерные сети / Вильям Столлингс.– СПб.: Питер, 2003. – 778 с.

55. Таненбаум Э. Компьютерные сети / Эндрю Таненбаум; пер. с англ. А. Леонтьев. – СПб.: Питер, 2002. – 848 с.