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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД”

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

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

м. Кропивницький

АНОТАЦІЯ

Шарова А.Ю. Дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кешування, флеш-технології та реплікації ЦОД.

Метою розробки є дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД.

Об'єктом дослідження є процес кешування, флеш-технології та реплікації ЦОД.

Предметом дослідження є методи кешування, флеш-технології та реплікації ЦОД.

Методи дослідження базуються на методах теорії кодування, теорії великих даних,, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи кешування, флеш-технології та реплікації ЦОД.

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

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

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

Програму розроблено в середовищі Visual C#.

Ключові слова: Комп'ютерні науки, кешування, флеш-технології, реплікація, ЦОД

ABSTRACT

Sharova A.Yu. Research and software implementation of the caching system, flash technology and replication of data centers. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed, which is intended for the caching system, flash technology and replication of data centers.

The purpose of the development is the research and software implementation of the caching system, flash technology and replication of data centers.

The object of the research is the process of caching, flash technology and replication of data centers.

The subject of the research is the methods of caching, flash technology and replication of data centers.

The research methods are based on the methods of coding theory, big data theory, methods of mathematical statistics, methods of software development.

The result of the work is the software implementation of the caching system, flash technology and replication of data centers.

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

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

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

The program was developed in the Visual C# environment.

Keywords: Computer science, caching, flash technologies, replication, data center

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АВІ	–	адміністратора віртуальної інфраструктури
АІБ	–	адміністратор інформаційної безпеки
ВІ	–	віртуальна інфраструктура
ЗЗІ	–	засоби захисту інформації
ПД	–	персональні дані
ПЗ	–	програмне забезпечення
ЦЗОД	–	центр зберігання й обробки даних
ЦОД	–	центр обробки даних
DLP	–	захист від витоків даних
IPS	–	системи запобігання вторгнень
CIRC	–	Cross Interleaved Reed Solomon Code
EAB	–	Embedded Array Block, блок зосередженої пам'яті
ECC	–	error-correcting code, код корекції помилок
FEC	–	метод прямої корекції помилок
LDPC	–	коди Галлагера
NAK	–	негативне підтвердження

ВСТУП

Актуальність теми. Надійне зберігання даних – завдання, що доводиться вирішувати кожної організації. Що саме потрібно компаніям, щоб організувати роботу з інформацією щонайкраще? Зорієнтуватися на високотехнологічному ринку – непроста, але розв'язне завдання. Для зберігання даних використовують центри обробки даних.

Необхідність у центрах обробки даних (ЦОД) виникла, коли масиви збереженої й переданої інформації перевищили всі мислимі на той момент межі.

Згідно даним TAdviser, з 2010 р. обсяг збереженої інформації щороку зростає приблизно на 50% від її первісного обсягу. Росте й вартість інформації, оскільки від її прямо залежать всі бізнес-процеси.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД.

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

– Огляд існуючих систем кешування, флеш-технології та реплікації ЦОД.

– Дослідження системи кешування, флеш-технології та реплікації ЦОД.

– Програмна реалізація системи кешування, флеш-технології та реплікації ЦОД.

Об’єктом дослідження є процес кешування, флеш-технології та реплікації ЦОД.

Предметом дослідження є методи кешування, флеш-технології та реплікації ЦОД.

Методи дослідження базуються на методах теорії кодування, теорії великих даних,, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод кешування, флеш-технології та реплікації ЦОД.

– Розроблено вітчизняний продукт кешування, флеш-технології та реплікації ЦОД, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі кешування, флеш-технології та реплікації ЦОД.

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

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

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Минуло близько 20 років з тих пір, як флеш-пам'ять – у своєму надзвичайно домінантному варіанті NAND – вперше з'явилася в корпоративних центрах обробки даних. Відтоді вона трансформувала сховище даних і значно підвищила продуктивність широкого кола програм, замінивши набагато повільніші диски як носій за замовчуванням для основного зберігання даних.

Коли флеш-пам'ять вперше з'явилася в центрах обробки даних наприкінці 90-х, вона використовувалася лише для зберігання підмножини даних для підмножини програм, чутливих до продуктивності. Але оскільки ціни на флеш-пам'ять продовжували падати, а флеш-пам'ять використовувалася для зберігання даних для дедалі ширшого кола програм, спостерігачі галузі почали запитувати, скільки часу знадобиться, перш ніж флеш-пам'ять повністю витіснить диски, щоб створити так звані повністю флеш-центри обробки даних.

1.2 Область застосування

Основні елементи

ЦОД складається з накопичувачів інформації, серверів, інфраструктури, що забезпечує зв'язок між ними, і системи керування.

Типи ЦОД

Системи зберігання даних по типі накопичувачів інформації діляться на три великих групи.

– Дискові. Використовуються найперші, розповсюджені й недорогі накопичувачі. У сучасних умовах істотним недоліком стає те, що швидкість передачі інформації обмежується швидкістю обертання шпинделя, на якому

закріплені пластини жорсткого диска, однак сучасні дискові ЦОД дуже економічні й «розумні» у порівнянні з їхнім попередниками.

– Стрічкові (касетні). Мобільність касет у сполученні з можливістю тривалого зберігання й відновлення інформації роблять їхнім популярним засобом для створення надійного електронного архіву з фізичним обмеженням доступу до інформації. Широко використовуються в мультимедійних бібліотеках, де особливо важлива низька вартість терабайта інформації.

– Флеш. Напівпровідникові накопичувачі відрізняються найвищою швидкістю роботи. Якщо в жорсткого диска на обробку запиту йде в середньому 6-7 мс, то для флеш-накопичувачів цей показник досягає 0,1 мс. Таким чином, кількість транзакцій у секунду зростає на 1-2 порядки. Донедавна флеш-накопичувачі вважалися дорогими й використовувалися в гібридних системах разом з дисковими. Зараз ситуація міняється й всі частіше впроваджуються ЦОД повністю на флеш-накопичувачах, які дозволяють істотно заощадити простір серверів.

Створення системи зберігання даних

Для створення сховищ даних потрібна розробка логічної моделі, що буде повністю відбивати очікування клієнта й можливості розроблювача. Після цього можна розглядати технологічні аспекти – наприклад, розміри сховища. Логічна модель може містити тисячі атрибутів і зв'язків. Вартість ЦОД варіюється залежно від масштабу, логічної моделі й устаткування. В одних випадках мова йде про сотні тисяч гривень, в інші – про десятки мільйонів. На створення ЦОД може піти від одного місяця до півроку. Важливим фактором, якому варто враховувати, є необхідність сервісної підтримки устаткування.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Система зберігання даних (ЦОД) – це конгломерат спеціалізованого устаткування й програмного забезпечення, що призначений для зберігання й передачі великих масивів інформації. Дозволяє організувати зберігання інформації на дискових площадках з оптимальним розподілом ресурсів.

«Фізика» зберігання

Можливо, сама захоплююча частина комп'ютерної історії – це хроніка систем зберігання даних (ЦОД), тому що в цій області була велика розмаїтість і у фізику, і в системній організації, до того ж багато років тут усе було дуже наочно. Комп'ютери незабаром втратилися зорової привабливості, на зміну гарній і різноманітній вакуумній лампам і окремим напівпровідниковим компонентам (тріодам і діодам) прийшли однакові інтегральні схеми й мікропроцесори. Тепер ми можемо розрізняти по написах щось, укладене в корпуси різних розмірів, що розрізняються кількістю контактів. Фізика напівпровідникових новацій в остаточному підсумку зводиться до пошуку наукових і технологічних рішень, що забезпечують збільшення щільності транзисторів на підложці. Ці найважливіші досягнення не мають зовнішнього вигляду й для споживача зводяться до цифр 0.18, 0.13, 0.11..... Втім, сьогодні теж саме можна сказати й про диски – зовні це коробки декількох типорозмірів, що розрізняються вмістом.

Дискові масиви

Наприкінці 90-х вдало зійшлися дві новації – наукова база RAID і вінчестери, що випускаються масовим тиражем. Якщо зібрати їх разом, виявилося можливим створити комерційний накопичувач кластерного типу,

здатний конкурувати з дисками IBM за технічними показниками при істотно меншій ціні.

Гігантизм дисків, що випускалися до появи вінчестерів, суперечив немудрою логікою цих пристроїв. Їхня логіка була примітивна, вона майже повністю відповідала фізичній інфраструктурі (сектори й доріжки). І, як малотиражні й спеціалізовані вироби, вони були дорогими.

В 1988 році Майкл Рюттгерс, що надалі став головним стратегом EMC, запропонував розробити дискову систему, що складається з вінчестерів, і поставляти їх для мейнфреймів, сумісних з IBM, і для AS/400. Інший, мабуть, самий щасливий фахівець з ЦОД Моше Янаі висунув ідеологію кеш-пам'яті Integrated Cached Disk Array (ICDA), у результаті народився прабатько дискових кластерів EMC Symmetrix.

Восени 1990 року, коли EMC представила Symmetrix, що став легендою дискових масивів, модель 4200 ICDA мала ємність 24 Гбайт, кеш-пам'ять 256 Мбайт і контролер на базі 32-розрядного процесора. Symmetrix за кілька років вивів компанію на позицію провідного постачальника накопичувачів для мейнфреймів. По данимIDC, її частка на ринку накопичувачів для мейнфреймів зросла з 1% (в 1990 р.) до 42,5% (в 1996 р.).

Symmetrix був дешевий для мейнфреймів, але занадто дорогий для Unix-серверів і тим більше для x86 серверів, тому чимало компаній рвонуло в сегмент, що відкрився, ринку, вони запропонували продукти, що уступають Symmetrix по якості, але не настільки дорогі. Надалі на ринку з'явилася безліч моделей дискових масивів самого різного призначення.

Багаторівневе зберігання даних

Багаторівневе зберігання даних (Data multy tiering) можна розглядати як один з компонентів більше широкого давнього поняття віртуалізації пам'яті.

Термін virtual стосовно пам'яті й ЦОД виник в 1959 році для позначення віртуальної по своїй суті зовнішньої пам'яті на дисках, використовуваної для розширення внутрішньої пам'яті, що у ту пору збирали з магнітних сердечників.

Вона по визначенню була дуже маленької, але при цьому надзвичайно дорогою. Маленьку й дорогу пам'ять підмінювали прозорим для процесора способом більше дешевою дисковою пам'яттю незрівнянно більшого розміру. У сучасних системах зберігання точніше вести мову про інтеграцію зберігання, заміні фізичних адрес і номерів пристроїв логічними адресами й логічними номерами пристроїв і про більше ефективні методи керування.

Поява SSD дало новий імпульс до продовження робіт з віртуалізації, нинішній етап називають Automated Tiered Storage (AST), на ньому автоматично виконуються процедури DataTiering, тобто переміщення даних по рівнях зберігання.

Необхідність у міграції даних пов'язана із природою даних. Крива розподілу числа звертань до даних за часом нагадує гауссову криву – кількість звертань до свіжих даних, що вимагають швидкого доступу, невелико, у міру старіння даних воно зростає, а далі падає й до архівіруваних даних на повільних пристроям кількість обігів істотно менше пікового. Це властивість даних спонукає до створення багаторівневих ЦОД, на нинішньому рівні розвитку технології можна реалізувати 4-х рівневу модель: на 0 рівні – SSD, на них зберігаються найбільш затребувані дані; на 1 рівні – швидкі диски SAS; на 2 рівні – повільні диски SAS або SANA, на 3 рівні – стрічки. Прийнята колись трирівнева схема з дисків SAS, SATA і стрічок застаріла.

AST можна вважати розвитком раніше відомого керування ієрархічним зберіганням даних Hierarchical Storage Management (HSM), створеного в 1974 році для дискової бібліотеки IBM 3850, що разом з дисками вперше дозволила утворити єдиний простір даних. Можливо, використання нової назви відбиває прискорення процесів міграції до рівня реального часу, що дозволяє використовувати SSD.

AST – це процес перманентного переміщення даних між різними за вартістю пристроями відповідно до «температури» даних: чим дані гаряче, тим дорожче й відповідно швидше може бути пристрій, тобто SSD, а холодні дані

можна перемістити на стрічку. Для цього AST по заданих алгоритмах періодично переглядає дані й здійснює переміщення, керуючись температурою.

Варто розрізняти функції AST з тією роллю, що грає кеш-пам'ять на флеш, що підключається по NVMe. Принцип роботи кешу простіше, ніж AST, будь-який кеш є інструментом, у нього на час копіюється фрагмент із більше повільної пам'яті. Кеш – простий прискорювач, AST – оптимізує використання ресурсів ЦОД.

Варто розрізняти функції AST з тією роллю, що грає кеш-пам'ять на флеш, що підключається по NVMe. Принцип роботи кешу простіше, ніж AST, будь-який кеш є інструментом, у нього на час копіюється фрагмент із більше повільної пам'яті. Кеш – простий прискорювач, AST – оптимізує використання ресурсів ЦОД.

Робота з корпоративними даними є однією з найважливіших складових цифрових змін у компаніях. Ця робота вимагає наявності ефективних засобів, що підтримують інтерфейс між ієрархічно організованими багаторівневими системами зберігання, аналітичними й іншими технологіями, що безпосередньо служать цілям бізнесу. Такий інтерфейс дає можливість трансформувати пасивно зберігаються дані в найважливіший актив підприємства, що дозволяє витягати корисні для прийняття рішень знання з накопичених даних. Зі зростанням обсягу даних і появою великих даних, значення взаємозв'язку між даними й бізнесом багаторазово зростає.

Відповідаючи на запити, що виникають із боку сучасного бізнесу, компанія КРОК запропонувала власну концепцію «Розумне зберігання даних», відповідно до якої їхнє зберігання організується з обліком їхнього подальшого використання й можливості добування з них максимуму корисної інформації. Впровадження «Розумного зберігання даних» дозволяє одержати бізнес-переваги за рахунок більше ефективного використання корпоративної інформації. Технології, закладені в концепцію «Розумне зберігання даних», поширюється як на структуровані дані, що зберігаються в реляційних СУБД, так і на стрімко

зростаючі обсяги неструктурованих даних. Разом з партнером DellEMc КРОК надає можливість створювати продуктивну інфраструктуру для зберігання даних на базі лінійки масивів Dell EMC Unity. Завдяки гнучкості й простоті керування можна легко поєднувати хмарні середовища, можливості all flash і гібридних ЦОД для переходу на новий рівень цифрової трансформації.

«Розумне зберігання даних» підвищує економічну ефективність роботи з інформацією за рахунок її розподілу по сховищах, виходячи із затребуваності при одночасному дотриманні доступності даних для аналітичних систем. До того ж підтримка робочих процесів засобами «Розумного зберігання даних» дозволяє підвищити їхня надійність, оскільки дані зберігаються й обробляються в загальній і захищеній від збоїв середовищу. Перейти до нового підходу до зберігання даних за допомогою технологій Dell EMC можна максимально швидко й без капітальних витрат, скориставшись моделлю Hardware as a Service.

Програмний і апаратний RAID

Всі існуючі ЦОД діляться на ті, які використовують апаратний RAID і спеціалізоване ПЗ для розрахунку RAID – програмний RAID [6]. Останні системи є більше економічними. Тепер багато завдань обробки й зберігання даних значно ефективніше вирішуються в рамках ЦОД із програмним RAID. Наприклад – резервування системних дисків і віртуальних машин, зберігання й обробка відео, робота з великими файлами в системах документообігу.

Після лідерства програмного RAID на початку дев'яностих років на зміну йому прийшов апаратний, і донедавна саме він переважав на ринку ЦОД. Програмному RAID приділялася роль недорогих аматорських і домашніх систем зберігання. Зараз є клас завдань, яким цілком достатньо програмного RAID, надаваного безпосередньо ОС Windows, Unix і іншими. ЦОД із програмним RAID з категорії систем початкового рівня вийшли на корпоративний ринок.

Розвиток напрямку ЦОД із програмним RAID багато в чому визначають компанії, що випускають стандартні комплектуючі: процесори з новими убудованими командами, комутатори й кошики, що підтримують більше

продуктивні протоколи передачі даних. Серверні комплектуючі нового покоління і їхня приваблива ціна, інноваційні алгоритми розрахунку, – все це дозволило ЦОД із програмним RAID перевершити по характеристиках аналоги з апаратним RAID.

Виробники ЦОД із програмним RAID використовують всю міць нового покоління апаратних комплектуючих і на один-два роки випереджають виробників апаратних RAID-масивів по строках випуску нових моделей. У той час як виробникам апаратного RAID необхідно модернізувати виробничий процес, для ЦОД із програмним RAID досить протестувати новий кошик або процесор, – і нова модель готова до поставки.

Серед переваг програмного RAID можна відзначити високу продуктивність на платформі x 86-64, недорогі, доступні й взаємозамінні серверні комплектуючі, а також привабливу вартість обробки й зберігання даних. При цьому вартість модернізації системи буде досить низкою за рахунок покомпонентного відновлення апаратних і програмних засобів, а також їх значно великих функціональних можливостей. Програмний RAID дозволяє реалізувати шифрування на рівні коду процесора, наприклад, IntelCore i7). Подібні системи володіють підвищеної відказостійкісті N+2 і навіть N+3.

Про інтерес українських споживачів до ЦОД на основі програмного RAID свідчить ряд факторів. Великі українські інтегратори включили у свої пропозиції системи зберігання на основі програмного RAID. У прайс-аркушах українських збирачів серверів і систем зберігання подібні системи займають приблизно 20-30%. Власники ЦОДів розміщають ресурси на програмних RAID відповідно до практиків багаторівневого зберігання даних (див. далі).

Світовий ринок ЦОД

На тлі кризи й обсягів, що летять долілиць, продажів на світовому IT-ринку, сегмент систем зберігання даних залишається не тільки стабільним, але й росте.

Розроблювачі систем зберігання даних

Умовно всіх виробників ЦОД можна розділити на світових лідерів (бренди «А»), інших великих виготовлювачів (бренди «Б») і локальних (місцевих) збирачів. Донедавна на українському ринку ЦОД був представлений обмежений набір рішень, що базуються в основному на архітектурах DAS (Direct Access Storage) або SAN (Storage Area Networks) на основі протоколу Fibre Channel.

Компанії з першої групи завоювали найвищий авторитет на ринку. Вони мають у своєму розпорядженні широку партнерську й сервісну мережу по усьому світі, вкладають величезні кошти в розробку нової продукції й маркетинг, мають мільярдні фінансові оберти (зокрема, у секторі систем зберігання даних)і т.д. На продукцію провідних світових виробників (А-бренди) доводиться основна частка продажів зовнішніх дискових ЦОД, як у кількісному, так і в грошовому вираженні.

Ринок ЦОД переживає бурхливий підйом, і закономірно на ньому триває низка поглинань незалежних розроблювачів великими гравцями. Зокрема, Dell придбала Compellent Technologies, раніше HP поглинула компанію 3PAR (нітрохи поторгувавшись із тією ж Dell),EMC приєднала до своєї лінійки продуктів Kazeon і Isilon Systems.

Оскільки ЦОД невіддільні від обчислювальних ресурсів, те не дивно, що багато найбільших світових виробників систем зберігання є одночасно й лідерами на серверному ринку. З перерахованих вище виробників тільки три займаються винятково ЦОД – це EMC, Hitachi і NetApp.

Концепція публічних хмар впливає на сегмент ЦОД. Власники публічних хмар менш схильні до виплати бренд-премії, що може відкрити широкі можливості для виробників другого ешелону, нишевих або нових гравців [7].

Важливою відмінністю систем А-брендів від ЦОД місцевого виробництва є наявність у них спеціального ПЗ, призначеного для відновлення й захисту даних, резервного копіювання, віддаленого керування й моніторингу, «керування

життєвим циклом інформації» (Information Lifecycle Management, ILM), діагностики й т.д. ПЗ зі схожими функціями розробляє й безліч незалежних компаній, тому його можна придбати окремо. Звичайно, при відсутності проблем із сумісністю.

Вартість ЦОД дуже сильно залежить від функціональних можливостей і додаткових опцій – модулів розширення, типу жорстких дисків, сервісного обслуговування й т.д. [8].

Український ринок ЦОД

Український ринок систем зберігання даних розвивається надзвичайно динамічно в силу того, що він ще дуже молодий. Відсутність успадкованого устаткування не робить на нього значного впливу, оскільки через підривний ріст обсягів даних старі системи попросту не відповідають вимогам клієнтів і «вимиваються» значно швидше, ніж, наприклад, древні сервери й робочі станції.

Стрімке зростання обсягів даних всі частіше змушує вітчизняні компанії здобувати зовнішні дискові системи зберігання. Цьому в чималому ступені сприяє й традиційна тенденція зниження вартості IT-компонентів. Якщо раніше зовнішні ЦОД сприймалися тільки як атрибут великих організацій, те тепер потреба в цих системах не відкидають навіть невеликі компанії [9].

Тенденції й перспективи

Проблемою №1 для більшості великих корпоративних замовників стала на сьогоднішній день різнорідна інфраструктура ЦОД: організаціям нерідко доводиться підтримувати десятки ЦОД різних класів і поколінь від різних виробників, оскільки різні додатки висувають різні вимоги до зберігання даних. Так, критично важливим транзакційним системам (біллінговим, процесинговим, ERP і т.п.) потрібні висока надійність і продуктивність, властиві ЦОД верхнього цінового сегмента. Для аналітичних систем потрібні висока продуктивність і низька вартість розраховуючи на одиницю зберігання, тому для них резервуються ЦОД із твірдотільними дисками (SSD). А, наприклад, для роботи з файлами потрібні функціональність і низька вартість, тому тут застосовуються традиційні

дискові масиви. У різнорідній інфраструктурі рівень утилізації ЦОД виявляється низьким, загальна вартість володіння (ТСО) – непомірно високої, керованість – слабкої, до того ж складність такої інфраструктури зберігання, як правило, велика [10].

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Програмне забезпечення написане мовою Visual C#. Ця мова обрана виходячи з наступних міркувань. Visual C# – строго типізована об'єктно-орієнтована мова, призначена для розробки різноманітних безпечних і потужних додатків, виконуваних у середовищі .NET Framework. Мовою Visual C# можна розробляти звичайні клієнтські додатки Windows, веб-служби XML, розподілені компоненти, додатки типу " сервер-клієнт", додатки 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 додатків 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 застосунку – інкапсулюється у визначення класів. Клас може успадковувати безпосередньо з одного родового класу, але може реалізовувати будь-яке число інтерфейсів. Для методів, які перевизначають віртуальні методи в батьківському класі, необхідно ключове слово override, щоб виключити випадкове повторне визначення. У мові Visual C# структура схожа на полегшений клас: це тип, що розподіляється по стопках, що реалізує інтерфейси, але не підтримуюче спадкування.

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

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

– Властивості, що виступають у ролі методів доступу для закритих змінних-членів.

– Атрибути з декларативними метаданими про типи під час виконання.

– Вбудовані коментарі XML-документації.

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

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

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

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. Збірка містить маніфест із відомостями про типи збірка, версії, мови й регіональні параметри та вимоги безпеки.

При виконанні програми на Visual C# збірка 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 додатку мовою Visual C# бібліотека класів .NET Framework інтенсивно використовується для "устрою" коду.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кешування, флеш-технології та реплікації ЦОД.

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

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

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

в) розробити програмне забезпечення системи, що дозволить реалізувати

поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Цілком розумно очікувати, що повністю флеш-центри обробки даних стануть більш поширеними в майбутньому. Однак перехід до повністю флеш-центрів обробки даних може зайняти деякий час через такі фактори, як вартість, сумісність та вимоги до продуктивності.

Існують дві причини невизначено тривалого майбутнього дискових накопичувачів. Першою є різниця в ціні за терабайт ємності між флеш-накопичувачами та дисковими накопичувачами у п'ять-сім разів. Другою є потреба підприємств зберігати великі та постійно зростаючі обсяги даних, які активно не використовуються, для таких цілей, як навчання зі штучного інтелекту/машинного навчання та аналітики, архівування та дотримання вимог, а також резервне копіювання.

Потреба в дешевшому архівному сховищі, яке в багатьох місцях все ще включає стрічку, залишиться. Обсяг диска, який купують гіперскейлери, перевищує кількість флеш-накопичувачів сьогодні. Вам потрібне швидке сховище лише для того, над чим ви активно працюєте, а не для того, що ви просто зберігаєте для подальшого використання.

Незважаючи на прогнози щодо смерті стрічок протягом останніх двох десятиліть, використання стрічок – принаймні з точки зору обсягу даних, що зберігаються на них – збільшилося, а не скоротилося, через необхідність зберігати постійно зростаючі обсяги холодних або рідко використовуваних даних.

Як не парадоксально, нові технології, такі як штучний інтелект/модельне навчання, підсилюють цей аргумент. Компаніям потрібен доступ до величезних пулів сховищ для навчання машинному навчанню, але як тільки це буде

завершено, дані потрібно буде перенести на більш економічно ефективні технології зберігання. Тож, хоча це може здатися парадоксальною, можна навести вагомий аргумент, що нові програми штучного інтелекту фактично збільшують потребу в системах зберігання другого та третього рівнів, таких як диски та навіть стрічки.

Я сумніваюся, що коли-небудь з'являться повністю флеш-центри обробки даних з кількох причин. Завжди буде потреба в стрічках або дисках просто тому, що вони можуть зберігати величезні обсяги даних дешевим способом офлайн, і буде зростати обсяг даних, до яких рідко звертаються, але які все одно потрібно зберігати. І до того часу, як флеш-пам'ять захопить центри обробки даних, новіші NVM [незалежні пам'яті], такі як ReRAM та MRAM, почнуть займати частину центрів обробки даних.

Сьогодні економія коштів у 5 разів на флеш-пам'ять порівняно з диском робить більший простір для зберігання дисків, що використовуються в об'єктному сховищі, сильнішою альтернативою стрічці, а Seagate протягом наступних кількох років представить довгоочікувані дискові накопичувачі ємністю 30, 40 і 50 ТБ, що розширить можливості для дисків.

Однак, обираючи між дисковою та флеш-пам'яттю, IT-команди враховують не лише початкові витрати на придбання. Є один аспект переходу на флеш-технологію, який часто недооцінюється щодо її цінності для клієнтів: зі збільшенням продуктивності завдяки флеш-пам'яті з'являється цінність простоти. Під простотою я маю на увазі необхідність керування характеристиками пристрою для розміщення/розподілу даних з точки зору продуктивності. Це просто простіше, коли є вища продуктивність сховища. Це фактор, який спонукатиме організації до використання високопродуктивних технологій.

Однак деякі центри обробки даних вже повністю використовують флеш-пам'ять.

Відповідь на це питання про повністю флеш-центри обробки даних буде

залежати від розміру, масштабу та обсягу роботи центру обробки даних. Існує багато менших та середніх центрів обробки даних, які вже повністю використовують флеш-пам’ять. Але це, як правило, центри обробки даних для одного клієнта. Великі підприємства, регіональні хостинг-постачальники послуг та масивні гіперскейлери матимуть «живу іржавію» протягом будь-якого прогнозованого періоду час».

До 2028 року існує клас корпоративних центрів обробки даних, які будуть повністю флеш-сховищами. Це центри обробки даних, що експлуатуються підприємствами, які все частіше використовують хмарні інфраструктури публічної інфраструктури для розміщення своїх менш чутливих до продуктивності або критично важливих програм, що не потребують флеш-сховища та використовують ті ж хмари для зберігання своїх холодних або рідко використовуваних даних.

Локальні центри обробки даних, які здебільшого зосереджені на запуску критично важливих програм, швидко переходять на повністю флеш-сховище. Omdia прогнозує, що протягом наступних трьох-п’яти років більшість цих локальних центрів обробки даних повністю перейдуть на флеш-технологію.

Іншими словами, дискові накопичувачі переходять з корпоративних центрів обробки даних у гіперхмарні, де вони існуватимуть ще багато років через низьку вартість та відсутність потреби в продуктивності під час зберігання холодних даних. Хан навів ще одну причину, чому не варто використовувати флеш-пам'ять для зберігання цього типу даних: «Оскільки ці [сховища масового зберігання, архівування та резервного копіювання] часто взаємодіють з відносно повільним Інтернетом, пропускна здатність, а не низька затримка пошуку, є більш важливою». З тієї ж причини, чому пропускна здатність важливіша за випадковий доступ, він додає: «Основні випадки використання, такі як відео та мультимедіа, зможуть ефективно використовувати жорсткі диски протягом тривалого часу, а також інші технології, такі як IoT та ELT [Вилучення, завантаження, перетворення] конвеєрів збору даних.

Ще одним міркуванням є тенденція до продовження терміну служби IT-обладнання як частини стратегії екологічної стійкості та економії коштів. Тому клієнти зараз менш охоче позбавляються від технології, яка здається старою, та замінюють її блискучою новою. Вплив на центри обробки даних полягатиме в використанні низки технологій, яким у деяких випадках може бути до семи років, а це означає, що поява повністю флеш-центрів обробки даних не є найближчою перспективою.

Вважаємо, що існує межа, де розгортання з ємністю менше заданої мають сенс як повністю флеш-пам'яті, а розгортання вище цієї межі можуть бути недоцільними. Ця межа повільно рухатиметься вгору, але, на нашу думку, здебільшого йтиме в ногу зі зростанням потреб у ємності, тому центри обробки даних, що використовують лише флеш-пам'ять, залишаться назавжди «через два роки».

Компанія може бути рада зберігати відносно невеликі файлові системи обсягом 200 ТБ у флеш-пам'яті, оскільки це коштуватиме лише приблизно на 150 000 доларів більше, ніж зберігання на диску, і працюватиме набагато краще. Але для зберігання в 100 разів більшої ємності, 20 ПБ, додаткові витрати становитимуть 12 мільйонів доларів, що важко виправдати. Якщо у вас немає якихось особливих вимог, ці гроші краще було б використати для придбання процесорів та графічних процесорів.

Розвиток технологій часто слідує кривій, за якою темпи покращення вартості або продуктивності повільно зменшуються з часом, слідуючи кривій, яка поступово стає більш пологою, оскільки технічний прогрес стає дедалі складнішим для досягнення. Дійсно, приблизно до 2010 року багато спостерігачів передбачали, що технічний розвиток NAND-флеш-пам'яті ось-ось зіткнеться з проблемою кількості комірок пам'яті, які можна упакувати в один флеш-чіп. На той час флеш-пам'ять була добре утвердженою та зростаючою рисою корпоративного IT-ландшафту не лише завдяки своїй продуктивності та іншим перевагам порівняно з диском, але й тому, що її ціна падала протягом

попереднього десятиліття. Якби виробники флеш-чіпів зіткнулися з цією прогнозованою технологічною проблемою, ціни почали б падати набагато повільніше в перерахунку на одиницю ємності пам'яті.

Однак у 2013 році Samsung обійшла передбачуване обмеження, випустивши перші так звані 3D-флеш-чіпи, які складалися з кількох шарів комірок пам'яті, а не з одного шару комірок, що використовувався раніше. Це означало більше комірок пам'яті на чіп, і як надзвичайно цінний побічний ефект, можливість зберігати більше бітів даних у кожній комірці пам'яті, що знову ж таки знизило ціни за терабайт. Усі основні виробники флеш-пам'яті невдовзі наслідували приклад Samsung, і з того часу кількість шарів на чіп швидко зростала. Але це було десять років тому. Чи наближається флеш-пам'ять зараз до кінця чи до більш пологої частини своєї технологічної кривої?

Люди, які кажуть: «Закон Мура мертвий», ігнорують 3D NAND. Ця технологія дала флеш-пам'яті NAND новий механізм для подальшого додавання бітів до чіпа, і щороку інженери-технологи знаходять геніальні способи просунути його далі, ніж хтось міг би вважати можливим. Це далеко не те, щоб сказати «ні» на це питання. Очікуйте побачити щонайменше ще кілька порядків зниження вартості протягом наступних кількох років, оскільки щільність чіпів продовжуватиме зростати.

Виробники флеш-пам'яті, такі як Samsung, SK Hynix, Kioxia, Western Digital та Micron, продовжуватимуть впроваджувати інновації з дорожніми картами для більшої щільності з більшою кількістю шарів, використовуючи методи стекування, інновації в архітектурі та дизайні, а також більше бітів на комірку (наприклад, п'ятирівнева комірка або ПЛК). Перші багатошарові флеш-чіпи, що вийшли на ринок у 2013 році, складалися з 24 шарів комірок пам'яті та зберігали 128 Гбіт/с. Ю згадав про демонстрацію SK Hynix цього року 321-шарового чіпа, що зберігає 1 Тбіт/с, а також прогноз Samsung, зроблений минулого року, що до 2030 року вона поставить 1000-шарові чіпи.

Дискова технологія також все ще розвивається, і тому ціни на диски також

продовжуватимуть падати приблизно з тією ж швидкістю, що й на флеш-пам'ять. «Флеш-технологія продовжить свою невблаганну криву вдосконалення, але ми не бачимо, щоб ця крива прискорювалася, щоб набирати обертів на дисках, тобто знижувала множник 5x-7x ($/ТБ) або уповільнювала його порівняно з дисками.

Є кілька факторів, деякі з яких були окреслені в попередніх відповідях, які свідчать про те, що флеш-пам'ять все ще перебуває на підйомі з точки зору технологічної кривої. Існує аргумент, який слід враховувати, де поточна технологія флеш-пам'яті переходить у нові технології, проте, враховуючи рівні розвитку, які все ще досягають компанії, що розробляють флеш-сховища, це, здається, свідчить про те, що ще є багато можливостей для просування вгору, перш ніж буде досягнуто плато.

Флеш-пам'ять має багато технологічних переваг не лише всередині чіпів

Ми тільки починаємо бачити, як флеш-пам'ять можна використовувати в центрах обробки даних. Цінність комунікаційних протоколів NVMe тільки починає визнаватися в центрах обробки даних, і знадобиться щонайменше десятиліття, щоб вони замінили масивну інфраструктуру на основі SCSI. Крім того, еволюція інтерфейсу PCIe та нові технології, такі як CXL, відкриють нові можливості для реалізації флеш-сховищ. Крім того, транспортні протоколи, такі як NVM-oF, зазвичай через RDMA Ethernet, тільки починають об'єднуватися як життєздатна альтернатива. Тож, замість того, щоб досягти піку, я вважаю, що ми тільки починаємо бачити перші кроки революції флеш-технологій.

Флеш-пам'ять трансформувала корпоративне сховище даних і була головною рушійною силою, яка рушійною силою революції мобільних обчислень. Зараз це основна технологія. Окремо від виробництва продуктів на базі флеш-пам'яті, таких як накопичувачі або повноцінні системи зберігання даних, виробництво лише NAND-флеш-чіпів зараз генерує близько 80 мільярдів доларів річного доходу, і ця цифра продовжує зростати.

Це викликає очевидне питання: коли з'явиться наступна нова технологія

твердотільного зберігання даних з таким самим впливом на масовий ринок? Протягом останніх кількох десятиліть у дослідницьких лабораторіях було витрачено мільярди доларів на спроби знайти іншу таку технологію. Спільно розроблена Intel та Micron пам'ять Optane стала результатом таких досліджень і вперше була поставлена у твердотільних накопичувачах у 2017 році. Швидша, але дорожча за флеш-пам'ять, Optane була проголошена першою з майбутнього класу так званих пам'ятей класу сховищ (SCM), які мали доповнити або замінити флеш-пам'ять і мати так само великий загальний вплив на IT. Однак Optane продавалася погано, і у 2021 році Intel оголосила про свій план припинити виробництво цієї пам'яті, лише через чотири роки після її перших поставок.

Тим часом дослідження інших потенційних SCM тривають. Для Objective Analysis, яка зосереджена на нових моделях пам'яті, Генді сказав, що протягом цього та наступного десятиліття, ймовірно, не з'явиться нових технологій пам'яті з потенціалом такого ж впливу, як флеш-пам'ять. Він розрізняв два типи пам'яті: ті, що вбудовані в процесори або інші чіпи, та ті, що, як NAND-флеш та Optane, продаються або продавалися як чіпи лише для дискретної пам'яті у набагато більших кількостях і тому мають набагато більший ринковий потенціал.

Optane зазнав невдачі через свою вартість. Ми попереджали про це, щойно його оголосили. Але інші технології, ймовірно, процвітатимуть на певних ринках, особливо як вбудована пам’ять у мікроконтролерах, ASIC та інших SoC [процесорах Systems-on-a-Chip]. Однак дискретні мікросхеми пам’яті навряд чи масово перетворяться на нові SCM у 2020-х роках, і, ймовірно, не у 2030-х роках.

Optane, також відомий як 3D XPoint, зазнав невдачі з економічних причин, але створення альтернативи флеш-пам'яті неминуче:

Кілька технологій NVM, включаючи ReRAM, MRAM, PCM та FRAM, стають потенційними альтернативами флеш-пам’яті. 3D XPoint від Intel була першою спробою вирішити цю проблему, але не увінчалася успіхом значною мірою з економічних причин. Ключем до успішної альтернативи флеш-пам’яті є розробка пам’яті, яка може масштабуватися до достатньо великої щільності, але

за достатньо низькою ціною. Intel змогла впоратися лише з частиною цієї проблеми, пов’язаною з щільністю. Це лише питання часу, коли ми побачимо технологію, яка зможе відповідати обом критеріям, і ми вважаємо, що ReRAM буде відповіддю, оскільки вона має фундаментальні технічні переваги, включаючи швидкість, енергоефективність та вартість. Триває розробка, щоб довести технологію до ще більшої щільності.

За 10 років може статися багато чого – сказати, що станеться, легко (наприклад, атомну пам’ять, яку досліджує IBM); сказати коли – набагато важче.

Технологічний імпульс та варіант флеш-пам'яті SLC стануть перешкодою для будь-якої нової пам'яті.

Нездатність досягти достатніх продажів, щоб виправдати обсяги виробництва, необхідні для забезпечення життєздатно низьких цін, призвела до скасування Optane. Відносно нові високошвидкісні варіанти флеш-пам'яті SLC [single-level cell] виконують завдання, для яких була призначена Optane. Легко уявити, де пам'ять класу сховища може вписатися в піраміду пам'яті сховища, але було важко забезпечити правильне поєднання характеристик продуктивності, затримки та вартості, забезпечуючи при цьому збереження даних у реальному світі. Optane SCM здавався гарним проєктом, але його виробництво згортається через відсутність належної економіки обсягу. Очевидно, що є кілька хороших варіантів використання для Optane SCM, але, чесно кажучи, вони вирішуються за допомогою нещодавно випущеної технології NAND-флеш SLC. Пропозиції SLC NAND SSD набирають популярності завдяки своїй фантастичній довговічності та хорошій продуктивності запису, особливо для багаторівневого створення масивів даних та кешування даних.

«Будь-яка нова технологія має перевершити існуючу технологію, яка виграє від десятиліть оптимізації та великомасштабного виробництва. Отже, нова технологія буде у значно невигідному становищі.

SCM мали/мають величезний потенціал, але зміна архітектури програмного забезпечення, необхідна для того, щоб програми отримали всі

переваги, які може запропонувати SCM, була занадто високою, тому програми так і не прийняли їх. Флеш-пам'ять не зіткнулася з цією перешкодою, оскільки її специфічне поєднання вартості та продуктивності ніколи не вимагало та не виправдовувало її використання як доповнення до пам'яті DRAM.

3.2 Розробка структурної схеми

Крім планування оптимального використання дискової підсистеми, деякі ЦОД дозволяють вибрати й розмір кешу. Про те, як використовується кеш у системі зберігання даних і скільки його мабуть, існує багато думок. Одне з найважливіших призначень кешу – зберігання метаданих ЦОД. Для реалізації будь-якої функціональності ЦОД, будь те дискові групи, логічні томи, багаторівневе зберігання або реплікація, необхідні службові дані – це і є метадані. Вони постійно використовуються мікрокодом ЦОД, і для їхнього зберігання потрібний самий швидкодіючий носій інформації в системі. Їм і є кеш, для реалізації якого використовується оперативна пам'ять рівня DDR3 або DDR4, чия продуктивність на порядок вище, ніж у самого швидкісного флеш-диска.

Друга важлива функція кешу – безпосереднє кешовані даних. Оброблювані хостами дані розміщаються в кеші й можуть використовуватися повторно без звертання до більше повільних дисків. Крім того, кешовані дозволяє оптимізувати запис даних на диски.

Однак використання кешу ефективно не для всіх профілів навантаження. Дані, які послідовно зчитуються із системи зберігання, кешуються добре – це так зване лінійне читання. Навіть якщо додаток зчитує дані, наприклад, дрібними блоками по 8 Кбайт замість що рекомендуються 256 Кбайт і більше, система зберігання розпізнає лінійне читання й, знаючи, які дані будуть запитані наступними операціями вводу-виводу, зчитує їх заздалегідь. Таким чином, що випливає операція вводу-виводу не викликає звертання до більше повільної дискової підсистеми – дані надходять із кешу, що істотно прискорює вводу-вивід.

У той же час кеш майже марний, коли мова йде про випадкове читання (random read) даних зі ЦОД. У цьому випадку ймовірність знаходження потрібних даних у кеші прагне до нуля, оскільки обсяг кешу в системі зберігання на порядок менше, ніж максимальний корисний обсяг даних сучасних ЦОД. Саме тому кілька років назад з'явилася ідея використовувати флеш-диски для прискорення випадкового читання.

Флеш-технології дозволяють оптимізувати й інші типи навантажень. Однак приріст продуктивності, у порівнянні з використанням механічних дисків, виходить не настільки значним, тоді як різниця у вартості зберігання 1 Тбайт даних на флеш-диску й механічному диску поки залишається досить відчутної.

При будь-яких операціях запису дані спочатку містяться в кеш і лише після цього записуються на диски. Крім повторного використання для наступних операцій вводу-виводу, нові дані можуть групуватися спеціальним образом, але тільки в тому випадку, якщо надалі вони будуть записані на групи RAID з парністю (RAID5 і RAID6).

Чому важлива оптимізація запису при використанні RAID-груп з парністю? Вся справа в тому, що в них застосовується особливий захист даних від збою диска в групі. Якщо, допустимо, робити запис на масив RAID5 випадковим образом, без оптимізації, то одна операція запису, зроблена хостом, буде генерувати чотири операції вводу-виводу на системі зберігання (два читання й два записи). Для RAID6 цей показник дорівнює вже шести. У результаті продуктивність ЦОД серйозно знизиться.

Що в цьому випадку може почати система зберігання даних? У першу чергу вона спробує сформувати в кеш-пам'яті набір даних (full stripe), щоб зробити розрахунок парності без додаткового звертання до дисків і записати відразу всі дані на диски. При послідовному записі зібрати full stripe у кеш-пам'яті досить легко, але при випадкової – система або буде чекати, поки в кеші не збереться необхідний набір даних, або почне зчитувати відсутні блоки даних з дисків. Тому кешовані запису даних практично у всіх випадках прискорює вводу-вивід.

Існує думка, що, якщо збільшити обсяг кешу, ЦОД стане працювати швидше. Як правило, між корисним обсягом збережених даних і ємністю кешу є деяке визначене співвідношення, що бажано витримувати. Однак це повністю здійснено тільки в системах зберігання старшого класу, де можливий більше гнучкий вибір обсягу кешу. У системах зберігання середнього класу не завжди вдається розширити кеш до бажаних величин.

Якщо ЦОД не справляється із читанням або записом даних, справа аж ніяк не в кеші. Швидше за все, обрана невідповідна модель ЦОД або неправильно зроблена конфігурування дискової підсистеми, який просто не вистачає ресурсів для читання й запису всіх даних, що надходять,. Точна оцінка планованого навантаження й вибір потрібної конфігурації ЦОД є ключовими критеріями для одержання бажаної продуктивності IT-системи. Тому збільшення обсягу кешу не панацея для підвищення продуктивності.

Особливості флеш-накопичувачів

Говорячи про продуктивність, неможливо не торкнутися докладніше теми флеш-технологій. Якщо обчислювальна потужність контролерів ЦОД не є вузьким місцем, то використання флеш-дисків завжди дозволяє збільшити продуктивність доступу до даних при будь-якому профілі навантаження – лінійному й випадковому, при читанні й при записі. Однак треба брати до уваги й фінансову складову: при послідовному читанні й записі флеш-диски можуть працювати в кілька разів швидше механічних, але вартість зберігання 1 Тбайт даних буде в десятки разів вище. Тому для лінійних навантажень звичайно використовуються самі повільні механічні диски NL-SAS, оскільки навіть вони прекрасно справляються з поставленим завданням.

Флеш-технології дуже ефективні при випадковому профілі вводу-виводу – особливо при випадковому читанні, коли кеш нічим допомогти не може. Вони відрізняються не тільки можливістю здійснювати більше операцій вводу-виводу (IOPS), але й низьким часом відгуку таких операцій: дві мілісекунди й менше. Саме тому при високих вимогах до продуктивності постачальники рішень

пропонують флеш-диски в складі ЦОД або систему, повністю оснащену флеш-дисками (All Flash Array, AFA).

У той же час флеш-диски «не люблять» операції запису – дані на них записуються повільніше, ніж зчитуються. Це пов'язане з використанням пам'яті NAND. На відміну від механічних дисків, на які можна довільно записувати й перезаписувати будь-які дані, флеш-диск потрібно спочатку очистити. Причому не можна стерти один біт або байт – дані стираються блоками по 1-2 Мбайт залежно від типу диска, виробника й інших параметрів. Попередньо необхідно перенести всі актуальні дані у вже чисті (стерті) блоки. Таким чином, якщо при занадто інтенсивному записі місце в блоках звільняється недостатньо швидко, флеш-диск починає «гальмувати».

Флеш і синхронна реплікація

Чи ефективні флеш-диски при віддаленій синхронній реплікації? Дані реплікуються на віддалену площадку для забезпечення їхньої схоронності й доступності на той випадок, якщо основна ЦОД повністю вийде з ладу. Як правило, площадка розташовується досить далеко, і тому до звичайних затримок вводу-виводу додається ще й затримка від реплікації.

Існує думка, що флеш-диски неефективні при такому виді реплікації. Це пов'язане з тим, що синхронна реплікація може звести нанівець одну з основних особливостей флеш-дисків – дуже невелику затримку вводу-виводу. Насправді додаткові затримки в основному стосуються тільки операцій запису, а їхня частка в профілі навантаження застосунку, як правило, не перевищує 20-30%. Таким чином, навіть при синхронній реплікації флеш-диски будуть досить ефективне і їхнє використання в подібних конфігураціях цілком виправдано.

Необхідна конфігурація для синхронної реплікації

Чи потрібні при віддаленій синхронній реплікації дві однаково зконфігуровані ЦОД? Це залежить від виду синхронної реплікації (Active/Active або Active/Passive), розподілу навантаження вводу-виводу між двома центрами обробки даних, вимог SLA і інших параметрів. Інтенсивність навантаження на

ідентичні набори даних, що зберігаються на віддалених площадках, завжди буде різною. У випадку синхронної реплікації Active/Active (див. рисунок 3.1) навантаження буде залежати від балансування вводу-виводу застосунку між двома ЦОДами. Якщо використовується Active/Passive, копія основних даних перебуває чекаючи «години ікс», коли основний набір стане недоступний і їй буде потрібно відповідати за все вводу-вивід. І в тому і в іншому випадку для зберігання додаткового набору даних потрібні ресурси: дискові, обчислювальні й т.д.

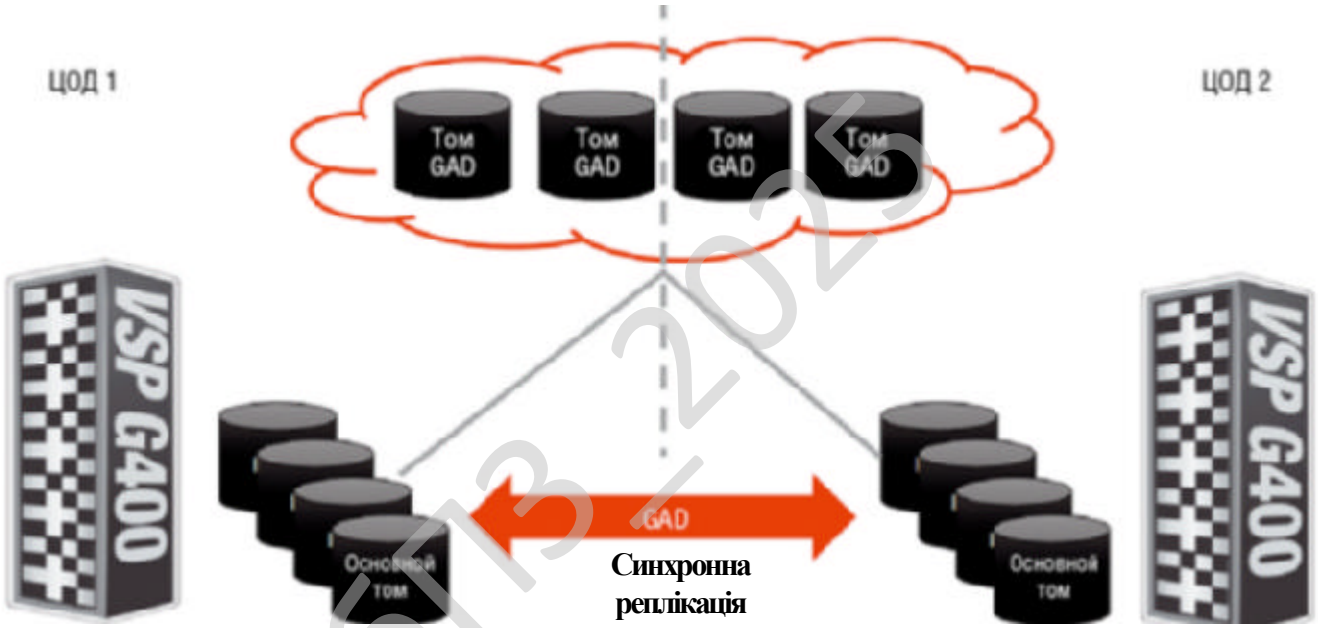

Рисунок 3.1 – Структурна схема системи

Необхідність використання флеш-дисків і однакових обчислювальних контролерів ЦОД для обох наборів даних залежить від того, яку продуктивність вводу-виводу повинна забезпечити що залишилася ЦОД після виходу з ладу основної системи. Якщо до усунення аварії допускаються зниження продуктивності ЦОД і погіршення роботи IT-системи, то для зберігання другого набору даних досить ресурсів з меншою продуктивністю.

Як правило, такі схеми застосовуються в тих випадках, коли фінансові втрати від погіршення роботи IT-системи менше вартості зберігання другого

набору даних на високопродуктивних ЦОД. Якщо ж відмова одного із ЦОДів приведе до фінансових втрат, що перевищують вартість зберігання копії даних на високопродуктивної ЦОД, то використання тих же самих ресурсів для зберігання нехай навіть неактивної копії даних обов'язково.

Продуктивність СХД при збоях

Продуктивність ЦОД при апаратних збоях – не менш важливий момент, на який IT-фахівці не завжди звертають увагу при виборі конфігурації ЦОД. Продуктивність ЦОД завжди оцінюється в той момент, коли всі її складові функціонують у штатному режимі. Але так буває не завжди: іноді вихід з ладу одних компонентів ЦОД не робить ніякого впливу на продуктивність, у той час як поломка інших приводить до її істотного зниження.

Наприклад, при виході з ладу обчислювального контролера у двоконтролерної ЦОД продуктивність може знизитися більш ніж у два рази. Це пов'язане із забезпеченням захисту цілісності даних: при роботі на запис кеш переводиться в режим Write-through, що поряд із втратою обчислювальної потужності одного контролера приводить до подальшого зменшення продуктивності ЦОД.

Таким чином, щоб при виході з ладу будь-якого компонента продуктивність як і раніше відповідала певним вимогам, у конфігурації ЦОД доводиться передбачати додаткові ресурси, які в штатному режимі роботи ЦОД використовуватися не будуть. Якщо ж передбачається застосування віддаленої синхронної реплікації для захисту від збою ЦОД, у випадку сильного зниження продуктивності основний ЦОД, викликаного збоєм одного з її компонентів, ввод-вивід IT-системи може перемикатися на резервну площадку.

Проблема вибору конфігурації

Вибір конфігурації як ЦОД, так і рішення в цілому залежить від вимог до продуктивності IT-системи й від можливих фінансових втрат при погіршенні якості її роботи.

Ми торкнулися далеко не всі аспекти вибору конфігурації ЦОД і планування її експлуатації, пов'язані із продуктивністю. Існує безліч різних методик оптимізації конфігурації рішення, у яких ураховуються вихідні вимоги, тип застосунку й багато чого іншого. Більше того, зі зміною підходу до зберігання даних – від приватних сховищ до хмарних, від блокового зберігання даних до файлового й об'єктного зберігання – будуть трансформуватися й методики розрахунку параметрів сховищ даних, що забезпечують необхідні експлуатаційні характеристики. Однак можна із упевненістю затверджувати, що тема продуктивності зберігання даних була, є й буде важливої складової при виборі будь-якого сучасного IT-рішення.

3.3 Розробка функціональної схеми

Для реалізації системи кешування, флеш-технології та реплікації ЦОД, пов’язаної з перешкодостійким зберіганням інформації застосуємо алгоритм Ріда-Соломона.

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

У цей час широко використовується в системах відновлення даних на різних носіях, у тому числі й у ЦОД, при створенні архівів з інформацією для відновлення у випадку ушкоджень, у завадостійкому кодуванні.

Код Ріда-Соломона був винайдений в 1960 році співробітниками лабораторії Лінкольна Массачуссетского технологічного інституту Ірвином Рідом і Густавом Соломоном. Ідея використання цього коду була представлена в статті «Polynomial Codes over Certain Finite Fields». Перше застосування код Ріда-Соломона одержав в 1982 році в серійному випуску компакт-диск ЦОДів. Ефективний алгоритм декодування був запропонований в 1969 році Елвином

Берлекемпом і Джеймсом Месси.

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

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

Поліном, що породжує

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

Теорема 3.1

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

Теорема 3.2

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

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

Матриця, що породжує

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

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

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

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

Матрицю G можна записати в символьній формі:

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

Перевірочна матриця

Для кожного кодового слова циклічного коду справедливо $c(x) = 0 \mod g(x)$. 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:

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

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

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

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

відстанню (є оптимальним у змісті границі Синглтона).

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

Властивості

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

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

Виправлення багаторазових помилок

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

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

Практична реалізація

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

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

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

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

Кодування

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

– До вихідного слова приписуються $2t$ нулів, виходить поліном $T = Sx^{2t}$.

– Цей поліном ділиться на поліном, що породжує, G, перебуває залишок R, $Sx^{2t} = QG + R$, де Q – частка.

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

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

Наведений як приклад кодер Ріда-Соломона генерує 16 коригувальних

байт, що дозволяє виправляти до 8 і виявляти до 16 помилок у кадрі даних.

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

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

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

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

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

Символу «C» 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 EAB. 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 EAB, 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 EAB, використовуючи 4 пари "суматор – EAB", можна тактувати кодер із частотами, що перевищують байтову частоту не в 8, а в 4 рази, що дозволить підняти її до 25...30 Мгц.

Декодування

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

– Обчислює синдром помилки.

– Будує поліном помилки.

– Знаходить корінь даного полінома.

– Визначає характер помилки.

– Виправляє помилки.

Обчислення синдрому помилки

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

Побудова полінома помилки

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

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

Алгоритм Евкліда виконується наступним чином.

Нехай a і b суть цілі числа, не рівні одночасно нулю, і послідовність чисел

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

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

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

Існування таких $r_1,r_2,...$, 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.

Застосування

У даний момент коди Ріда-Соломона мають дуже широку область

застосування завдяки їхній здатності знаходити й виправляти багаторазові пакети помилок.

Код Ріда-Соломона використовується при записі й читанні в контролерах оперативної пам'яті, при архівуванні даних, запису інформації на жорсткі диск ЦОДи (ECC), запису на диск ЦОДи. Навіть якщо ушкоджено значний обсяг інформації, зіпсовано кілька секторів диск ЦОДового носія, то коди Ріда-Соломона дозволяють відновити більшу частину загубленої інформації. Також використовується при записі на такі носії, як магнітні стрічки й штрихкоди.

Можливі помилки при читанні з диску ЦОДу з'являються вже на етапі виробництва диск ЦОДу, тому що зробити ідеальний диск ЦОД при сучасних технологіях неможливо. Так само помилки можуть бути викликані подряпинами на поверхні диск ЦОДу, пилом і т.д. Тому при виготовленні компакт-диск ЦОДу, що читається, використовується система корекції CIRC (Cross Interleaved Reed Solomon Code). Ця корекція реалізована у всіх пристроях, що дозволяють зчитувати дані з CD диск ЦОДів, у вигляді чипа із прошиванням firmware. Знаходження й корекція помилок заснована надмірності й перемежені (redundancy & interleaving). Надмірність приблизно 25% від вихідної інформації.

При кодуванні на першому етапі відбувається додавання перевірочних символів до вихідних даних, на другому етапі інформація знову кодується.

Крім кодування здійснюється також перемішування (перемеження) байтів, щоб при корекції блоки помилок розпалися на окремі біти, які легше виправляються. На першому рівні виявляються й виправляються помилкові блоки довжиною один і два байти (один і два помилкових символи відповідно). Помилкові блоки довжиною три байти виявляються й передаються на наступний рівень. На другому рівні виявляються й виправляються помилкові блоки, що виникли в C2, довжиною 1 і 2 байти. Виявлення трьох помилкових символу є фатальною помилкою, не можуть бути виправлені.

Цей алгоритм кодування використовується при передачі даних по мережах WiMAX, в оптичних лініях зв'язку, у супутниковому й радіорелейному зв'язку. Метод прямої корекції помилок у минаючому трафике (Forward Error Correction,

FEC) ґрунтується на кодах Ріда-Соломона.

Над вхідними даними, перед записом у ЦОД, відбуваються перетворення кодеком Ріда-Соломона. Після кодування дані записуються у дата-центр. У якості носія окрім дата-центру може використовуватися любий інший носій інформації. Після цього інформація зберігається на носієві.

При зчитуванні інформації, розроблене програмне забезпечення декодує інформацію, яка зберігається на носієві, і якщо потрібно, після проведення відповідних перевірок, проводить відновлення втраченої інформації. Якщо таке відновлення неможливе, то програма видає відповідне повідомлення.

На рисунку 3.2 зображена функціональна схема системи. З неї бачимо, що розроблена система підвищення стійкості до уражень інформації яка записана на диск ЦОДах складається з наступних основних функціональних блоків:

– сам носій інформації – диск ЦОД;

– кодер Ріда-Соломона, який використовується, коли відбувається запис інформації на диск ЦОД, при цьому необхідно враховувати, що об’єм інформації, яка записується на диск ЦОД, повинна бути меншою, ніж об’єм диск ЦОДу, в зв’язку, з тим, що коди Ріда–Соломона відносяться до кодів з надмірністю, за рахунок якої й відбувається кодування;

– декодер Ріда-Соломона, який використовується при читанні даних с відповідного диск ЦОДу.

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

– дані, які потрібно зберегти у ЦОД;

– поліном для кодування;

– завадостійке кодування Ріда-Соломона.

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

або спеціалізованого програмного забезпечення.

Кодове слово Ріда-Соломона формується із залученням спеціального полінома. Всі коректні кодові слова повинні ділитися без залишку на ці утворюючі поліноми. Загальна форма утворюючого полінома має вигляд: $g(x) = (x-a^{i})(x-a^{i+1})\ldots(x-a^{i+2t})$, а кодове слово формується за допомогою операції: $c(x) = g(x).i(x)$, де $g(x)$ є утворюючим поліномом, $i(x)$ являє собою інформаційний блок, $c(x)$ – кодове слово, що називається простим елементом поля.

2t символів парності в кодовому слові Ріда-Соломона визначаються з наступного співвідношення: $p(x) = i(x)\cdot x^{n-k} \bmod g(x)$.

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

Цей функціональний блок містить у собі наступні блоки:

– блок обчислення поліномів синдрому та помилок;

– блок обчислення локації та значень помилок;

– блок корекції помилок;

– відновлені за допомогою коді Ріда-Соломона дані.

Декодер працює наступним чином.

Введемо позначення:

– r(x) – Отримане кодове слово.

– Si – Синдроми.

– L(x) – Поліном локації помилок.

– Xi – Положення помилок.

– Yi – Значення помилок.

– c(x) – Відновлене кодове слово.

– v – Число помилок.

Отримане кодове слово r(x) являє собою вихідне (передане) кодове слово c(x) плюс помилки: $r(x) = c(x) + e(x)$.

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

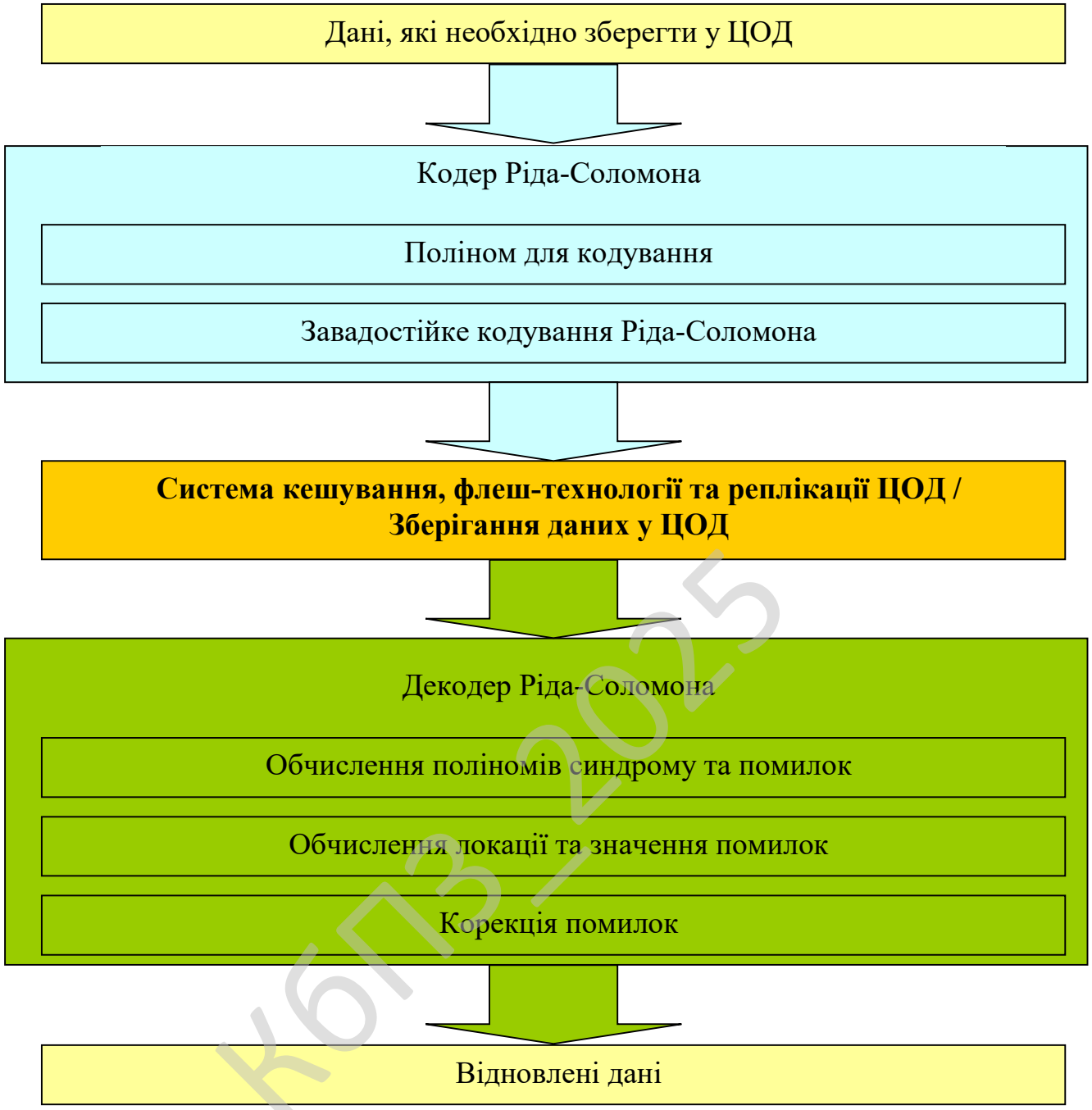

Рисунок 3.2 – Функціональна схема системи

Обчислення синдрому

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

Знаходження позицій символьних помилок

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

1. Визначення полінома локації помилок

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

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

Знаходження значень символьних помилок

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

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

3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей. Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

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

Діаграми потоків даних містять чотири типи елементів:

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

– Сховища даних (репозиторії).

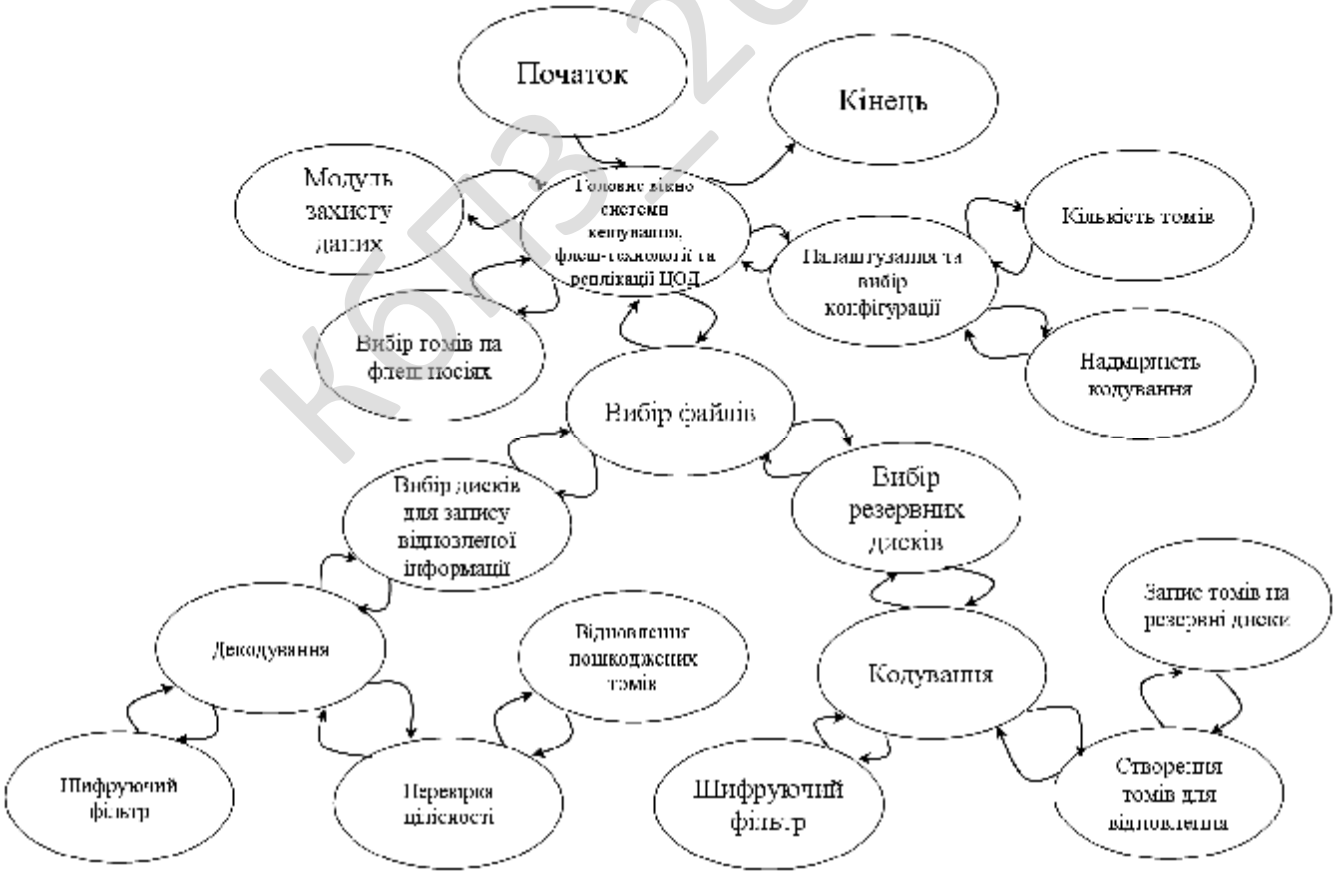

Рисунок 3.3 – Діаграма взаємодії процесів

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо реалізацію магістерської дипломної роботи. Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень.

Було створено блок-схеми роботи системи. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується. Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо.

Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

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

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

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

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач. Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції. Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу.

Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується.

Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи. Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані. Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи). З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми.

Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає високого рівня декомпозиції задач на класи.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підсистеми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

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

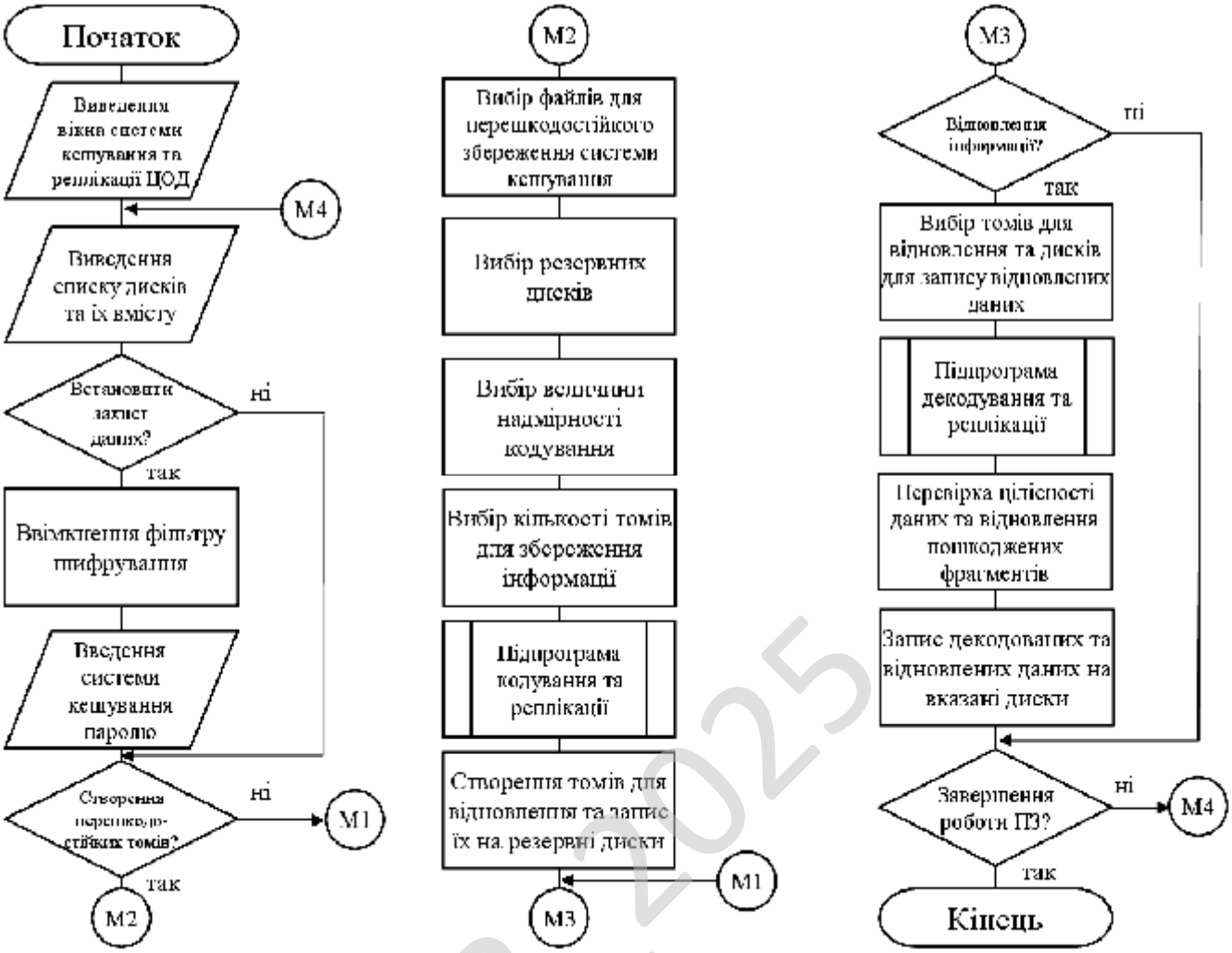

Рисунок 4.1 – Блок-схема основної програми

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

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії.

Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності.

Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

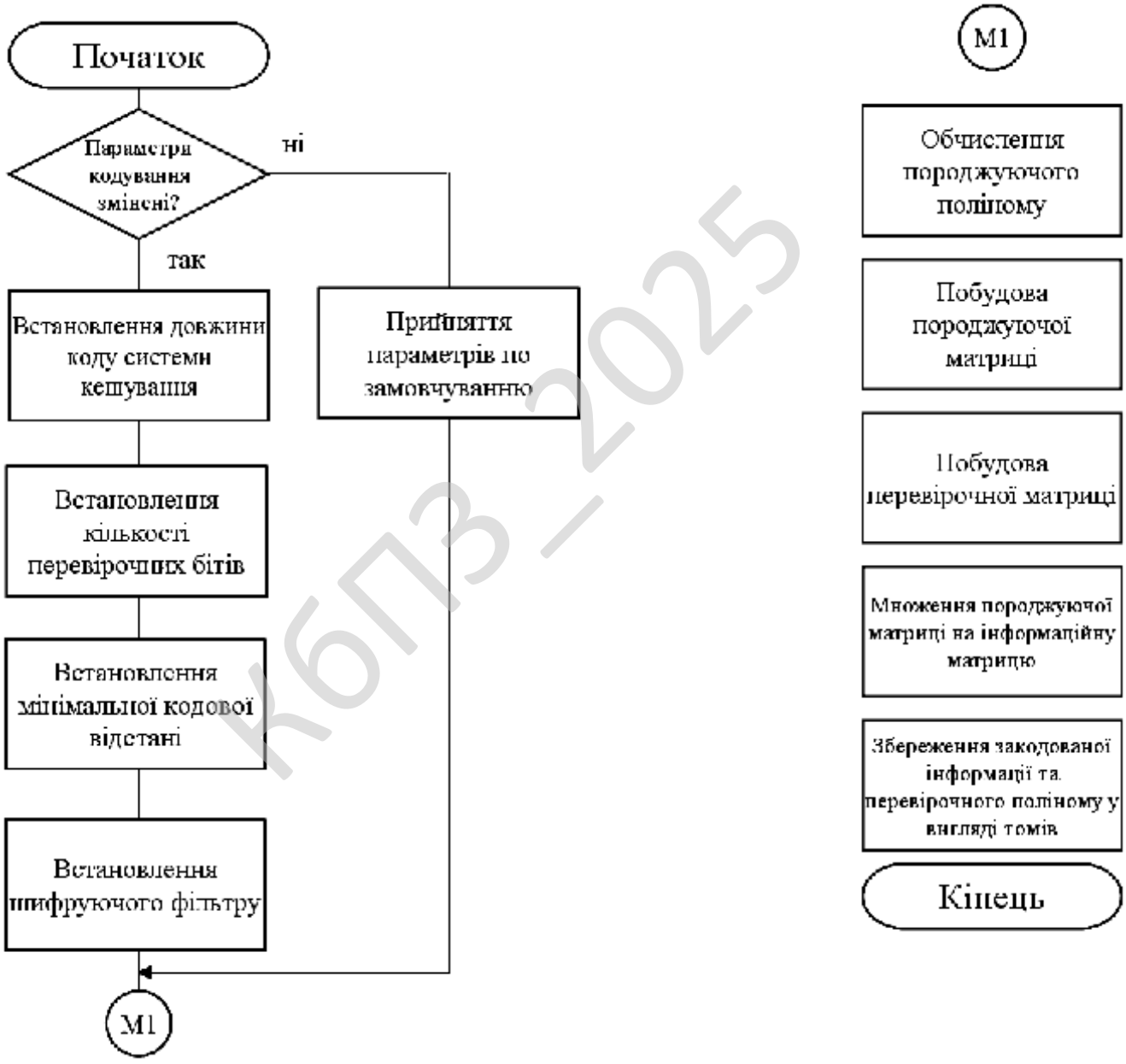

Рисунок 4.2 – Блок-схема роботи підпрограми

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

Діаграма розгортання (deployment diagram) це діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

Нижче наведемо ту частину коду, яка виконує вищеперераховані дії.

```
D1_RS()
{
        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]
// тобто складаємо черговий "захоплений" символ вихідних
// даних з молодшим символом біт парності (відповідного
// "регістру" b2 t-1, (рисунок 4.2) і переводимо його в
// індексну форму, зберігаючи результат у регістрі feedback;
// як ми вже говорили, сума двох індексів є добуток  поліномів
              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
         // якщо поточний коефіцієнт g - це нульовий коефіцієнт,
         // виконуємо одне лише зрушення без множення, переміщаючи
         // символ з одного m-регістра в інший
                              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;
                  }
            }
      }
```

Нижче наведемо вихідний текст підпрограми декодера Ріда-Соломона.

```
      D2_RS()
      {
            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
      // виконуємо s[i] += recd[j] * ij
```

```
// тобто беремо черговий символ декодуємих даних,
// множимо його на порядковий номер даного символу,
// помножений на номер чергового оберту й складаємо
// отриманий результат із умістом s-регістра;
// по факті вичерпання всіх декодуємих символ ми
// повторюємо весь цикл обчислень знову – по одному
// разу для кожного символу парності
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
//---------------------------
// обчислюємо поліном локатора помилки через ітеративний алгоритм
// Берлекемпа. Випливаючи термінології Lin and Costello (див. "Error
// Control Coding: Fundamentals and Applications" Prentice Hall 1983
// ISBN 013283796) d[u] являє собою m ("мю"), що виражає
// розбіжність (discrepancy), де u = m + 1 і m є номер кроку
// з діапазону від -1 до 2t. У Блейхута та ж сама величина
// позначається D(x) ("дельта") і називається нев'язання.
// l[u] являє собою ступінь elp для даного кроку ітерації,
// u_l[u] являє собою різницю між номером кроку й ступенем elp
              // 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);
                                };
 // як тільки ми знайдемо q, такий що d[u] !=0
// і u_lu[q] є максимум
// запишемо ступінь нового elp полінома
if (l[u] > l[q] + u - q) l[u + 1] = l[u]; else l[u + 1] = l[q] + u - q;
// формуємо новий elp(x)
                        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];
 // формуємо (u + 1)'ю нев'язання
//---------------------------
                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)
                        { // записуємо корінь і індекс позиції помилки
                                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]];
                }
// обчислення значення помилок у позиціях loc[i]
//---------------------------
                for (i = 0; i < n; i++)
                {
                        err[i] = 0;
                        // переводимо recd[] у поліноміальну форму
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[i] повинен бути в поліноміальній формі
                              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
        }
                else                    // помилок не виявлено
for (i = 0; i < n; i++) if (recd[i] != -1) recd[i] =
alpha_to[recd[i]]; else recd[i] = 0;
}
```

Хоча я реалізовував програму сам, було використано підходи Scrum для саморозвитку та пришвидшенню розробки, розглянемо цей метод. Scrum – підхід управління проектами для гнучкої розробки програмного забезпечення. Скрам чітко робить акцент на якісному контролі процесу розробки.

Підхід вперше описали Гіротака Такеучі та Ікуджіро Нонака в статті The New New Product Development Game (Гарвардський Діловий Огляд, січ–лют 1986). Вони відзначили, що проекти, над якими працюють невеликі, крос–функціональні команди, зазвичай систематично продукують кращі результати, і пояснили це, як «підхід регбі».

У 1991 році ДеҐрейс та Шталь у книжці Злі проблеми, справедливі рішення послалися на цей підхід, як на Scrum (штовханина; сутичка навколо м'яча (у регбі)), спортивний термін, згаданий в статті Такеучі і Нонака. Кен Швабер на початку 1990–х використовував підхід який привів Scrum в його компанію.

Вперше метод Scrum було представлено на загальний огляд задокументованим, чітко сформульованим та описаним спільно Сазерлендом та Швабером на OOPSLA'96 в Остіні. Швабер та Сазерленд протягом наступних років працювали разом щоб обробити та описати весь їхній досвід та найкращі практичні зразки для індустрії в одне ціле, в ту методологію, що відома сьогодні як Scrum. Швабер об'єднав зусилля з Майком Бідлом в 2001, щоб детально описати метод в книжці Agile Software Development with SCRUM. Не зважаючи на те, що для Scrum нарікли долю управління проектами з розробки ПЗ, він може також використовуватися в роботі команд обслуговувань програмного забезпечення (software maintenance teams), або як підхід управління розробкою і супроводом програм: Scrum of Scrums.

Scrum – це кістяк процесу, який включає набір методів і попередньо визначених ролей. Головні дійові особи – ScrumMaster, той хто опікується процесами, веде їх і працює як керівник проекту, Власник Продукту, людина, що представляє інтереси кінцевих користувачів та інших зацікавлених в продукті сторін, та Команду, яка включає розробників.

Протягом кожного спринту, 15–30 денного періоду (тривалість визначається командою), працівники створюють функціональний ріст програмного забезпечення.

Набір можливостей, які імплементуються кожного спринту, приходять з етапу, що має назву product backlog (документація запитів на виконання робіт), який має найвищу пріоритетність за рівнем вимог до роботи, що повинна бути виконана.

Запити на виконання робіт (backlog items), що визначені протягом наради з планування спринту (sprint planning meeting), переміщуються в етап спринту.

Протягом цієї наради Власник Продукту інформує про завдання, які він хоче, аби були виконані. Тоді Команда визначає, скільки з бажаного вони можуть зробити, щоб завершити необхідні частини протягом наступного спринту. Протягом спринту команда виконує визначений фіксований список завдань (т.з. backlog items). Впродовж цього періоду ніхто не має права змінювати перелік запитів на виконання робіт, що слід розуміти, як заморожування вимог (requirements) протягом спринту.

Product backlog – це документ, який має список вимог до функціональності, які упорядковані згідно зі ступенем важливості. Product backlog представляє список того, що повинно бути реалізовано. Елементи цього списку називається «історіями» (user story) або елементами backlog–у (backlog items). Product backlog відкритий для редагування усім учасникам Scrum–процесу.

Обов'язкові поля:

1. ID – унікальний ідентифікатор, порядковий номер, який використовується для ідентифікації історій у разі їх перейменування.

2. Назва (Name) – стислий опис історії. Він повинен бути однозначним, щоб і розробники і product owner могли зрозуміти, про що йдеться і відрізнити одну історію від іншої.

3. Важливість (Importance) – ступінь важливості даної історії на погляд product owner 'а. Зазвичай являє собою натуральне число, іноді для цієї цілі використовуються числа Фібоначчі. Чим більше значення, тим більше пріоритет.

4. Попередня оцінка (initial estimate) – початкова оцінка об'єму робіт, необхідного для реалізації історії порівняно з іншими історіями. Вимірюється у story point'ах. Приблизно відповідає числу «ідеальних людино–днів».

5. Як продемонструвати (how to demo) – стисле пояснення того, як завершена задача буде продемонстрована у кінці спринта. Дане поле може являти собою код автоматизованого приймального тесту.

Додаткові поля. Іноді, також, використовуються додаткові поля у product backlog, в основному для того, щоб допомогти product owner'у визначитися з його пріоритетами.

Категорія (track). Наприклад, «панель управління» чи «оптимізація». За допомогою цього поля product owner може легко вибрати усі пункти категорії «оптимізація» і задати їм низький пріоритет.

Компоненти (components) – указує, які компоненти (наприклад, база даних, сервер, клієнт) будуть зачеплені при реалізації історії. Дане поле складається з групи checkbox'ів, які відмічаються, якщо відповідні компоненти потребують змін.

Ініціатор запиту (requestor). Product owner може захотіти зберігати інформацію про усіх замовників, зацікавлених у даній задачі. Це потрібно для того, щоб тримати їх у курсі діла про хід виконання робіт.

ID у системі обліку помилок (bug tracking ID) – якщо ви використовуєте окрему систему обліку помилок, тоді у описі історії корисно зберігати посилання на всі дефекти, які до неї відносяться.

Sprint backlog – містить функціональність, обрану Product Owner із Product Backlog. Всі функції розбиті по задачах, кожна з яких оцінюється командою. Кожен день команда оцінює об'єм роботи, який необхідно провести для завершення задачі.

Burndown chart – показує, скільки вже виконано і скільки ще залишається зробити.

Планування спринта (Sprint Planning Meeting)

Проходить на початку нової ітерації Спринта:

– Із Product Backlog обираються задачі, зобов'язання по виконанню яких за спринт приймає на себе команда;

– На основі обраних задач створюється Sprint Backlog. Кожна задача оцінюється у ідеальних людино–годинах;

– Рішення задачі не повинно займати більше 12 годин або одного дня. При необхідності задача розбивається на підзадачі;

– Обговорюється та визначається, яким чином буде реалізовано цей об'єм робіт;

– Тривалість наради обмежена зверху 4–8 годинами в залежності від тривалості ітерації, досвіду команди тощо;

– (перша частина наради) Беруть участь Product Owner + Команда: обирають задачі із Product Backlog;

– (друга частина наради) Бере участь лише команда: обговорюють технічні деталі реалізації, наповнюють Sprint Backlog.

Щоденна нарада (Daily Scrum Meeting)

Відбувається кожен день протягом спринта. Є «пульсом» ходу спринта. Нараді властиві наступні обмеження:

– починається точно вчасно;

– всі можуть спостерігати, але говорять тільки обрані;

– триває не більш ніж 15 хвилин;

– проводиться в одному і тому ж місці протягом одного спринта.

Протягом наради кожен член команди відповідає на 3 запитання:

– Що зроблено з моменту попередньої щоденної наради?

– Що буде зроблено з моменту поточної наради до наступної?

– Які проблеми заважають досягненню цілей спринта? (Над рішенням цих проблем працює ScrumMaster. Зазвичай це рішення проходить за рамками щоденної наради і у складі осіб, що безпосередньо займаються даною перешкодою.)

Демонстрація (Sprint Review Meeting):

– Проходить у кінці ітерації (спринта).

– Команда демонструє внесок функціональності до продукту всім зацікавленим особам.

– Залучається максимальна кількість глядачів.

– Усі члени команди беруть участь у демонстрації (одна людина на демонстрацію або кожен показує, що зробив за спринт).

– Обмежена 4–ма годинами в залежності від тривалості ітерації і змін у продукті.

Ретроспектива (Sprint Retrospective):

– Члени команди висловлюють свою думку про минулий спринт.

– Відповідають на два основних запитання: Що було зроблено добре у минулому спринті?; Що потрібно покращити в наступному?.

– Виконують покращення процесу розробки (вирішують питання та фіксують вдалі рішення).

– Обмежена 1–3ма годинами.

Jira – була використана комерційна система відслідковування помилок, призначена для організації взаємодії з користувачами, хоча в деяких випадках використовується і для управління проектами. Розроблено компанією Atlassian, є одним з двох її основних продуктів (поряд з вікі-системою Confluence). Має веб-інтерфейс.

Назва системи отримано шляхом усічення слова «Gojira» – Японського імені монстра Годзилла, що, в свою чергу, є відсиланням до назви конкуруючого продукту – Bugzilla; створювалася в якості заміни Bugzilla і багато в чому повторює її архітектуру. Система дозволяє працювати з декількома проектами. Для кожного з проектів створює і веде схеми безпеки і схеми оповіщення.

До версії 3.13.5 (включно) розрізнялися редакції Enterprise, Professional і Standard, після – Залишилася тільки редакція Enterprise (для великих організацій).

Система заснована на Java EE і працює на кількох популярних системах управління базами даних і операційних системах.

Основний елемент обліку в системі – завдання (ticket або issue). Завдання містить назву проекту, тему, тип, пріоритет, компоненти і зміст. Завдання може бути розширена додатковими полями (також і нові призначені для користувача поля можуть бути визначені), додатками (наприклад – Фотографіями, скриншотами) або коментарями. Завдання може редагуватися або просто змінювати статус, наприклад, з «відкритий» в «закритий». Які переходи між станами можливі, визначається через настроюється потік операцій. Будь-які зміни в задачі записуються в журнал.

Jira має велику кількість можливостей конфігурації: для кожної програми може бути визначений окремий тип завдання з власним workflow, набором статусів, одним або декількома видами уявлення (screens). Крім того, за допомогою так званих «схем» можна визначити для кожного індивідуального Jira-проекту власні права доступу, поведінку і видимість полів і багато іншого.

Завдяки універсальному підходу можна пристосувати Jira для багатьох непрофільних завдань, наприклад, керування вимогами, керування ризиками, аж до реалізації невеликої системи бронювання, автоматизації процесу рекрутингу.

Для інтеграції з зовнішніми системами підтримує інтерфейси SOAP, XML-RPC і REST. Поставляється із засобами інтеграції з такими системами управління версіями, як Subversion, CVS, Git, Clearcase, Team Foundation Server, Mercurial і Perforce.

Існують доповнення, що дозволяють вбудувати Jira в інтегровані середовища розробки, в тому числі Eclipse і IntelliJ IDEA. Перекладена багатьма мовами, включаючи українську, англійську, японську, німецьку, французьку, іспанську.

Для сторонніх розробників надаються кошти розробки розширень системи – плагінів. Розробники розширень можуть викладати плагіни для продажу на спеціальний розділ сайту Atlassian.

Є комерційним продуктом, який може бути ліцензований для роботи на локальному сервері або доступний в якості віддаленого додатки. Ціноутворення залежить від максимального числа користувачів, при цьому близько $50 за користувача для локального і $7 на місяць за користувача для віддаленого доступу є типовими цінами.

Для академічних і комерційних клієнтів доступний повний вихідний код під ліцензією розробника.

Для проектів з відкритим вихідним кодом Atlassian надає спеціальну безкоштовну ліцензію при дотриманні наступних правил:

– проект використовує ліцензії, схвалені Open Source Initiative;

– вихідний код проекту доступний для скачування;

– у проекту є публічно доступна веб-сайт;

– програмне забезпечення від Atlassian є на веб-сайті проекту.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм SHACAL-2, який шифрує дані 256-бітними блоками з використанням 512-бітного ключа. Допускається використання ключів менших розмірів (не менш 128 біт), які доповнюються бітовими нулями до 512 біт.

Шифруємий блок даних ділиться на 8 фрагментів по 32 біта (які позначені буквами *A*...*H*). Алгоритм виконує 64 раунду перетворень, у кожному з яких дані фрагменти обробляються в такий спосіб:

$$T = H_i + S_1(E_i) + Ch(E_i, F_i, G_i) + M_i + K_i,$$

$$H_{i+1} = G_i,$$

$$G_{i+1} = F_i,$$

$$F_{i+1} = E_i,$$

$$E_{i+1} = D_i + T,$$

$$D_{i+1} = C_i,$$

$$C_{i+1} = B_i,$$

$$B_{i+1} = A_i,$$

$$A_{i+1} = T + S_0(A_i) + Maj(A_i, B_i, C_i).$$

де T – тимчасова змінна.

Використовувані функції визначені в такий спосіб:

$$S_0(x) = (x >>> 2) \oplus (x >>> 13) \oplus (x >>> 22),$$

$$S_1(x) = (x >>> 6) \oplus (x >>> 11) \oplus (x >>> 25),$$

$$Ch(x, y, z) = (x \,\&\, y) \oplus (x' \,\&\, z),$$

$$Maj(x, y, z) = (x \,\&\, y) \oplus (x \,\&\, z) \oplus (y \,\&\, z),$$

де >>> – операція побітового циклічного зрушення вправо.

Константи, що модифікують, M_i ($i = 0…63$) наведено нижче (одна за одною від M_0 до M_{63}):

428A2F98	71374491	B5C0FBCF	E9B5DBA5
3956C25B	59F111F1	923F82A4	AB1C5ED5
D807AA98	12835B01	243185BE	550C7DC3
72BE5D74	80DEB1FE	9BDC06A7	C19BF174
E49B69C1	EFBE4786	0FC19DC6	240CA1CC
2DE92C6F	4A7484AA	5CB0A9DC	76F988DA
983E5152	A831C66D	B00327C8	BF597FC7
C6E00BF3	D5A79147	06CA6351	14292967
27B70A85	2E1B2138	4D2C6DFC	53380D13
650A7354	766A0ABB	81C2C92E	92722C85
A2BFE8A1	A81A664B	C24B8B70	C76C51A3
D192E819	D6990624	F40E3585	106AA070
19A4C116	1E376C08	2748774C	34B0BCB5
391C0CB3	4ED8AA4A	5B9CCA4F	682E6FF3
748F82EE	78A5636F	84C87814	8CC70208
90BEFFFA	A4506CEB	BEF9A3F7	C67178F2

Фрагменти розширеного ключа $K_0 \ldots K_{63}$ обчислюються в процесі процедури розширення ключа, що виконується в такий спосіб:

Етап 1. 512-бітний вихідний ключ шифрування ділиться на 16 фрагментів по 32 біта $K_0 \ldots K_{15}$...

Етап 2. Інші фрагменти розширеного ключа $K_{16} \ldots K_{63}$ обчислюються з перших 16 фрагментів у такий спосіб:

$$K_i = O_1(K_{i-2}) + K_{i-7} + O_0(K_{i-15}) + K_{i-16},$$

де функції O_0 і O_1 визначені так:

$$O_0(x) = (x >>> 7) \oplus (x >>> 18) \oplus (x >> 3),$$

$$O_1(x) = (x >>> 17) \oplus (x >>> 19) \oplus (x >> 10),$$

де >> – операція побітового зрушення (не циклічного) вправо.

Раунди розшифрування алгоритму виконуються у зворотній послідовності:

$$T = A_{i+1} + S_0'(B_{i+1}) + Maj'(B_{i+1}, C_{i+1}, D_{i+1}) + 2,$$

$$H_i = T + S_1'(F_{i+1}) + Ch'(F_{i+1}, G_{i+1}, H_{i+1}) + M_i' + K_i' + 4,$$

$$G_i = H_{i+1},$$

$$F_i = G_{i+1},$$

$$E_i = F_{i+1},$$

$$D_i = E_{i+1} + T' + 1,$$

$$C_i = D_{i+1},$$

$$B_i = C_{i+1},$$

$$A_i = B_{i+1}.$$

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у магістерської дипломної роботі програмне забезпечення системи кешування, флеш-технології та реплікації ЦОД. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Верхнього меню: Файл; Зберігання; Тестування; Параметри; Довідка.

– Розділу обрання групи файлів.

– Розділу виведення результату роботи системи.

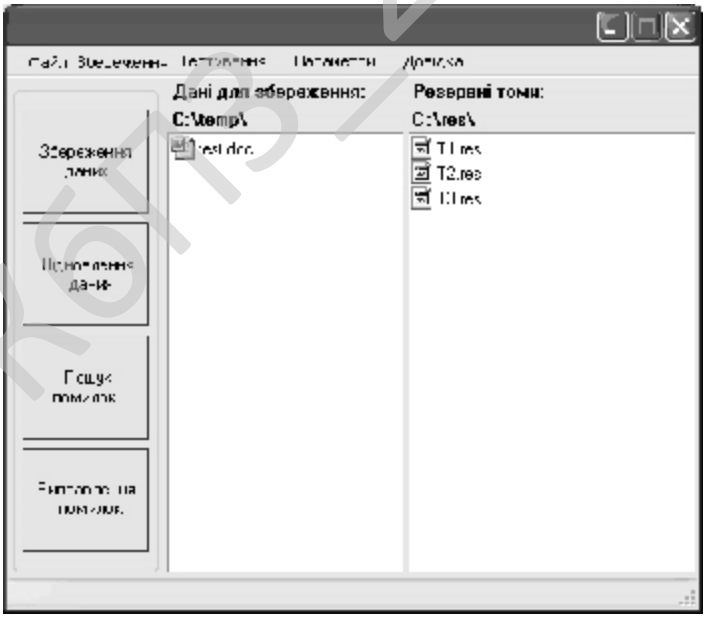

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним

середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

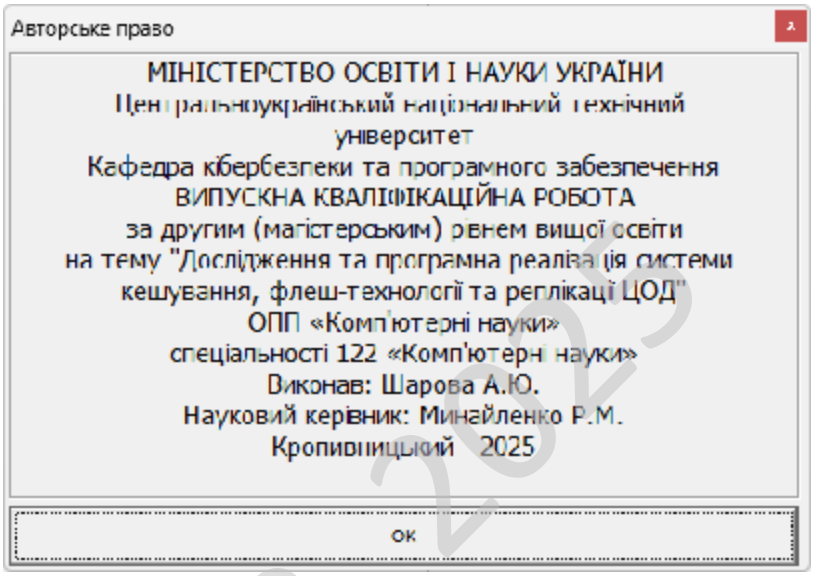

Рисунок 5.2 – Авторське право

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

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є унікальною, то усі процеси та процедури під час розгортання важко передбачити.

Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

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

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

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються в ІТ рішення за принципом найбільшої корисності для більшості учасників.

Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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

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

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

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

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

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

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

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

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

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

– У програмі можуть бути пропущені деякі маршрути.

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

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

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

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

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

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

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

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

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

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кешування, флеш-технології та реплікації ЦОД.

Метою розробки є дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД.

Об'єктом дослідження є процес кешування, флеш-технології та реплікації ЦОД.

Предметом дослідження є методи кешування, флеш-технології та реплікації ЦОД.

Методи дослідження базуються на методах теорії кодування, теорії великих даних,, методах математичної статистики, методах розробки програмного забезпечення.

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

– Удосконалено метод кешування, флеш-технології та реплікації ЦОД.

– Розроблено вітчизняний продукт кешування, флеш-технології та реплікації ЦОД, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати розробки системи кешування, флеш-технологій і реплікації в центрі обробки даних можуть зацікавити насамперед компанії, які працюють із великими обсягами даних і мають критично важливі бізнес-процеси, що залежать від стабільної роботи ІТ-інфраструктури. Це можуть бути банки, страхові компанії, інтернет-провайдери, державні органи або підприємства, що надають цифрові послуги клієнтам у режимі 24/7. Для них зменшення часу відгуку системи навіть на кілька мілісекунд може прямо вплинути на рівень прибутковості та довіру користувачів.

Також така розробка буде цікавою великим корпораціям, які мають розподілені офіси або філії в різних містах. Реплікація даних між серверами допомагає забезпечити безперервність бізнесу в разі збоїв або аварій. Компанії з e-commerce або онлайн-сервісами, які щодня обробляють сотні тисяч транзакцій, отримують від такого рішення не лише підвищення швидкодії, а й упевненість у тому, що інформація не буде втрачена навіть при критичних ситуаціях.

Науково-дослідним установам і закладам вищої освіти результати можуть бути цінними як практичний приклад застосування сучасних технологій зберігання даних. Для студентів ІТ-спеціальностей це демонструє, як взаємодіють між собою різні компоненти інфраструктури ЦОД – від кешування запитів до реплікації баз даних. Водночас провайдери хмарних сервісів також можуть інтегрувати подібну систему у свої рішення, щоб підвищити конкурентоспроможність і запропонувати клієнтам більшу надійність обслуговування.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для визначення привабливості впровадження такої системи доцільно провести експертну оцінку за участю фахівців у галузі IT-інфраструктури, кібербезпеки та фінансових аналітиків. Вони можуть оцінити ключові аспекти – технологічну новизну, ефективність використання ресурсів, рівень автоматизації процесів, надійність, масштабованість та очікувану окупність інвестицій. Наприклад, якщо за кожним параметром виставити оцінки від 1 до 10, середній інтегральний показник привабливості дозволить об'єктивно оцінити потенційну вигоду проєкту.

Припустимо, що експерти дали такі оцінки: технологічна інноваційність – 9 балів, надійність – 10, масштабованість – 8, окупність – 9, простота впровадження – 7. Середній бал 8,6 свідчить про високий рівень доцільності реалізації. Крім того, експерти можуть підкреслити, що перехід на флеш-накопичувачі та впровадження реплікації даних значно підвищують безперебійність і знижують ризики втрати критичних даних.

Експертна оцінка також допомагає врахувати ризики. Наприклад, можлива складність інтеграції з існуючими системами або потреба у додатковому навчанні персоналу. Проте зваживши ризики та вигоди, більшість експертів дійдуть висновку, що впровадження таких технологій у ЦОД має стратегічний характер і позитивно вплине на довгострокову стійкість підприємства.

7.3 Вибір методу оцінки вартості ПЗ

Найдоцільніше використовувати метод повної вартості володіння (TCO – Total Cost of Ownership) у поєднанні з аналізом дисконтованих грошових потоків (DCF). Перший метод дозволяє врахувати всі прямі та непрямі витрати, включно з закупівлею обладнання, ліцензій, навчанням персоналу, енергоспоживанням і

технічною підтримкою. Другий метод допомагає оцінити, коли саме інвестиції почнуть приносити реальний дохід і яку прибутковість вони забезпечать у довгостроковій перспективі. Флеш-технології та кешування мають вищу початкову вартість у порівнянні з традиційними HDD-рішеннями, проте вони забезпечують суттєве зниження витрат на енергоспоживання, обслуговування та простої. Тому саме DCF-аналіз допоможе точно розрахувати термін окупності, який, зазвичай, не перевищує 8–12 місяців. Також доцільно враховувати нефінансові вигоди, які складно безпосередньо виміряти грошима, наприклад – підвищення швидкодії бізнес-процесів, зниження репутаційних ризиків або підвищення лояльності клієнтів. Ці фактори можуть стати вирішальними у прийнятті інвестиційного рішення, особливо для компаній, орієнтованих на клієнтський сервіс.

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

Аналіз охоплює вихідні показники, прогнозовані результати після впровадження, фінансові обчислення та нефінансові ефекти, що підкріплюють доцільність інвестицій. Вхідні дані зафіксовано в таблиці 7.1. Розрахунок економічного ефекту демонструє наступне: економія від скорочення простоїв – 6 000 000грн, економія електроенергії – 400 000грн, економія на адмініструванні – 600 000грн, додатковий прибуток від збільшення пропускної здатності (35%) орієнтовно – 3 000 000грн, загальний річний ефект – 10 000 000грн, витрати на обслуговування – -700 000грн, чистий річний ефект (прибуток) – 9 300 000 грн, термін окупності – 0,65 року (~8 місяців), ROI = 155%. Нефінансові результати: скорочення ризиків втрати даних завдяки реплікації у режимі реального часу – підвищення рівня надійності сервісів, підвищення задоволеності користувачів – швидше завантаження додатків і стабільний доступ до корпоративних ресурсів, зменшення часу резервного

копіювання – SSD-технології та кеш-схеми дозволили зменшити тривалість резервування з 4 годин до 40 хвилин, покращення корпоративного іміджу – компанія демонструє цифрову зрілість та відповідальне ставлення до управління IT-ресурсами.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	**До впровадження**	**Після впровадження**
Кількість запитів до дискових масивів на добу	1 000 000	300 000
Середній час відгуку системи	20 мс	4 мс
Кількість критичних простоїв на рік	6	1
Середня тривалість простою	3 години	0,5 години
Середня вартість простою (втрачені прибутки, не оброблені транзакції тощо)	400 000 грн/год	400 000 грн/год
Витрати на електроенергію для зберігання даних	1 200 000 грн/рік	800 000 грн/рік
Витрати на адміністрування ЦОД	2 000 000 грн/рік	1 400 000 грн/рік
Початкові інвестиції в модернізацію ЦОД (SSD-модулі, кеш-системи, реплікатори)	—	6 000 000 грн
Щорічні витрати на обслуговування нових систем	—	700 000 грн

Таке рішення доцільне для підприємств, які працюють з великими обсягами даних – банків, телекомунікаційних операторів, виробничих корпорацій і хмарних провайдерів, адже воно знижує операційні ризики й забезпечує швидке повернення інвестицій.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Першим етапом просування проєкту має стати пілотне впровадження на невеликій частині інфраструктури, щоб продемонструвати реальний ефект – зменшення часу відгуку, підвищення швидкості обробки запитів і стабільність при реплікації даних. Це допоможе отримати практичні докази ефективності, які можна використати в подальших маркетингових матеріалах.

Наступним кроком буде участь у спеціалізованих ІТ-конференціях і виставках, де розробники можуть презентувати систему потенційним клієнтам – системним інтеграторам, дата-центрам і корпоративним клієнтам. Розміщення аналітичних статей, кейсів і демонстраційних відео також допоможе підвищити впізнаваність продукту.

Після цього можна перейти до етапу масштабного впровадження через партнерські програми з постачальниками ІТ-обладнання або хмарними провайдерами. Їм можна запропонувати інтеграцію технології кешування та реплікації у власні рішення як частину їхнього сервісу. Завершальним етапом має бути впровадження системи технічної підтримки та оновлень, щоб гарантувати довготривале функціонування та підвищення довіри клієнтів.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Одним із найефективніших шляхів оптимізації збуту є створення партнерських програм із виробниками серверного обладнання та постачальниками корпоративних ІТ-рішень. Це дозволить інтегрувати систему у готові рішення “під ключ”, що підвищить її доступність для кінцевих клієнтів. Крім того, можна розробити модульну архітектуру, яка дає змогу адаптувати систему під різні масштаби ЦОД – від невеликих приватних до великих корпоративних.

Ще одним напрямом може стати використання моделі ліцензування за підпискою, що забезпечить стабільний грошовий потік і дозволить клієнтам поступово впроваджувати систему без значних одноразових витрат. Для залучення нових клієнтів варто запропонувати безкоштовний тестовий період або демонстраційні впровадження, які дадуть змогу переконатися у перевагах системи.

Також ефективною може бути співпраця з консалтинговими компаніями, що спеціалізуються на оптимізації IT-інфраструктур. Вони можуть виступати як незалежні експерти, які рекомендують дане рішення своїм клієнтам. Таким чином, збутовий процес стане більш ефективним і менш затратним.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовим фактором успіху є стабільність та надійність системи, адже будь-які збої в ЦОД можуть мати катастрофічні наслідки для бізнесу. Тому особлива увага повинна приділятися тестуванню та перевірці на різних навантаженнях. Не менш важливою є масштабованість рішення – воно повинно легко адаптуватися до зростання обсягів даних і нових технологічних вимог.

Другим визначальним фактором є зручність впровадження. Якщо система легко інтегрується у наявну IT-інфраструктуру без потреби в значних змінах, це зменшує бар’єр входу для потенційних клієнтів. Також важливу роль відіграє ефективна технічна підтримка, яка гарантує своєчасне оновлення та усунення можливих проблем.

Крім того, довіра клієнтів до постачальника рішення є не менш важливою. Компанія повинна забезпечити прозорість роботи системи, безпечність реплікації даних і відповідність сучасним стандартам кіберзахисту. Якщо всі ці аспекти поєднуються з реальним економічним ефектом, проєкт має всі шанси стати успішним і масштабуватися на міжнародному рівні.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Охорона праці – це: система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини в процесі трудової діяльності;

Охорона праці є складовою частиною безпеки життєдіяльності [3,4].

Законом України “Про охорону праці” [3] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

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

Ці шкідливі фактори можуть привести до професійних захворювань.

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

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

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

Загальний нагляд за додержанням норм охорони праці покладено на прокуратуру, спеціальний покладено на професійні спілки. За безпекою контроль праці здійснюють державні й відомчі спеціалізовані інспекції.

У Законодавстві про працю міститься вимоги і норми з виробничої санітарії, техніки безпеки та норми, що регулюють робочий час, час відпочинку, звільнення та переведення на іншу роботу, а також норми праці щодо жінок, молоді, гігієнічні норми і правила, тощо.

8.2 Аналіз умов праці

Фірма дотримується всіх правил з охорони праці і слідкує за їх дотриманням працівниками.

При виконанні робіт на комп'ютерах працівникам необхідно дотримуватись вимог загальної інструкції з охорони праці.

До роботи на комп'ютерах допускаються особи, які пройшли: медичний огляд, навчання за професією, вступний інструктаж з охорони праці та первинний

інструктаж з охорони праці на робочому місці. В подальшому вони проходять повторні інструктажі з охорони праці на робочому місці один раз на півріччя, періодичні медичні огляди один раз на два роки.

Основним обладнанням робочого місця є монітор, системний блок, миша та клавіатура.

Робочі місця розташовані на відстані не менше 1,5 м від стіни з вікнами, від інших стін – на відстані 1м та між собою на відстані не менше 1,5 м. Відносно вікон робоче місце доцільно розташовувати таким чином, щоб природне світло падало на нього збоку, переважно зліва.

Монітор розташовується на робочому місці так, щоб поверхня екрана знаходилася в центрі поля зору на відстані 400-700 мм від очей користувача. Елементи робочого місця розміщуються так, щоб витримувалася однакова відстань очей від екрана, клавіатури, тексту.

Джерела освітлення розташовані з обох боків екрану паралельно напрямку погляду. Для уникнення світлових відблисків екрану та клавіатури в напрямку очей користувача від світильників загального освітлення або сонячних променів також використовують антивідблискові сітки, спеціальні фільтри для екранів, захисні козирки, на вікнах – жалюзі.

Використовуються скляні поляризаційні фільтри, які забезпечують найкращу якість зображення. Вони усувають практично всі відблиски, роблять зображення чітким і контрастним.

Зручна робоча поза при роботі з комп'ютером забезпечується регулюванням висоти робочого столу, крісла та підставки для ніг. Працівники мають пам’ятки, що раціональною робочою позою вважається положення, при якому ступні людини розташовані горизонтально на підлозі або підставці для ніг, стегна зорієнтовані у горизонтальній площині, верхні частини рук – вертикальні. Кут ліктьового суглобу коливається в межах 70 – 90°, зап'ястя зігнуті під кутом не більше ніж 20°, нахил голови 15-20°.

Для нейтралізації зарядів статичної електрики в приміщенні, де виконується робота на комп'ютерах, в тому числі на лазерних та світлодіодних принтерах, збільшується вологість повітря за допомогою кімнатних зволожувачів. Не рекомендується носити одяг з синтетичних матеріалів.

Для попередження травм усе електричне обладнання заземлене. Приступаючи до роботи, працівникам необхідно перевірити справність обладнання. В разі виявлення порушень їм треба негайно повідомити про це свого керівника для вжиття заходів щодо усунення несправності. Проводити самостійно ремонт електроустаткування забороняється.

8.3 Розробка заходів з охорони праці

Працівники повинні дотримуватися статті 18 Закону України ”Про охорону праці” згідно цій статті працівники зобов'язані:

– знати і виконувати вимоги нормативних актів про охорону праці, правила поводження з устаткуванням та іншими засобами виробництва, користуватися засобами колективного та індивідуального захисту;

– співробітничати з власником у справі організації безпечних і нешкідливих умов праці, особисто вживати посильних заходів щодо усунення будь-якої виробничої ситуації, яка створює загрозу його життю чи здоров'ю, або людей, які його оточують, повідомляти про небезпеку свого безпосереднього керівника або іншу посадову особу;

– дотримуватись зобов'язань щодо охорони праці, передбачених колективним договором та правилами внутрішнього трудового розпорядку підприємства.

Також повинні виконуватися вимоги безпеки перед початком роботи, працівникам потрібно:

– увімкнути систему кондиціювання в приміщенні;

– перевірити надійність встановлення апаратури на робочому столі.

Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) і трохи зверху вниз, при цьому екран має бути трохи нахиленим, нижній його край ближче до оператора;

– перевірити загальний стан апаратури, перевірити справність електропроводки, з'єднувальних шнурів, штепсельних вилок, розеток, заземлення захисного екрану;

– відрегулювати освітленість робочого місця;

– відрегулювати та зафіксувати висоту крісла, зручний для користувача нахил його спинки;

– приєднати до системного блоку необхідну апаратуру. Усі кабелі, що з’єднують системний блок з іншими пристроями, слід вставляти та виймати при вимкненому комп'ютері;

– ввімкнути апаратуру комп'ютера вимикачами на корпусах в послідовності: монітор, системний блок, принтер (якщо передбачається друкування);

– відрегулювати яскравість монітора, мінімальний розмір світної точки, фокусування, контрастність.

Працівники повинні дотримуватися рекомендацій:

– яскравість монітору – не менше 100 Kg/м^2;

– відношення яскравості монітора до яскравості оточуючих його поверхонь в робочій зоні – не більше 3:1;

– мінімальний розмір точки свічення не більше 0,4 мм для монохромного монітора і не менше 0,6 мм для кольорового, контрастність зображення знаку – не менше 0,8.

При виявленні будь-яких неполадок роботу не розпочинати, повідомити про це керівника.

Працівникам потрібно дотримуватися вимог безпеки під час виконання роботи:

– необхідно стійко розташовувати клавіатуру на робочому столі, не опускати її хитання. Під час роботи на клавіатурі сидіти прямо, не напружуватися;

– для уникнення несприятливого впливу на користувача пристроїв типу ”миша” належить забезпечувати вільну велику поверхню столу для переміщення ”миші” і зручного упору ліктьового суглоба;

– не дозволяються сторонні розмови, подразнюючі шуми;

– періодично при вимкненому комп'ютері прибирати ледь змоченою мильним розчином бавовняною ганчіркою порох з поверхонь апаратури. Екран ВДТ та захисний екран протирають ганчіркою, змоченою у спирті. Не дозволяється використовувати рідинні або аерозольні засоби чищення поверхонь комп'ютера.

Працівники не повинні порушувати правил з охорони праці, їм забороняється:

– самостійно ремонтувати апаратуру. Ремонт апаратури здійснюється спеціалістами з технічного обслуговування комп'ютера, один раз на півроку повинні відкривати процесор і вилучати пилососом пил і бруд, що накопичилися;

– класти будь-які предмети на апаратуру комп'ютера;

– закривати будь-чим вентиляційні отвори апаратури, що може призвести до її перегрівання і виходу з ладу.

Для зняття статичної електрики рекомендується час від часу доторкатися до металевих поверхонь.

Розташувати принтер необхідно поруч з системним блоком таким чином, щоб з’єднувальний шнур не був натягнутий. Забороняється ставити принтери на системний блок.

Для досягнення найбільш чистих, з високою роздільністю зображень і щоб не зіпсувати апарат, має використовуватися папір, вказаний в інструкції до принтера. При зминанні паперу потрібно відкрити кришку і обережно витягнути лоток з папером.

Працівникам потрібно дотримуватися вимог безпеки після закінчення роботи:

– закінчити та зберегти у пам'ять комп'ютера файл, що знаходиться в роботі;

– вимкнути принтер та інші периферійні пристрої. Штепсельні вилки витягнути з розеток. Накрити клавіатуру кришкою для запобігання попаданню в неї пилу;

– прибрати робоче місце;

– ретельно вимити руки теплою водою з милом;

– вимкнути кондиціонер, освітлення і загальне електроживлення;

– пройти в спеціально обладнаному приміщенні сеанс психофізиологічного розвантаження і зняття втоми з виконанням спеціальних вправ аутогенного тренування.

8.4 Пожежна безпека

Пожежі в приміщеннях з оргтехнікою становлять особливу небезпеку, бо поєднані з великими матеріальними збитками. Пожежа може виникнути при взаємодії горючих речовин і джерел запалювання. Горючими речовинами є будівельні та опоряджувальні матеріали, пластмасові корпуси техніки, шнури тощо. Джерелами запалювання можуть бути електронні схеми комп'ютерів, принтерів, пристроїв електроживлення, де внаслідок різних порушень виникає перегрівання елементів, утворюються електричні іскри та дуги, здатні спричинити займання горючих матеріалів.

З метою виявлення початкової стадії займання необхідно використовувати пристрої систем автоматичного пожежогасіння там, де цього вимагають правила пожежної безпеки.

При обслуговуванні, ремонтних та профілактичних роботах використовуються різні легкозаймисті рідини, прокладаються тимчасові

електропровідники, здійснюється паяння. Виникає додаткова пожежна небезпека, яка потребує відповідних заходів пожежного захисту. До засобів гасіння пожежі, призначених для локалізації невеликих займань, належать вогнегасники, сухий пісок, азбестові ковдри. Приміщення, в який встановлено комп'ютери і де немає необхідності влаштування систем автоматичного пожежогасіння, необхідно оснащувати переносними вуглекислотними з розрахунку 2 шт. на кожні 20 $м^2$ в приміщеннях. Звуковбирне облицювання стін, стель приміщень треба виконувати з негорючих та важко горючих матеріалів.

Електроустановки (можливість їх застосування, монтаж, накладка експлуатація) повинні відповідати вимогам чинних правил улаштування електроустановок, правил технічної експлуатації, електроустановок та інших нормативних документів.

Ймовірність виникнення пожежі від електротехнічного та іншого одиничного виробу не повинна перевищувати 10^{-6} на рік. При короткому замиканні в місцях з'єднання проводів опір практично дорівнює нулю, звідси величина струму досягає дуже великих значень.

Персональні комп'ютери після закінчення роботи повинні відключатися від мережі. Не рідше 1 разу на квартал, необхідно очищати від пилу агрегати та вузли, кабельні канали та простір між підлогами. Не дозволяється розміщувати комп'ютерні зали у підвалах; проводити ремонт вузлів (блоків) ЕОМ безпосередньо у залах, де знаходяться ПК (персональні комп'ютери), залишати без нагляду ввімкнену в мережу електронну апаратуру, яка використовується для контролю ЕОМ.

Електричний струм силою 0,1 А є небезпечним для людини. Для попередження травм усе електричне обладнання повинне бути заземлене. Приступаючи до роботи необхідно перевірити справність обладнання, ізоляцію проводів і надійність заземлення. Доторкання до оголених струмоведучих і незахищених частин в електроустаткуванні забороняється. В разі виявлення порушень ізоляції електропроводів, відкритих струмоведучих частин

електроустаткування або порушення заземлення треба негайно повідомити про це свого начальника для вжиття заходів щодо усунення несправності. Проводити самому ремонт електроустаткування забороняється.

8.5 Розрахункова частина

Проведемо розрахунок штучного освітлення за методом коефіцієнту використання світлового потоку для приміщення ширина якого складає 2,6 м, довжина – 2,6 м, висота – 3 м.

У зазначеному приміщенні працює 1 особа.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F = E \cdot S \cdot K \cdot Z / n,$$

де:

F – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; E = 300 Лк;

S – площа освітлюваного приміщення (у нашому випадку S=2,6 х 2,6 = 6,76 м^2);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку K = 1,5);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку Z = 1,1);

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами

відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50%.

Обчислимо індекс приміщення за формулою:

$$i=S/(h\cdot(A+B)),$$

де

S – площа приміщення, S = 6,76 м2;

h – розрахункова висота підвісу, h = 3 м (співпадає з висотою стелі, оскільки лампи освітлення закріплюються на стелі);

A – ширина приміщення, A = 2,6 м;

B – довжина приміщення, B = 2,6 м.

Підставимо всі значення у формулу та визначимо індекс приміщення: i=0,43.

Знаючи індекс приміщення, знаходимо n = 0,23 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників з відповідним типом ламп). Підставимо всі значення у формулу, визначимо світловий потік: F=14548 Лм.

Будемо використовувати світлодіодні панелі Lebron L-LPU-Prismatic, світловий потік яких $F_л$ = 3000 Лм.

Число ламп визначається по формулі:

$$N=F/F_л$$

де

F – світловий потік,

$F_л$ – світловий потік однієї лампи.

Підставимо всі значення у формулу та визначимо кількість ламп:

$$N= 14548/ 3000=4,8 \text{ шт.}$$

Приймаємо необхідну кількість ламп 5 шт.

Висновки до розділу

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

З цих міркувань було здійснено аналіз умов праці, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок штучного освітлення, як одного з ключових факторів впливу на працездатність та здоров’я програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи кешування, флеш-технології та реплікації ЦОД.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів кешування, флеш-технології та реплікації ЦОД.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем кешування, флеш-технології та реплікації ЦОД.

– Досліджена система кешування, флеш-технології та реплікації ЦОД.

– На основі отриманих результатів досліджень створена програмна реалізація системи кешування, флеш-технології та реплікації ЦОД.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання кешування, флеш-технології та реплікації ЦОД.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм SHACAL-2.

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

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шарова А.Ю. Дослідження та програмна реалізація системи кешування, флеш-технології та реплікації ЦОД // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

24. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and

cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

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

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

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

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

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

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

31. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019,

Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

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

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

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

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

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

37. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

38. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special

Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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