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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи центру обробки даних SSDC”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-23Мз
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Підгорна Ю.В.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Коваленко А.С.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Центр *Заочної та дистанційної освіти*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *магістр*
Галузь знань _12 _____*"Інформаційні технології"*_____
Спеціальність 123 *"Комп'ютерна інженерія"*
Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерна інженерія"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2024 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Підгорній Юлії Валентинівні
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи центру обробки даних SSDC*

2. Керівник роботи *Коваленко Анна Степанівна, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 21-13 від 07.08.2024 року

3. Строк подання студентом роботи до захисту *2.12.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи центру обробки даних SSDC*

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

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Маркетингове та економічне обґрунтування ІТ-проєкту.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Доренська А.О.	05.10.2024	14.11.2024
Охорона праці	Марченко К.М., к.т.н., доцент	06.10.2024	16.11.2024

7. Дата видачі завдання « 6 » вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2024 р.	
3.	Розробка моделі компонента	20.10.2024 р.	
4.	Розробка структур даних	25.10.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2024 р.	
6.	Програмування алгоритмів	10.11.2024 р.	
7.	Розрахунок економічної ефективності	13.11.2024 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2024 р.	
9.	Оформлення ПЗ	17.11.2024 р.	
10.	Попередній захист роботи	2.12.2024 р.	

Дата видачі завдання
« 6 » вересня 2024 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2024 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Підгорна Ю.В. Дослідження та програмна реалізація системи центру обробки даних SSDC. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи центру обробки даних SSDC.

Метою розробки є дослідження та програмна реалізація системи центру обробки даних SSDC.

Об'єктом дослідження є процес центру обробки даних SSDC.

Предметом дослідження є методи центру обробки даних SSDC.

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

Результат роботи – програмна реалізація системи центру обробки даних SSDC.

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

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

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

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, центр обробки даних, SSDC

ABSTRACT

Podgorna Yu.V. Research and software implementation of the SSDC data center system. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the SSDC data center system.

The purpose of the development is the research and software implementation of the SSDC data center system.

The object of study is the SSDC data center process.

The subject of the study is SSDC data center methods.

Research methods are based on computer network theory methods, mathematical statistics methods, and software development methods.

The result of the work is the software implementation of the SSDC data center system.

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

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

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

The program was developed in the Python environment.

Keywords: computer engineering, data center, SSDC

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

БНМ	–	балансування навантаження мережі, Network Load Balancing
ЕЦП	–	електронний цифровий підпис
ІВК	–	інфраструктура відкритих ключів
ІТ	–	інформаційні технології
ЛОМ	–	локальна обчислювальна мережа
ПЗ	–	програмне забезпечення
СКЗІ	–	система комплексного захисту інформації
УК	–	управляючі компоненти
УЦ	–	удостовірюючий центр
ЦЗД	–	центр зберігання даних
ЦОД	–	центр обробки даних
IGMP	–	Internet Group Management Protocol
NLB	–	Network Load Balancing, балансування мережного навантаження
PKI	–	Public Key Infrastructure
VPN	–	віртуальна приватна мережа

ВСТУП

Актуальність теми. Провідні виробники систем зберігання даних розглядають програмно обумовлені СЗД як стратегічний напрямок розвитку галузі й один з найважливіших компонентів програмно обумовлених ЦОД. Але по суті це черговий етап еволюції давно відомої технології – віртуалізації систем зберігання.

Ідея SDDC була висловлена VMware в 2012 році. На думку багатьох, у цьому – скоріше, маркетинговому – терміну немає якогось глибокого змісту, хоча в цілому ідея зрозуміла. Але як би до даного підходу не ставилися, проблема лише в складності його повноцінної практичної реалізації на нинішньому етапі.

Відповідно до визначення IDC, SDDC – набір слабко зв'язаних програмних компонентів, що забезпечують віртуалізацію й об'єднання ресурсів ЦОД (обчислювальних, мережних й призначених для зберігання даних). Завдання SDDC – зв'язати різнорідні ресурси IT для надання інтегрованих сервісів. В VMware програмно обумовлений ЦОД розуміють як інфраструктуру з високим ступенем віртуалізації й автоматизованим програмним керуванням, надавану по моделі «IT як сервіс». Інакше кажучи, архітектура SDDC припускає поширення концепції віртуалізації на всі ресурси ЦОД, а також абстрагування ресурсів і автоматизацію керування.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи центру обробки даних SSDC.

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

– Огляд існуючих систем центру обробки даних SSDC.

– Дослідження системи центру обробки даних SSDC.

– Програмна реалізація системи центру обробки даних SSDC.

Об'єктом дослідження є процес центру обробки даних SSDC.

Предметом дослідження є методи центру обробки даних SSDC.

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

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

– Удосконалено метод центру обробки даних SSDC.

– Розроблено вітчизняний продукт центру обробки даних SSDC, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі центру обробки даних SSDC.

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» (2024 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 центру обробки даних SSDC, 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

Термін «програмно-визначаємий ЦОД» (у термінології VMware – Software-Defined Data Center, SDDC) уживається професіоналами вже не перший рік. Але найчастіше він служить «ярликом», що наклеюється на різний зміст – залежно від контексту й особливостей портфоліо постачальників рішень.

Фахівці розуміють програмно-визначаємий ЦОД як підхід, що передбачає комплексну оркестрацію компонентів дата-центра – віртуалізованих (здебільшого) і фізичних: віртуальних машин, СЗД, мережі, мережних сервісів, інженерної інфраструктури та ін. У готовому виді (у форматі «коробкового» рішення) інструмент, що дозволяє забезпечити оркестрацію для всіх значимих компонентів ЦОД, на ринку відсутній, хоча є його окремі компоненти.

Підприємства, у середині минулого року які планували будівництво власних ЦОД, у силу економічної ситуації виявилися в умовах лімітованих бюджетів. Від 30 до 40% підприємств будують нові ЦОД, намагаючись скоротити витрати за рахунок більше інтенсивного використання ресурсу інженерної інфраструктури існуючих ЦОД; відмови від застосування дорогих екологічних/енергозберігаючих рішень; переходу на встаткування азіатських і/або вітчизняних виробників; комбінуючи перераховані вище підходи. Але більшість підприємств не можуть собі дозволити навіть таких, скорочених, витрат. Тому фокус уваги користувачів ЦОД зміщається від будівництва власних площадок до оренди готових.

Тенденція переходу на сервісну модель в IT і прагнення замовників мінімізувати витрати служать драйверами руху ринку убік програмно-визначаємих ЦОД. Цьому сприяють також масовий перехід на платформу x86 і наявність в існуючих ЦОД надлишкового обчислювального ресурсу, якому

можна використовувати для створення віртуальних СЗД і комутаторів замість фізичних.

Є комерційні й Open Source-продукти для керування окремими віртуалізованими компонентами ЦОД. Для створення рішення, що підходить під визначення повноцінного SDDC, їх потрібно інтегрувати між собою й з рівнем керування/оркестрації.

Оркестрація SDDC припускає забезпечення моніторингу, обліку й тарифікації, автоматизованого зміни конфігурації (provisioning), планування ресурсів, погодженої реакції на події суміжних систем і т.д. В умовах різноманітного корпоративного середовища реалізувати повною мірою концепцію SDDC на базі наявних продуктів складно, довго й дорого, оскільки інтеграція компонентів і кастомизація рішень вимагають високої кваліфікації програмістів.

Бажаючої рухатися убік SDDC рекомендується спершу визначити ключові показники дата-центра, забезпечити їхній незалежний моніторинг і працювати над поліпшенням найбільш значимих показників за допомогою окремих впроваджуваних рішень, вбудовуючи їх у загальну концепцію SDDC.

1.2 Область застосування

SDDC як закінчене рішення вже працює в провайдерів послуг (таких як Amazon або Google), що мають однорідну інфраструктуру й власний штат розроблювачів. Для корпоративного сектора закінчених SDDC-рішень, що покривають весь спектр прикладних систем і гетерогенних інфраструктур, поки ні, і перспективи їхньої появи протягом найближчих п'яти років сумнівні. Причини тому – несильна зацікавленість провідних виробників апаратних рішень у розвитку тематики SD, організаційні складності, що виникають в IT-служб компаній-замовників при експлуатації таких гіперконвергентних рішень, а також дорожнеча ключових компонентів SDDC – систем автоматизації й керування – як

у реалізації, так і в обслуговуванні. Всупереч розхожій думці, ПЗ OpenStack є не готовим продуктом, а набором програмних модулів, які потрібно інтегрувати вручну, а потім і експлуатувати, забезпечуючи досить високий рівень доступності. Це трудомісткий процес, що вимагає участі кваліфікованих програмістів (власної або притягнутої команди). Крім того, архітектура сучасних корпоративних додатків не «заточена» під хмарну архітектуру, що пропонує OpenStack. Тому OpenStack поки не дуже підходить для корпоративного застосування.

Всупереч запевненням виробників, чисто програмні реалізації віртуалізації СЗД поки ще не забезпечують функціонала, надійності й продуктивності рівня масивів Hi-End або серйозних Midrange. З апаратними реалізаціями справи йдуть набагато краще, але й вони з ряду причин показані далеко не у всіх випадках. Не завжди вони знімають проблеми сумісності встаткування, дорожнечі й складності експлуатації гетерогенних СЗД. Проте застосування віртуалізації СЗД добре підходить для певних завдань, таких як розподілений («розтягнутий») на різні площадки кластер віртуальних машин, розширення ємності Hi-End-масивів більше бюджетними дисковими полками або міграція даних зі старих масивів на нові.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи центру обробки даних SSDC, 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

Крос-хмарна архітектура й VMware Cloud Foundation, єдина платформа для керування SDDC

На VMworld 2016 був анонсований розвиток власної стратегії компанії в сфері гібридної хмари на базі інноваційній крос-хмарної архітектури VMwareCross-Cloud Architecture; така архітектура дає можливість створення уніфікованого середовища, у якій буде здійснюватися весь цикл керування додатками, у фізичному плані які знаходяться в різних хмарних середовищах і на різних девайсах. Для втілення цієї стратегії в життя компанія запропонувало ряд програмних компонентів, що підвищують ефективність використання хмарних моделей:

– VMware Cloud Foundation (платформу для керування SDDC – програмно-визначаємими ЦОД);

– Technology Preview of Cross-Cloud Services (прев'ю сервісів керування приватними й публічними хмарами від різних постачальників);

– VMware vCloud Availability (нову лінійку засобів резервного копіювання);

– нову версію VMware vCloud Air Hybrid Cloud Manager (уможливлює моментальну міграцію додатків з vSphere в vCloud Air).

Єдина платформа для програмно-визначаємих ЦОД

VMware Cloud Foundation забезпечує гіперконвергентну структуру наступного покоління для побудови приватних хмар, які вперше сполучать у собі

масштабоване гіперконвергентне ПЗ VMware (для VMware vSphere і VMware Virtual SAN) із провідною світовою платформою віртуалізації, VMware NSX.

VMware SDDC Manager, основний компонент VMware Cloud Foundation, допомагає клієнтам і постачальникам послуг автоматизувати розгортання й керування хмарним програмним забезпеченням VMware. SDDC Manager допомагає створювати й підтримувати весь стек хмарного програмного забезпечення VMware, звільняючи хмарних адміністраторів від складного й трудомісткого завдання установки, настроювання, керування й відновлення хмарної інфраструктури, що дозволяє побудувати повноцінну хмару в лічені годинники. У результаті час розгортання скорочується в 6-8 разів, а також досягається економія в 30-40% від сукупної вартості володіння (TCO).

Уперше VMware Cloud Foundation пропонує нову опцію « as-a-service», що забезпечує повну потужність ЦОД у гібридному хмарному середовищі. Для приватної хмари замовники можуть використовувати VxRack комплексні рішення під ключ від EMC, або комбінувати ПЗ Cloud Foundation з VMware Virtual SAN Ready Nodes від Dell, Hewlett Packard Enterprise і QCT.

VMware Cloud Foundation може запускати будь-який традиційний або хмарний додаток. Незалежно від того, чи перебувають вони у віртуальних машинах або контейнерах, VMware Cloud Foundation є послідовною інфраструктурною платформою, що забезпечує високу продуктивність, відказостійкість, безпеку й керованість – всі переваги vSphere, Virtual SAN і VMware NSX. VMware Cloud Foundation інтегрується з існуючими рішеннями VMware для підтримки хмарної гнучкості й волі вибору, і уможливлює мобільність бізнесу; у тому числі, підтримуються наступні продукти:

– **VMware vrealize Suite** забезпечує комплексну платформу хмарного керування корпоративного рівня, що може прискорити надання IT-послуг, поліпшити їх і забезпечити вибір в області контрольованих, гетерогенних мультихмарних середовищ (vSphere і не-vSphere).

– **VMware vSphere Integrated Containers** дозволяють розроблювачам впроваджувати інновації швидше з безпечним багатокористувальницьким доступом до контейнерів, у той час як IT могтимуть використовувати існуючі інструменти, знання й процеси розгортання й керування контейнерних сервісів.

– **VMware Integrated OpenStack** дає клієнтам найкоротший шлях до розгортання хмари OpenStack на вершині програмно-визначаємой інфраструктури VMware.

– **VMware Horizon** дозволяє клієнтам швидко доставити віртуальні робочі столи й додатки через єдину платформу, створюючи безпечне цифрове робоче місце.

IT-департаменти посилено шукають рішення для спрощення інфраструктури шляхом усунення бар'єрів, які сьогодні істотно утрудняють їхня здатність швидко реагувати на потребі бізнесу. З побажань клієнтів стає зрозуміло, що в багатьох випадках побудова хмари стає пляшковим горлечком, і що клієнтам потрібний комплексний підхід для його проходження. Це саме і є те, чого планується досягти за допомогою VMware Cloud Foundation.

Простіше говорячи, Cloud Foundation інтегрується з vSphere, VSAN і NSX у єдиний уніфікований стек, що може надаватися на вимогу або запускатися у форматі «as a service» у хмарі.

Секретний інгредієнт, що дозволяє зібрати таку платформу, це VMware SDDC Manager, ключовий компонент Cloud Foundation, що дозволяє хмарним адміністраторам управляти програмним забезпеченням хмарної інфраструктури як єдиним стеком. SDDC Manager забезпечує наскрізну автоматизацію життєвого циклу програмного забезпечення хмарної інфраструктури, скорочуючи час, необхідне для побудови й роботи хмарного середовища більш ніж на 80% у порівнянні із традиційними підходами. Перші польові випробування довели, що клієнти можуть створити приватне хмарне середовище за лічені годинники.

З погляду архітектури Cloud Foundation створює платформу програмного забезпечення гіперконвергентної інфраструктури наступного покоління шляхом

об'єднання обчислювальних ресурсів, систем зберігання й мережний віртуалізації в споконвічно інтегрований стек від одного постачальника. Це набагато більше, ніж може запропонувати будь-який постачальник гіперконвергентної інфраструктури. Клієнт не тільки одержує вигоду від найбільш зріла й надійного стека віртуалізації в галузі, але й значну економію засобів (30-40% у порівнянні з іншими гіперконвергентними рішеннями у всіх флеш-розгортаннях) укупі з перевагами NSX.

Ще одна унікальна особливість VMware Cloud Foundation – це розгортання на вимогу й уперше робота у форматі «as a service» у хмарі. Для приватної хмари, клієнти можуть сполучати Cloud Foundation з поруч кваліфікованих VSAN Ready Nodes від Dell, QCT і HP і комутаторів Cisco і Arista, або закупити систему VCE VxRack 1000 SDDC «під ключ».

Ми все знаємо, як віртуалізація серверів спростила надання обчислювальних ресурсів для додатків. Компанія VMware поклала початок цієї революції в області ЦОД. З Cloud Foundation ми піднімаємося на наступний рівень і можемо гранично просто надати програмно-визначаємий ЦОД на вимогу в хмарі. Це важливий крок уперед, що дозволяє повною мірою скористатися стеком програмно-визначаємих ЦОД VMware для побудови гібридної хмари.

Intel SDI

Гігант напівпровідникової індустрії Intel створює мікрочипи для величезної безлічі серверів, мережних пристроїв і систем зберігання даних корпоративного класу, які використовуються в дата-центрах по усьому світі. Той факт, що багато сторонні вендори орієнтуються на референс-дизайни різних IT-систем, розроблювальні інженерами Intel, а також активно використовують продукти чипмейкера, робить нову ініціативу американської корпорації, що одержала назва SDI (Software Defined Infrastructure або Програмно-визначаєма інфраструктура), украй цікавої й здатної в перспективі докорінно змінити індустрію ЦОД.

Проект Intel SDI, перша інформація про яке з'явилася ще восени минулого року під час виставки-конференції Intel Developer Forum 2016 (IDF13), припускає створення основної архітектури для обчислювальних систем майбутнього. Реалізується він у рамках ініціативи Facebook Open Compute Project (OCP), учасником якої є американський чипмейкер. На цей момент до складу SDI були включені пропрієтарні архітектури, використовувані IBM, що також є учасником OCP, і Oracle Corporation, що робить проект Intel ще цікавіше.

Уже зараз очевидно, що індустрія ЦОД стоїть на порозі масштабного перехідного періоду, що зведеться до виходу архітектури обчислювальних систем на якісно новий рівень. Сприяти цьому будуть впровадження хмарних технологій, повсюдне поширення інтернету речей (IoT), а також потреба корпоративних клієнтів у більше ефективних інструментах для бізнес-аналітики в режимі реального часу й обробки більших даних.

На думку експертів, дата-центри, усередині яких установлені обчислювальні системи на базі поточної архітектури, більше не здатні задовольняти всі потреби великих корпорацій і їхніх клієнтів за умови збереження доступної ціни IT-сервісів. При цьому ЦОД стають усе менш екологічними. Ситуацію потрібно міняти. Варто підвищити енергоефективність обчислювальної інфраструктури, домогтися мінімізації капітальних і експлуатаційних витрат, а також забезпечити гнучкість і масштабованість дата-центрів, щоб вони могли мінятися «на лету», пристосовуючись до динамічного бізнесу-середовища.

Це цілком реально, адже в IT-світі все міняється дуже швидко. Ще десять років тому ніхто не чув про Facebook або Twitter, тоді як соцмережа MySpace була справжнім феноменом. В 2008 році Nokia належала 50-процентна частку ринку смартфонів, у той час як Apple контролювала лише 3% цього ринку. Хто б міг подумати, що ЦРУ буде клієнтом Amazon, використовуючи IT-сервіси хмарного гіганта, а також, що багато споживачів зволіють соцмережу Facebook телебаченню. Зміна парадигми відбувається на наших очах.

Сучасна індустрія ЦОД на готова до майбутнього

На сьогоднішній день IT-індустрія, по більшій частині, є пропрієтарною і негнучкою. Якщо ви купуєте мережне встаткування в таких компаній як Juniper або Cisco, вам навряд чи вдасться при необхідності замінити частина кампусної мережі за допомогою апаратного забезпечення, зробленого сторонніми вендорами. А що щодо систем зберігання даних? Знову те ж саме: внесення змін у вже розгорнуту мережу зберігання даних (SAN) на базі встаткування EMC або NetApp з використанням продуктів сторонніх вендорів буде непростим завданням.

У цей час саме сервери на базі процесорів з архітектурою x86 є найбільш гнучким з елементів IT-інфраструктури дати-центра. Ви можете обробляти на них по кілька додатків. Ви можете віртуалізувати їх, і запускати безліч додатків на віртуальних машинах. На ринку представлені серверні системи широкого спектра великих вендорів начебто Dell, HP, Lenovo, а також сотні варіантів «білих коробок» (сервери немарочного складання), які також мають сумісність із більшістю корпоративних додатків.

У сегменті інструментів для віртуалізації серверів всі не так гладко: у цей час тут домінує VMWare. Проте, є альтернативи начебто Xen або KVM – головне, щоб корпоративне програмне забезпечення їх підтримувало.

Тому що ж нам домогтися підвищення гнучкості дата-центрів і підготувати їх до майбутнього? Деякі експерти пропонують використовувати для цього концепцію SDDC (Software Defined Datacenter або Програмно-визначаємий центр обробки даних).

Intel робить ставку на SDI і SDDC

Ви можете розглядати SDDC як концепцію, що припускає стандартизацію не тільки серверів, але й систем зберігання даних, а також мережного встаткування усередині ЦОД. Крім того, дана технологія дає можливість розгортання, керування й оркестровки всієї фізичної, віртуальної й логічної IT-інфраструктури за допомогою єдиного інструмента.

Так, все це в значній мірі можна назвати IT-Утопією, при цьому від подібного світлого майбутнього нас відокремить пристойний відрізок часу. Щоб усе зложилося саме таким чином, багато технологій повинні бути об'єднані. Але перехідний період уже почався. Великі постачальники високотехнологічного встаткування (не тільки Intel) вкладають величезні кошти в SDDC-проекти, тому що бажають бути готовими до майбутнього й не хочуть залишатися за конкурентами.

Концепція Intel SDI розширює визначення SDDC. Це переоцінка системної архітектури, драйверами якої виступають бізнес-процеси, робочі навантаження й конкретні додатки, а не асортименти «заліза», доступного для покупки на даний момент. SDI може трансформувати мейнстримні корпоративні дати-центри. У нового проекту Intel є величезний потенціал. На думку експертів, з його допомогою нам удасться повністю витиснути з ринку класичні мейнфрейми й системи обробки транзакцій високої доступності вже до кінця цього десятиліття.

На даний момент ви поки ще нічого не зможете купити з «каталог» SDI-рішень від Intel, але ви можете, і ви повинні підготуватися до появи таких продуктів. Першим кроком на шляху до SDDC є знаходження розуміння основ « програмне-обумовлених» архітектур, а також того, як оркестровка й комбінування мультивендорного «заліза» можуть сполучатися з поточною інфраструктурою дати-центра. Крім того, необхідно зрозуміти, як всі ці технології можуть вплинути на архітектуру ЦОД у найближчі квартали. Другим кроком є відстеження прогресу Intel у роботі над проектом SDI протягом найближчих двох років. У цьому випадку також варто приділяти підвищену увагу напрямкам розвитку корпорації Intel, а також інвестиціям чипмейкера в програмне забезпечення й нові архітектури масштабу окремої стійки.

Ми рекомендуємо ознайомитися із програмним рішенням Intel SDI першого покоління, коли і якщо таке з'явиться. Схоже, що це буде досить ефективний інструмент для підвищення гнучкості корпоративних дата-центрів.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Python – це потужна мова програмування, яка проста у вивченні. Він має ефективні структури даних високого рівня та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис і динамічна типізація Python разом з його інтерпретованим характером роблять його ідеальною мовою для створення сценаріїв і швидкої розробки додатків у багатьох сферах на більшості платформ.

Інтерпретатор Python і обширна стандартна бібліотека доступні у вихідному або двійковому вигляді для всіх основних платформ на веб-сайті Python https://www.python.org/ і можуть вільно поширюватися. Цей же сайт також містить дистрибутиви та вказівники на багато безкоштовних сторонніх модулів Python, програм і інструментів, а також додаткову документацію.

Інтерпретатор Python легко розширюється за допомогою нових функцій і типів даних, реалізованих у C або C++ (або інших мовах, які можна викликати з C). Python також підходить як мова розширення для налаштовуваних програм.

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 центру обробки даних SSDC.

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

Структура програмно-визначаємого ЦОД

«Нижній шар» SDDC будуть становити як успадковані компоненти інфраструктури ЦОД, так і більше нові програмно-визначаємі компоненти на базі стандартних серверів. Ресурси цього інфраструктурного шару надаються вишележачому рівню сервісів SDDC через єдиний набір інтерфейсів взаємодії й подання ресурсів ЦОД. Над рівнем сервісів надбудовується рівень керування сервісами; ще вище перебувають інструменти керування політиками доступу й сервісний каталог.

«Верхній шар» програмно-визначаємого ЦОД представлений порталом самообслуговування й прикладних інтерфейсів для персоналізованої конфігурації пакета сервісів під замовника.

Для деяких з перерахованих шарів уже існують продукти на ринку, але єдиної конструкції поки не існує. Численні поки рішення на базі відкритих стандартів, у тому числі OpenStack, швидше за все згрупуються в невелику кількість пакетів, що розвиваються співтовариствами взаємодоповнюючих один одного розроблювачів.

Інженерна основа

При всіх перевагах програмно-визначаємих рішень не можна забувати, що вони не будуть надійно функціонувати без якісних інженерних інфраструктур і систем фізичної безпеки. Обстеження інженерної інфраструктури було проведено в 187 дата-центрах. Всі власники ЦОД заявляють про рівень надійності своїх інфраструктур на рівні не нижче Tier III. На ділі абсолютна більшість із них досить уваги приділяють системам кондиціювання й безперебійного електроживлення, але мало – питанням контролю доступу й приєднання до

зовнішніх розподільних мереж. Тому вторинному провайдеру (який розгортає послуги для кінцевих замовників, користуючись площадками сторонніх ЦОД) для підвищення надійності варто користуватися площадками декількох первинних провайдерів. Не виключено, що в міру росту популярності програмно-визначаємих ЦОД в області SDDC з'являться стандарти

Отже, експерти відзначили наступне:

– готового SDDC-рішення, придатного для всіх категорій замовників, на ринку сьогодні ні, і навряд чи воно з'явиться в найближчій перспективі;

– SDDC-рішення – складне. Для його розгортання й підтримки потрібні професійні послуги.

У міру появи нових розробок і продуктів поняття «програмно обумовлені системи зберігання» стало здобувати усе більше чіткі обриси. У числі прикладів – системи зберігання даних OpenStack, EMC ViPR, Nexenta і HP StoreVirtual.

Усе більше виробників СЗД називають свої рішення «програмно обумовленими», однак кожний з них вкладає в поняття SDS свій зміст і зорієнтуватися в цьому непросто. До того ж усе більше функцій СЗД реалізується саме на програмному рівні. Які системи дійсно ставляться до класу SDS, а які представляють уже відомі раніше технології «у новій обгортці»? Головним в SDS уважається абстрагування сервісів зберігання даних від апаратного рівня – фізичного встаткування.

На новому ринку SDS уже можна виділити кілька груп вендорів. Перша по суті пропонує нове покоління засобів віртуалізації систем зберігання: сервіси зберігання абстрагуються від фізичного встаткування й можуть застосовуватися до апаратних рішень різних виробників. Іноді вони реалізуються у вигляді відкритих платформ, щоб незалежні розроблювачі могли додавати нові сервіси зберігання у свої продукти (за принципом магазина додатків). Даний підхід рятує замовника від прив'язки до конкретної апаратної платформи й програмних сервісів.

Друга група вендорів іде ще далі – по шляху абстрагування контролерів систем зберігання. Це дозволяє задіяти ресурси зберігання вже наявних систем і спростити операції, однак все уведення-вивід здійснюється через віртуалізуючий контролер, тому він повинен мати високу пропускну здатність і продуктивність. Крім того, його потрібно резервувати.

Третя група розроблювачів SDS абстрагує керування СЗД, але задіє вже реалізовані ними функції. Наприклад, якщо потрібно зробити знімок тому, на центральній консолі вводиться відповідна команда, що передається для виконання системі зберігання. Хоча знімок тому різних дискових масивів можна виконувати однією командою, однаково для цього використовується сервіс конкретної СЗД. У результаті спрощуються операції, до того ж, на відміну від другого підходу, не вноситься додаткова затримка через комунікації між СЗД, а для виконання операцій не потрібно висока обчислювальна потужність. А при відмові консолі керування SDS дані однаково будуть доступними.

Як звичайно, вибір оптимального варіанта залежить від потреб. Якщо необхідні додаткові сервіси зберігання даних і планується закупівля нових систем зберігання, то перші два підходи переважніше. Коли потрібна консолідація ресурсів зберігання й операцій, допоможе третій варіант. Якщо ж потрібно просто обновити інфраструктуру зберігання й уніфікувати її, то максимальні вигоди від SDS можна одержати в «хмарних» ЦОД з високим ступенем віртуалізації.

Приклади SDS

Ряд провідних виробників СЗД уже «устали на шлях SDS», а деякі навіть випустили комерційні продукти. Звичайно розробки в області SDS базуються на системах віртуалізації СЗД, що розвиваються вже багато років, але іноді представляють і принципово нові рішення.

В IBM як перспективна платформа для хмарних і програмно конфігуруємих середовищ розглядається оновлене сімейство систем зберігання даних IBM Storwize. Випущена в жовтні модель IBM Storwize V5000 по

продуктивності й функціональності зайняла проміжне положення між моделями початкового рівня V3700 і старшими системами серії V7000.

V5000 може віртуалізувати СЗД різних вендорів, поєднуючи їх у пул ресурсів. Ця технологія знайома користувачам по IBM SVC. У числі її функцій – розподіл даних по рівнях зберігання (Easy Tier), оптимізоване резервне копіювання (FlashCopy), забезпечення катастрофостійкості (Metro Mirror і Global Mirror) і підтримка зовнішньої віртуалізації (External Virtualization), що дозволяє переміщати в V5000 дані з дискових масивів інших виробників. Однак система не уніфікована, тобто підтримується лише блоковий доступ.

Автоматична оптимізація допомагає скоротити витрати й підвищити віддачу від інвестицій, зокрема, завдяки спрощенню роботи системного адміністратора за рахунок його розвантаження від рутинних операцій. ПЗ SmartCloud Virtual Storage Center і Active Cloud Engine забезпечують інтеграцію у віртуальні середовища, керування інформацією й міграцію даних відповідно до заданих правил. Tivoli Storage Productivity Center ME здійснює аналіз і моніторинг системи, а IBM Tivoli Storage FlashCopy Manager – резервне копіювання й відновлення даних.

У поданні IBM, в SDS робоче навантаження визначає вимоги до ємності систем зберігання даних, їхньої продуктивності й інших параметрів, засобу оркестровки планують розподіл ресурсів для виконання угод про рівні обслуговування, а API підтримує взаємодія з компонентами інфраструктури. Це припускає поділ функцій керування й зберігання й застосування більше простих по функціональності систем зберігання. Однак поки що виробники, як уважають в IBM, пропонують лише фрагментарні рішення, а сама компанія перебуває на початковому етапі розробки програмно конфігуруємого зберігання, що характеризується віртуалізацією, консолідацією й оптимізацією СЗД.

Поряд з контролерами віртуалізації до SDS ставиться ще один клас рішень – віртуальні пристрої зберігання (Virtual Storage Appliance, VSA). Яскравим прикладом є HP StoreVirtual VSA – віртуальна програмно обумовлена

система зберігання, що підтримує основні стандартні сервери й різні гіпервізори. HP розвиває її з 2007 року, і за цей час було впроваджено понад 170 тис. віртуальні системи зберігання. HP StoreVirtual VSA являє собою ПЗ з відкритими API для керування й оркестрації, установлюване на стандартні сервери відкритої архітектури й що дозволяє створювати загальні системи зберігання на базі внутрішніх або дисків, що підключаються до серверів. Однак компанія пропонує й готові програмно-апаратні платформи x86 (Converged Storage Appliance, CSA).

Для керування й оркестрації горизонтально масштабованих платформ CSA можна застосовувати продукти HP або її партнерів. VSA і CSA з'явилися в HP у результаті придбання компанії LeftHand. Нові версії продуктів даного сімейства (наприклад, система зберігання HP StoreVirtual 4335) мають адаптивні функції оптимізації й автоматичним розподілом навантажень між підтомами різних класів.

Відповідно до оцінок HP, у порівнянні з типовою архітектурою SDS на 80% знижує витрати на системи зберігання, на 60% скорочує енерговитрати й на 50% зменшує займану системами площа. До класу SDS ставиться також HP StoreOnce VSA – незалежне від апаратного забезпечення рішення дедуплікації й реплікації резервних копій. Як затверджується, це програмне рішення значно знижує витрати на захист даних, особливо у випадку використання його в IT-інфраструктурі великих і середніх підприємств. Воно дозволяє обійтися без виділеного апаратного забезпечення для резервного копіювання й забезпечує керування всім середовищем резервного копіювання за допомогою єдиної консолі. StoreOnce розгортається як віртуальна машина на сервері й підтримує СЗД різних вендорів.

Однієї з найбільш резонансних нових розробок в області SDS стала платформа EMC ViPR, представлена на конференції EMC World у травні 2016 року. За задумом її творців, платформа ViPR здатна спростити роботу замовників, забезпечивши наступні переваги:

– абстрагування ресурсів зберігання від існуючої інфраструктури й об'єднання їх у стандартизовані віртуальні пули зберігання;

– автоматизацію процесів виділення ресурсів, захисту даних і керування змінами;

– надання доступу до ресурсів зберігання за принципом самообслуговування;

– високу масштабованість;

– динамічну адаптацію до нових сценаріїв використання й навантаженням.

Це рішення передбачає можливості інтеграції із серверами, оснащеними ПЗ VMware, OpenStack і Microsoft, а також з масивами інших виробників (наприклад, NetApp), підтримує програмні інтерфейси REST API для інтеграції з Amazon S3, OpenStack Swift і EMC Atmos, дозволяє створювати сервіси керування даними, що поєднують блокові, файлові й об'єктні ресурси зберігання. У жовтні EMC оголосила про комерційну доступність ViPR для замовників, розроблювачів і партнерів по усьому світі.

По суті ViPR – програмний пакет, що поєднує засоби віртуалізації СЗД із файловим, блоковим і об'єктним доступом і засобу керування зберіганням. Його характерні риси – поділ «шляхи даних» і «шляхи керування», а також підтримка об'єктних СЗД. Найбільш близькі рішення – DataCore SANsymphony (платформа віртуалізації, що поєднує СЗД у середовище Virtual SAN і підтримуюча при цьому традиційні жорсткі диски, флеш-накопичувачі й хмарні сховища), HDS VSP, IBM SVC, NetApp V-Series, а також Melio SANbolic, StorMagic SvSAN і інші VSA.

За допомогою API рішення EMC ViPR здійснює об'єднання СЗД різних вендорів, створює віртуальні пули ресурсів, управляє ними, надає користувачам портал самообслуговування для замовлення послуг, реалізує розподіл ресурсів на вимогу. При цьому «шлях керування» відділений від «шляху даних»: сервери й додатки можуть обмінюватися інформацією з фізичними пристроями прямо,

минаючи шар віртуалізації. Це дозволяє використовувати розширені функції без зниження рівнів надійності й продуктивності комплексу. «Шлях даних» забезпечує захист інформації, резервне копіювання, катастрофостійкість, міграцію даних, підтримку різних додаткових протоколів доступу й ін.

Ще одна новинка 2024 року – нова версія операційної системи Clustered Data ONTAP, анонсована компанією NetApp у липні. Одне з нововведень версії 8.2 – логічно ізольовані віртуальні машини зберігання даних (Storage Virtual Machines, SVM), що забезпечують керування якістю сервісу (QoS) для кожного користувача (наприклад, підрозділу організації). Це рішення розглядається в NetApp як основа SDS. SVM нагадує колишню технологію NetApp vFiler (Virtual Filer), але віртуальні машини SVM повністю відділені від апаратних платформ – їх можна переміщати між системами зі збереженням заданих параметрів QoS.

В області SDS працюють чимало недавно створених компаній. По оцінках аналітиків, за останні два роки венчурні інвестиції в цей сегмент світового ринку перевищили 300 млн доларів.

Отже, SDS звичайно трактується як інфраструктурне програмне забезпечення, що надає набір взаємодіючих служб зберігання даних, доступних додаткам і менеджерам робочих навантажень. Сервіси зберігання даних покликані забезпечувати потреби основних типів додатків, можуть використовувати різні платформи й/або носії або інтегруватися із серверами або виділеними системами зберігання. Вони доступні через каталог і управляються оркестратором. SDS забезпечує автоматизоване керування зберіганням, QoS продуктивності (пропускна здатність, IOPS, затримка), вибір правил резервного копіювання й DR, необхідний рівень безпеки.

В ESG виділяють наступні переваги SDS:

– Простота, що досягається завдяки централізованому керуванню й моніторингу в реальному часі. Кращі реалізації SDS забезпечують просту інтеграцію й роботу з різними системами зберігання, серверами й гіпервізорами.

– Економічність, забезпечувана за рахунок спрощеної підтримки й більше повного використання ресурсів зберігання із загального пула.

– Ефективність, що виражається в простому переміщенні даних між системами зберігання й виділенні ресурсів зберігання серверам.

Але SDS в остаточному підсумку – усього лише термін, що позначає напрямок еволюції СЗД. Відомий уже не один рік принцип абстрагування сутностей і керування ними на різних рівнях з використанням автоматизації знаходить усе більше широке застосування. Неспроста торік стали говорити про SDE – Software-Defined Everything. Хоча «програмно обумовлене всі» – скоріше, жартівний термін, але в кожному жарті є частка правди.

SDS – одночасно й новий, і старий підхід. Як уважають аналітики The 451 Group, він здатний вплинути на корпоративну інфраструктуру зберігання даних завдяки можливості виконання багатофункціонального програмного забезпечення на широко розповсюджених апаратних платформах. Однак у той час як ряд вендорів, включаючи HP, IBM і NetApp, уже підхопили ідею програмно обумовлених систем зберігання й пропонують власні рішення, інші, у числі яких Fujitsu, воліють іменувати свої СЗД «бізнес-орієнтованими», розуміючи під цим, зокрема, можливість завдання для додатків пріоритетів (наприклад, щоб одержати бажаний час відгуку) – інше система робить автоматично.

Незалежно від термінології галузь рухається в одному напрямку, і цей шлях не буде гладким. Ідея поділу функцій керування й зберігання, а також застосування більше простих по функціональності систем зберігання в остаточному підсумку може привести до конфлікту між комерційними інтересами виробників і технологічних рішень, що вже можна спостерігати на прикладі SDN: деякі вендори замість відкритого підходу сьогодні пропонують власні пропрієтарні розробки.

3.2 Розробка структурної схеми

SDDC передбачає можливості створення пулів ресурсів і гнучкого їхнього використання у вигляді сервісів. Для спрощення процесів виділення й звільнення ресурсів концепція серверної віртуалізації повинна бути поширена на мережу й системи зберігання даних (див. рисунок 3.1). Найважливіша роль при цьому приділяється керуванню. Завдання програмного керування – забезпечити гнучкість і масштабованість SDDC для швидкої адаптації інфраструктури IT до вимог нових додатків.

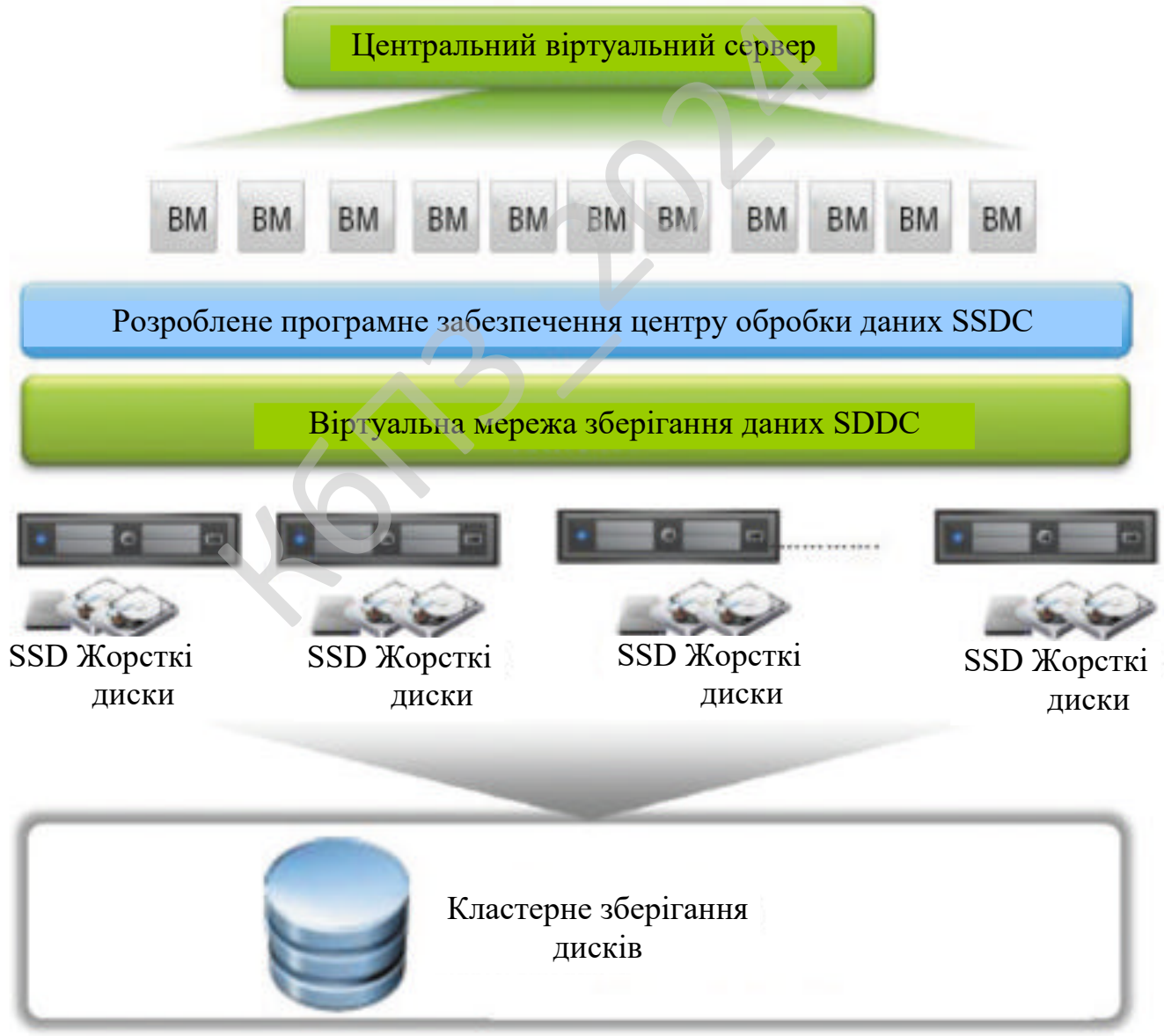

Рисунок 3.1 – Структурна схема системи

SDS у контексті SDDC

Частиною SDDC є програмно обумовлене зберігання даних (Software Defined Storage, SDS). У звіті IDC «Worldwide Software-Based (Software-Defined) Storage Taxonomy, 2016» затверджується, що програмно обумовлені СЗД будуть повільно, але вірно перетворюватися в найважливіший компонент будь-якого ЦОД (як програмно обумовленого, так і традиційного), де потрібно більш ефективно зберігати й обробляти дані.

Суть програмно обумовленого ЦОД складається у використанні «інтелекту» інфраструктури IT для надання сервісів. У випадку СЗД це сервіси зберігання даних. Однак, на відміну від програмно конфігуруємих мереж (Software-Defined Networking, SDN), також граюча важлива роль у програмно обумовлених ЦОД, програмно обумовлені (або централізовано конфігуруємі) системи зберігання припускають не поділ потоків даних і керування, а абстрагування встаткування зберігання даних від керуючого ПЗ. У результаті об'єднані в пул ресурси зберігання можна автоматично розподіляти на рівні програмного забезпечення відповідно до вимог додатків.

При такій моделі «інтелект» і функціональність СЗД переміщаються на рівень програмного забезпечення, що дозволяє централізовано управляти різнорідними ресурсами зберігання – від високопродуктивних дискових масивів до недорогих систем зберігання. Таким чином, SDS припускає перехід від твердої архітектури до програмувального, котра повинна забезпечувати динамічність і високу економічну ефективність. При цьому всі основні функції (дедуплікація, реплікація, динамічне виділення ресурсів і створення миттєвих знімків) виконуються програмно.

Завдяки поділу програмного й апаратного рівня можна купувати й розгортати різнорідні системи зберігання, не турбуючись про проблеми сумісності, неповного використання ресурсів конкретної СЗД і складностях ручного адміністрування ресурсів зберігання: як і в SDDC, центральне місце тут займає автоматизація.

Варто відзначити, що вже сьогодні основні зміни в області систем зберігання зв'язані саме з ПЗ, а не з апаратною частиною, і це підтверджують останні анонси ведучих вендорів СЗД. «Інтелект» системи зберігання реалізується на рівні програмного забезпечення – її функціональність «іде в ПЗ».

Від пула ресурсів до стандартизації

Відповідно до найпоширенішої точки зору SDS буде сприяти максимально ефективному використанню наявних систем зберігання й ефективному керуванню різнорідними ресурсами. Однак деякі виробники пропонують позбутися від успадкованих СЗД і замінити їх «стандартними» дисковими масивами. У їхньому поданні реалізація концепції SDS означає не тільки віртуалізацію й автоматизацію СЗД, але й стандартизацію. Передбачається, що програмне забезпечення SDS повинне працювати на відкритому, типовому встаткуванні (commodity), у той час як у традиційних системах мікропрограми розробляються для спеціалізованих мікросхем (ASIC), мікропроцесорів або контролерів.

Але що таке «стандартні СЗД»? Практично те ж саме, що й стандартні сервери. Так, на думку деяких експертів, першим кроком до SDS стало впровадження в СЗД флеш-пам'яті PCIe. Це дозволило значно підвищити продуктивність додатків і замінити дорогі пропрієтарні дискові масиви більше дешевими стандартними серверами із флеш-накопичувачами. Не секрет, що багато дискових масивів реалізуються на платформі x86 і оснащуються стандартними інтерфейсними модулями й накопичувачами. Наприклад, компанія EMC недавно випустила дисковий масив XtremIO, побудований повністю на базі флеш-памяти. Його апаратна платформа XtremIO фактично складена зі стандартних компонентів. Така архітектура дозволяє легко модернізувати систему – перейти на SSD більшої ємності й більше сучасні процесори й інтерфейси.

Тим часом більшість так званих серверів зберігання даних не мають всі ознаки програмно обумовленого зберігання й не є рішенням рівня підприємства.

До того ж «стандартизація» СЗД означала б заміну наявної в замовника інфраструктури зберігання, що пов'язане із серйозними капітальними витратами. Тому в якості першого практичного кроку пропонується інше рішення, а саме віртуалізація ресурсів зберігання даних. Це дозволяє об'єднати різнорідні ресурси в пул, що дає можливість по-новому використовувати наявні системи.

Крім того, SDS припускає об'єднання ресурсів зберігання на різних носіях (дисках, флеш-накопичувачах, магнітних стрічках) і в зовнішніх хмарах. А за допомогою системи керування адміністратор може одержати цілісне подання про те, що відбувається у всій цій інфраструктурі, і централізовано управляти нею. IDC відзначає, що рішення SDS повинне підтримувати повний спектр сервісів зберігання, еквівалентний традиційним апаратним платформам, але який охвачує всю інфраструктуру зберігання. У числі прикладів – сервіси міграції даних, реплікації й багаторівневого зберігання.

SDS визначає, як організовані дані, де перебуває місце їхнього постійного зберігання (DAS, SAN, NAS, хмарне або об'єктне сховище), який вид сервісів пропонується. Для замовника SDS може виглядати, як програмний пакет, що забезпечує віртуалізацію ресурсів і керування, як спеціальна програмно-апаратна платформа (appliance) або навіть як набір хмарних сервісів. (Відомі хмарні сервіси зберігання даних – ще один «пробний камінь» на шляху до програмно обумовленого зберігання.)

Керування здійснюється на основі заданих правил, а віртуальним машинам привласнюються атрибути зберігання (VM Storage Policy) – наприклад, ємність, продуктивність, доступність. Виходячи із цих атрибутів, ВМ надаються сервіси зберігання. Для цього використовувана платформа віртуалізації повинна підтримувати специфічні апаратні функції СЗД і забезпечувати операції ВМ із даними.

На думку аналітиків Evaluator Group, SDS дозволить поширити такі засоби систем зберігання, як знімки даних, реплікація або RAID на різнорідні дискові масиви, і доповнити їх новими коштовними функціями. Абстрагування сервісів

зберігання відкриває можливість для централізованого адміністрування й керування, а в традиційній архітектурі це робиться окремо для кожного дискового масиву.

У дійсності програмно обумовлене зберігання далеко не нове явище. Схожі підходи були реалізовані ще десять років тому: наприклад, компанією IBM у її продукті SUN Volume Controller (SVC) і HDS в Virtual Storage Product (VSP) – у цих рішеннях підтримуються можливості підключення до «контролера» СЗД систем зберігання різних вендорів і об'єднання їх у єдиний пул ресурсів. Однак реалізована ними віртуалізація систем зберігання даних і запропоновані засоби оптимізації, наприклад приоритизація уведення-виводу для конкретних додатків, – тільки перший крок на шляху до SDS, далі вдалося просунутися лише деяким вендорам.

До вигід, одержуваним від впровадження віртуалізованих СЗД, відносять раціональне використання ресурсів зберігання і їхню консолідацію, спрощене керування, розширення функціональності й підвищення продуктивності дискових масивів, підтримку реплікації й переміщення даних без зупинки сервісів, додавання багаторівневого зберігання до наявних ресурсів. Однак у такій архітектурі можуть виникати вузькі місця й потенційні крапки відмови, а інтеграція з додатками здійснюється не повною мірою.

Кінцева мета – перехід до відкритих платформ, що набудовується відповідно до вимог додатків. Її досягнення можливо лише в тому випадку, якщо будуть прийняті, схвалені й впроваджені галузеві стандарти, що регламентують відкриті рішення, які здатні працювати «поверх» пропрієтарних технологій.

Проблеми SDS

По суті, SDS – це ще один крок на шляху до хмари, але реалізувати цю гарну ідею непросто. Успадковані традиційні СЗД поряд з ризиками переходу на повністю нову платформу – серйозні перешкоди на шляху до програмно обумовлених систем зберігання. Та й віртуалізація СЗД поки відстає від віртуалізації серверів і мереж, а функціональність систем зберігання задіється

гіпервізорами лише почасти. Одна із причин – відсутність стандартизації. У системах зберігання звичайно використовується патентоване ПЗ контролерів, а інфраструктурою зберігання складно управляти, та й масштабувати її, як правило, непросто. Навіть якщо в ЦОД застосовуються СЗД одного вендора, для керування ними можуть знадобитися різні програмні засоби. У різнорідному середовищі завдання виявляється ще більш важкою.

Через наявність декількох крапок керування й різних підходів до керування СЗД ускладнюється керування віртуалізованими хмарними ЦОД і утрудняється одержання єдиної цілісної картини середовища зберігання даних, не говорячи вже про додаткові витрати, а виділення й вивільнення ресурсів зберігання перетворюється в багатоетапний процес, що вимагає участі адміністраторів серверів і адміністраторів систем зберігання. Програмно обумовлені СЗД покликані перебороти ці проблеми на шляху до хмарних ЦОД.

Як уже пояснювалося, SDS припускає абстрагування успадкованих ресурсів SAN і NAS і створення пула ресурсів. Але для цього буде потрібно об'єднати в пул системи з файловим і блоковим доступом без втрати функціональності відповідних СЗД, централізувати розподіл ресурсів, доступ і керування, а також забезпечити можливість додавання нових типів систем зберігання. Іноді ця модель охоплює й системи DAS. Однак ефективно управляти трафіком передачі даних у такій інфраструктурі непросто – неминуче виникають вузькі місця, що веде до втрати продуктивності й зниженню рівня доступності додатків.

Так звані сервіси зберігання даних повинні являти собою якусь комбінацію сервісів реплікації, знімків даних, шифрування, дедуплікації й т.д., але інтегрувати різнорідні пропрієтарні API, призначати сервіси даних СЗД різних вендорів і класів систем зберігання до того ж повністю використовувати можливості задіяних успадкованих СЗД дуже складно.

Включення в архітектуру зберігання додаткових видів СЗД порушує цілісність її подання. Така комбінація (наприклад, DAS на границі

інфраструктури зберігання й системи SAN/NAS у її центрі) дозволить ефективно використовувати наявну ємність, але одержати точну загальну картину такого середовища зберігання навряд чи вдасться. Крім того, завдання розподілу ресурсів багаторазово ускладнюється, а збільшення кількості крапок керування й інструментальних засобів веде до появи вузьких місць через технологічні обмеження. Автоматизувати розподіл ресурсів непросто, а при ручному керуванні росте ймовірність помилок, що виникають із вини системних адміністраторів. Але ж саме цього вендори прагнуть уникнути за допомогою SDS.

Для реалізації концепції SDS потрібні єдина крапка керування, відкриті API, простий процес виділення ресурсів, порівнянний за часом і трудомісткістю зі створенням віртуальної машини, централізований моніторинг і керування. Службові сервіси (автентифікації, обліку споживаних ресурсів і ін.) могли б допомогти IT-підрозділам і провайдерам сконцентрувати сили для створення нових послуг.

3.3 Розробка функціональної схеми

Щоб створена функціональна модель придбала закінчений характер розроблену на попередньому етапі структурну схему, доцільно доповнити функціональною схемою.

Моніторинг стану серверів центру обробки даних SDDC

Система балансування навантаження центру обробки даних SDDC постійно відслідковує рівень навантаження й стан довірених їй серверів центру обробки даних SDDC для того, щоб на підставі зібраної інформації в будь-який момент надати клієнтові доступ до того сервера центру обробки даних SDDC, що зможе щонайкраще відповісти на його запит. При цьому використовується два методи контролю: зовнішній моніторинг і внутрішній моніторинг.

При проведенні зовнішнього моніторингу система балансування навантаження центру обробки даних SDDC розраховує час відгуку сервера центру обробки даних SDDC, для чого направляє на сервер запит і замірить час відповіді. Найпростіша техніка виконання зовнішнього моніторингу сервера центру обробки даних SDDC припускає використання ping-тестів за протоколом керування повідомленнями Internet Control Message Protocol (ICMP). Ці тести дозволяють системі переконатися в готовності сервера центру обробки даних SDDC до роботи й довідатися, скільки часу необхідно для передачі інформації із сервера центру обробки даних SDDC на систему балансування й назад. Якщо система балансування навантаження центру обробки даних SDDC не одержує відгуку від сервера центру обробки даних SDDC після декількох послідовних запитів, уважається, що даний сервер недоступний. Як правило, адміністратори підключають сервери безпосередньо до системи балансування навантаження центру обробки даних SDDC, тому якщо час, затрачуваний на передачу, занадто велике, система містить, що сервер працює з великим навантаженням.

Тут, однак, треба відзначити, що в ході ping-тестів сервера центру обробки даних SDDC за протоколом ICMP діагностується тільки стек протоколів IP; описаний метод не дає подання про стан стека TCP, що використовується протоколом передачі гіпертексту HTTP. Щоб переконатися в правильності функціонування стека TCP сервера центру обробки даних SDDC, засіб балансування навантаження центру обробки даних SDDC вживає спробу встановити з'єднання за протоколом TCP, для чого потрібно здійснити обмін, що складається із трьох етапів, що підтверджують повідомленнями. Робиться це так. Спочатку засіб балансування навантаження центру обробки даних SDDC направляє серверу TCP-пакет, у якому значення біта SYN установлено рівним 1. Якщо після цього система балансування одержує від сервера центру обробки даних SDDC TCP-пакет, у якому значення біта SYN дорівнює 1, а значення біта ACK теж установлено рівним 1, вона направляє серверу другий TCP-пакет зі значенням біта SYN рівним 0 і значенням біта ACK рівним 1. Якщо обмін

підтверджувальними повідомленнями завершився успішно, виходить, TCP-стек сервера центру обробки даних SDDC функціонує нормально. По завершенні такого обміну засіб балансування навантаження центру обробки даних SDDC негайно розриває з'єднання із сервером, щоб виключити непродуктивне використання його ресурсів. Якість TCP-з'єднання із сервером оцінюється системою по такому показнику, як час, необхідне для виконання всіх трьох етапів обміну підтверджувальними повідомленнями.

Поряд з тестуванням стеків протоколів, кращі засоби балансування навантаження центру обробки даних SDDC можуть забезпечувати моніторинг часу відгуку й готовності як самого сервера центру обробки даних SDDC, так і встановлених на ньому додатків ще одним способом: на сервер направляється запит за протоколом HTTP на одержання інформаційних матеріалів або адреси URL. Нехай ім'ям початкової сторінки сервера центру обробки даних SDDC WEB1.acme.com буде index.htm. Час відгуку визначається системою балансування навантаження центру обробки даних SDDC як час із моменту відправлення запиту на надання інформації до моменту одержання коду повернення.

Однак при тім, що зовнішній моніторинг дає можливість одержати про сервер корисну інформацію, як метод діагностики він має свої недоліки. Про такі істотні характеристики сервера центру обробки даних SDDC, як стан центрального процесора, пам'яті, системної шини, шини уведення/виводу, мережний інтерфейсної плати, а також про ряд важливих ресурсів системи й прикладних програм адміністратор має лише уривчасті відомості або взагалі ніяких. Докладну інформацію про навантаження сервера центру обробки даних SDDC може надати тільки внутрішній моніторинг. Для його виконання в системі балансування навантаження центру обробки даних SDDC передбачені спеціальні агенти внутрішнього моніторингу, які встановлюються на кожному сервері. Агент постійно контролює стан свого "середовища перебування" і повідомляє про нього засобу балансування. Багато постачальників пропонують

інструментальні засоби для роботи зі сценаріями, які дозволяють адміністраторам створювати утиліти внутрішнього моніторингу для WEB-додатків. Внутрішній моніторинг широко застосовується в програмних системах балансування, але в апаратних пристроях і в рішеннях на базі комутаторів цей метод діагностики реалізується рідко.

Вибір сервера центру обробки даних SDDC

З урахуванням інформації, отриманої в ході зовнішнього й внутрішнього моніторингу серверів центру обробки даних SDDC, система балансування навантаження центру обробки даних SDDC може виділити сервер, що здатний краще інших упоратися з обробкою клієнтського запиту. Якщо робочі характеристики апаратних і програмних компонентів всіх серверів центру обробки даних SDDC пула рівнозначні, можна настроїти систему балансування навантаження центру обробки даних SDDC так, щоб вона виділяла сервер для обробки чергового запиту за принципом кругового списку, з огляду на при цьому стан сервера центру обробки даних SDDC. Могутнішому серверу буде діставатися більше число запитів. Інакше кажучи, мова йде про той же принцип кругового списку, тільки з урахуванням вагових коефіцієнтів.

Сучасні системи балансування навантаження центру обробки даних SDDC дозволяють адміністраторові визначати правила вибору сервера центру обробки даних SDDC за своїм розсудом. Можна, приміром, включити в ці правила такі критерії, як коефіцієнт завантаження ЦП і пам'яті, число відкритих з'єднань TCP і кількість пакетів, що надходять на мережну інтерфейсну плату того або іншого сервера центру обробки даних SDDC. Формула, по якій система балансування визначає рівень завантаженості серверів центру обробки даних SDDC, може виглядати приблизно так: (10 * рівень використання ЦП) + (3 * рівень використання пам'яті) + (6 * число відкритих TCP-з'єднань) + (3 * число переданих пакетів) = завантаження сервера центру обробки даних SDDC. При одержанні запиту від клієнта система балансування навантаження центру обробки даних SDDC розраховує по цій формулі навантаження кожного сервера

центру обробки даних SDDC й направляє запит серверу з найменшим навантаженням.

У деяких випадках після того, як засіб балансування призначає сервер для виконання запиту клієнта й клієнт установлює первісне з'єднання, для забезпечення виконання прикладної програми необхідно, щоб система балансування постійно підтримувала трафік між клієнтом і сервером. Таке з'єднання називається постійним. Припустимо, відвідувач електронної книгарні вибирає покупку й відкладає у віртуальний кошик три книги. Якщо сервер, що обробляє запит клієнта, кешує інформацію про вміст кошика локально, система балансування навантаження центру обробки даних SDDC вже не може виділити для наступних обмінів із клієнтом інший сервер, навіть якщо того вимагає співвідношення, що змінилося, навантажень серверів центру обробки даних SDDC вузла. Адже при такій переадресації дані про вміст віртуального кошика клієнта будуть загублені, тому що новий сервер попросту не має цих відомостей. Отже, система балансування повинна фіксувати інформацію про те, з яким сервером зв'язується кожний клієнт, і зберігати її в пам'яті протягом певного часу (час зберігання вказує адміністратор з урахуванням таких факторів, як звичайне поводження клієнтів і особливості прикладної програми). При активізації функції постійних (стійких) з'єднань вона буде увесь час скасовувати інші правила балансування навантаження центру обробки даних SDDC.

При організації стійкого з'єднання основне завдання системи полягає в тому, щоб ідентифікувати клієнта й зв'язати відповідний ідентифікатор із сервером-одержувачем запиту. Як правило, системи балансування навантаження центру обробки даних SDDC використовують як ідентифікатор клієнта, застосовувану їм IP-адресу відправника. Але вся справа в тому, що адреса відправника не обов'язково збігається з реальною IP-адресою клієнта. Багато компаній і провайдери, намагаючись удержати WEB-трафік під контролем і сховати від сторонніх очей IP-адреси своїх користувачів, установлюють сервери-посередники. На щастя, цю проблему можна вирішити, якщо скористатися

системою балансування навантаження центру обробки даних SDDC, оснащеної засобами ідентифікації IP-адрес відправників і номерів портів TCP. Подібні системи здатні пізнавати клієнтів навіть у тому випадку, коли останні виходять в Internet через той самий proxy-сервер. Така ідентифікація можлива тому, що кожне TCP-з'єднання має унікальна IP-адресу відправника й номер порту TCP. Ще один спосіб ідентифікації клієнта, що проводить захищений сеанс зв'язку за протоколом HTTP, полягає в тому, щоб зафіксувати ідентифікаційний номер сеансу зв'язку користувача за протоколом Secure Sockets Layer (SSL). Протокол SSL призначає кожному встановленому сеансу зв'язку спеціальний ідентифікатор, а прикладні програми для віртуальних магазинів часто користуються цим протоколом. Найсучасніший засіб підтримки стійких з'єднань – це розповсюджувані по мережі WEB cookie-файли. Нагадаю, що ці файли містять як відомості про клієнта, так і інші дані (наприклад, про те, з яким сервером клієнт зв'язувався востаннє). Аналіз умісту cookie-файлів допомагає системі балансування навантаження центру обробки даних SDDC ідентифікувати клієнтів і підбирати для них найбільш підходящий сервер. У число постачальників систем балансування навантаження центру обробки даних SDDC, оснащених засобами роботи з cookie-файлами, входять такі компанії, як Alteon WEBSystems, ArrowPoint Communications, F5 Networks і Resonate.

Нарешті, існує ще один спосіб вибору сервера центру обробки даних SDDC – так зване безпосереднє зв'язування (immediate binding). Відповідно до цього методу системи балансування навантаження центру обробки даних SDDC підбирають сервер для клієнта й направляють запит на нього в той самий момент, коли система одержує від клієнта пакет TCP SYN. При цьому система балансування вибирає сервер, керуючись заданими правилами розподілу навантаження серверів центру обробки даних SDDC, а також IP-адресою, що втримується в отриманому від клієнта пакеті TCP SYN. Цей метод забезпечує висока швидкодія, але треба сказати, що при його застосуванні система балансування просто не встигає проаналізувати іншу інформацію: зокрема,

ідентифікатор сеансу зв'язку за протоколом SSL, уміст cookie-файлу, адреса URL і дані прикладної програми. Щоб одержати більше докладні відомості про клієнта й, відповідно, точніше вибрати для нього сервер, системі балансування навантаження центру обробки даних SDDC потрібен час для аналізу інформації рівня додатка. При виборі сервера центру обробки даних SDDC по методу відкладеного зв'язування система балансування навантаження центру обробки даних SDDC ухвалює рішення щодо призначенні сервера центру обробки даних SDDC лише по завершенні трьохетапного обміну підтверджувальними повідомленнями й після установки з'єднання між нею й клієнтом. Система може враховувати вміст публікуемих матеріалів, якщо досліджує інформацію рівня прикладної програми до вибору сервера центру обробки даних SDDC для клієнта.

Переадресація трафіка

Системи балансування навантаження центру обробки даних SDDC можуть перенаправляти трафік клієнтів на вибраний сервер декількома способами: по методу трансляції адрес із керуванням доступом до середовища передачі (media access control (MAC) address translation, MAT), по методу трансляції мережних адрес (Network Address Translation, NAT), або – при використанні відкладеного зв'язування – за допомогою механізму шлюзу TCP (TCP gateway). Розглянемо, як реалізується кожний із цих методів перенапрямку трафіка засобами вирівнювання навантаження.

MAT. Цей метод може бути реалізований системою балансування навантаження центру обробки даних SDDC при тій умові, що кожний сервер поряд зі своїм фізичним IP-адресою використовує в якості інтерфейсного адресу зворотного зв'язка (loopback interface address) VIP-адреси системи балансування. Одержавши від клієнта пакет і призначивши йому відповідний сервер, система балансування заміняє в цьому пакеті MAC-адреса одержувача MAC-адресою відповідного сервера центру обробки даних SDDC, після чого направляє пакет на виділений сервер. У пакеті втримується IP-адреса клієнта, так що для прямої відповіді клієнтові сервер використовує в якості IP-адреси одержувача первісна

IP-адреса клієнта. Однак у якості IP-адреси відправника сервер указує VIP-адресу системи балансування навантаження центру обробки даних SDDC, як якби трафік надходив клієнтові саме від її. Таким чином, що випливає пакет від клієнта направляється не тому, що відповів, серверу, а системі балансування навантаження центру обробки даних SDDC.

NAT. При використанні цього методу система балансування направляє отриманий від клієнта пакет призначеному серверу лише після того, як виконає над пакетом кілька операцій: по-перше, вона заміщає в пакеті адресу одержувача (тобто власна VIP-адреси) IP-адресою призначеного сервера центру обробки даних SDDC, а по-друге, – міняє IP-адресу відправника на свій VIP-адресу. Даний метод дозволяє приховувати від клієнтів IP-адреси серверів центру обробки даних SDDC, так що останні можуть використовувати будь-які IP-адреси, у тому числі й частки. При цьому сервери не обов'язково повинні бути безпосередньо з'єднані із системою балансування (інакше кажучи, входити в той самий сегмент ЛОМ); досить, щоб вони могли встановлювати з'єднання по протоколах статичної або мережної маршрутизації.

Шлюз TCP. При установці безпосереднього (негайного) зв'язування системи балансування навантаження центру обробки даних SDDC можуть направляти трафік по методах MAT або NAT на рівнях 2 або 3. Але якщо мова йде про відкладене зв'язування, системи балансування повинні управляти трафіком на рівні TCP і на більше високих рівнях. Відкладене зв'язування припускає, що система балансування навантаження центру обробки даних SDDC й клієнт установлюють з'єднання за протоколом TCP так, щоб система могла одержати дані додатки ще до призначення сервера центру обробки даних SDDC. Потім засіб балансування встановлює TCP-з'єднання із призначеним сервером і передає йому клієнтський запит. Далі система балансування передає клієнтові відповідь сервера центру обробки даних SDDC, для чого знов-таки використовується TCP-з'єднання "система балансування – клієнт". Описана функція й називається шлюзом TCP. Фахівці компанії Resonate реалізують її в

системі балансування навантаження центру обробки даних SDDC за допомогою спеціального агента (цей агент установлюється на сервері, що забезпечує пряме TCP-з'єднання між клієнтом і сервером, що виступає в ролі засобу балансування). По термінології, запропонованою компанією-постачальником, дана реалізація йменується системою із транзитом TCP-з'єднань (TCP connection hop).

Вибір сайту й керування трафіком на глобальному рівні

У тих випадках, коли інформаційні ресурси розміщаються на декількох дзеркальних вузлах, системи балансування навантаження центру обробки даних SDDC (іменовані також глобальними системами балансування навантаження центру обробки даних SDDC) визначають підходящий для клієнта вузол за допомогою вже описаних механізмів вибору сервера центру обробки даних SDDC. Крім того, як критерій вибору сайту глобальна система балансування може використовувати такий показник, як відстань між сайтом і клієнтом (виражене в кількості транзитних ділянок і в тривалості мережної затримки). При визначенні найбільш підходящого сайту система балансування часто направляє трафік клієнта на відповідний вузол за допомогою інтелектуальної функції DNS.

Крім методу динамічного призначення клієнтові того або іншого вузла системи балансування навантаження центру обробки даних SDDC можуть використовувати для зв'язування конкретних клієнтів з конкретними сайтами метод статичного призначення (static mapping method).

Надлишкові системи балансування

Системи балансування навантаження центру обробки даних SDDC – не що інше, як засоби доступу до серверів центру обробки даних SDDC WEB, і тому вихід такого засобу з ладу може викликати повне припинення роботи вузла. Звідси вивід: при плануванні й реалізації інфраструктури з вирівнюванням навантажень важливо брати до уваги відказостійкість засобів балансування, а також вибирати рішення із широкою смугою пропущення, здатні забезпечити високу продуктивність системи в цілому. У розпорядженні адміністратора є дві схеми організації надлишкових систем балансування навантаження центру

обробки даних SDDC: перша припускає, що одна система балансування функціонує в активному режимі, а інша – перебуває в стані очікування (active-and-standby type); друга ж схема передбачає одночасне функціонування обох систем балансування (active-and-active type). Обидві схеми припускають наявність на одному вузлі двох екземплярів систем балансування навантаження центру обробки даних SDDC.

При використанні методу одна система активна, а інша перебуває в стані очікування, резервна система балансування постійно контролює стан головної системи. Коштує останньої вийти з ладу, як резервна система балансування приймає на себе функції головної (тобто починає управляти трафіком). Коли ж головна система відновляє роботу, резервна передає їй керування трафіком і знову переходить у режим очікування.

У схемі із двома активними системами балансування навантаження центру обробки даних SDDC, обидві системи обслуговують трафік і підстраховують одна одну. Цей метод повною мірою використовує ресурси засобів балансування й підвищує продуктивність вузла.

Компанії, які обслуговують інформаційні центри WEB і займаються електронною комерцією, – далеко не єдині організації, що застосовують системи балансування навантаження центру обробки даних SDDC для керування трафіком і підтримки порядку у своїх віртуальних господарствах. Багато компаній взяли на озброєння засоби балансування для того, щоб підвищити продуктивність і доступність своїх вузлів WEB. Системи вирівнювання навантаження забезпечують моніторинг навантажень і стану серверів центру обробки даних SDDC, правильний вибір з пула серверів центру обробки даних SDDC машини, здатної щонайкраще обробити запит клієнта, а також керування трафіком як усередині вузла, так і в глобальному масштабі. Завдяки цьому вони стають потужною зброєю в конкурентній боротьбі між компаніями, що відкрили свої представництва в кіберпросторі.

Кластерна технологія балансування навантаження центру обробки даних SDDC мережі

Кластерна технологія балансування навантаження центру обробки даних SDDC мережі (Network Load Balancing), підвищує масштабованість і відказостійкість критично важливих служб, що працюють по протоколах TCP/IP, таких як сервер, служби терміналів, служби віртуальних приватних мереж і потокового мультимедійного віщання. Служба балансування навантаження центру обробки даних SDDC функціонує на всіх вузлах кластера й не вимагає для роботи спеціального устаткування. Для забезпечення масштабованості продуктивності служба БНМ (Балансування навантаження центру обробки даних SDDC мережі, Network Load Balancing) розподіляє потік даних, переданих за протоколом IP, між декількома вузлами кластера. Крім цього, служба забезпечує високу відказостійкість, автоматично виявляючи збої вузлів і перерозподіляючи потік даних серед що залишилися. Служба БНМ включає функції віддаленого керування. Унікальна, повністю розподілена архітектура служби БНМ дозволяє забезпечувати найвищу продуктивність і захист від збоїв, особливо в порівнянні з технологіями розподілу навантаження на основі диспетчеризації.

Архітектура служби балансування навантаження центру обробки даних SDDC мережі

Для досягнення максимальної пропускної здатності й відказостійкості служба БНМ використовує повністю розподілену програмну архітектуру. На всіх вузлах кластера паралельно виконуються однакові драйвери служби БНМ. Ці драйвери поєднують всі вузли в єдину мережу для обробки вхідного потоку даних, що надходять на основну IP-адресу кластера (і на додаткові IP-адреси багатомережних вузлів). Для кожного окремого вузла драйвер виконує функції фільтра між драйвером мережного адаптера й стеком протоколів TCP/IP, дозволяючи розподіляти потік даних, одержуваних вузлом. Таким чином, що надходять запити розділяються й розподіляються між вузлами кластера.

Служба БНМ функціонує як мережний драйвер, розташований у мережній моделі нижче високорівневих протоколів додатків, таких як HTTP і FTP.

Така архітектура дозволяє домогтися максимальної пропускної здатності за рахунок використання широкомовної підмережі для доставки даних, що надходять, на всі вузли кластера, що дозволяє обійтися без маршрутизації вхідних пакетів. Оскільки фільтрація непотрібних пакетів працює швидше, ніж маршрутизація (при якій необхідно одержати, перевірити, перезаписати й повторно відправити кожний пакет), при використанні служби БНМ досягається більше висока пропускна здатність мережі в порівнянні з рішеннями на основі диспетчеризації. При росту швидкості роботи сервера центру обробки даних SDDC й мережі пропорційно росте й продуктивність; у такий спосіб усувається залежність від продуктивності апаратних рішень для розподілу навантаження на основі маршрутизації. Наприклад, у гігабітних мережах служба БНМ демонструє пропускну здатність до 250 Мб/с.

Іншою ключовою перевагою повністю розподіленої архітектури служби БНМ є чудові показники відказостійкості (N-1) для кластера з N вузлами. Навпроти, у рішеннях на основі диспетчеризації обов'язково є центральний елемент, що є «вузьким місцем» системи, для усунення якого необхідно використовувати резервний диспетчер, забезпечуючи лише односпрямоване переміщення навантаження при збої. Такий захист від збою менш ефективний у порівнянні з повністю розподіленою архітектурою.

В архітектурі служби БНМ для одночасної доставки даних, що надходять, на кожний вузол кластера використовується концентратор і/або комутатор підмережі. Проте, такий підхід веде до збільшення навантаження на комутатори й вимагає додаткових ресурсів пропускної здатності портів. Звичайно це не впливає на більшість широко використовуваних додатків (наприклад, WEB-служби й мультимедіа-мовлення), оскільки вхідні дані становлять дуже невелику частку загального потоку даних у мережі. Проте, якщо пропускна здатність лінії зв'язку до комутатора з боку клієнта значно вище пропускної здатності каналу з боку сервера центру обробки даних SDDC, що входять дані можуть становити досить значну частину загального потоку даних. Та ж проблема виникає й при підключенні декількох кластерів до одного комутатора, коли для окремих

кластерів не настроєні віртуальні локальні мережі LAN. Повністю конвеєрний механізм служби БНМ при надходженні пакетів одночасно передає їх у стек протоколів TCP/IP і одержує пакети від драйвера мережного адаптера. Це знижує загальний час обробки потоку даних і затримку, оскільки стек TCP/IP може обробляти пакет одночасно з одержанням драйвером NDIS наступного пакета. Крім того, потрібно менше ресурсів для координації операцій стека TCP/IP і драйвера, а також, у більшості випадків, у пам'яті не створюються додаткові копії пакетів даних. При відправленні пакетів служба БНМ також забезпечує підвищену пропускну здатність, малий час затримки й відсутність накладних витрат продуктивності за рахунок збільшення числа пакетів TCP/IP, які можуть бути відправлені за один виклик NDIS. Для досягнення настільки високої продуктивності служба БНМ використовує пул буферів і дескрипторів пакетів, використовуваний для конвеєрних операцій зі стеком TCP/IP і драйвером NDIS.

Розподіл потоку даних кластера

Для розподілу вхідного потоку даних між всіма вузлами кластера служба БНМ використовує широкомовні або багатоадресні пакети протоколу другого рівня. У використовуваному за замовчуванням одноадресному режимі служба БНМ перепризначує апаратну (MAC) адресу мережного адаптера, для якого вона включена (називаного адаптером кластера), і всім вузлам у кластері призначається одна MAC-адреса. Таким чином, пакети, що надходять, приймаються всіма вузлами кластера й передаються службі БНМ для фільтрації. Для забезпечення унікальності MAC-адреса формується на підставі основної IP-адреси кластера, зазначеної на сторінці властивостей служби БНМ. Служба БНМ автоматично змінює MAC-адресу адаптера кластера шляхом внесення зміни до реєстру й наступного перезавантаження драйвера адаптера; при цьому не потрібен перезапуск операційної системи.

Якщо вузли кластера підключені до комутатора, а не до концентратора, використання загальної MAC-адреси може викликати конфлікт, оскільки комутатори другого рівня здатні працювати тільки з унікальними MAC-адресами

джерел на всіх портах. Для усунення даної проблеми служба БНМ модифікує MAC-адресу джерела у вихідних пакетах. Таким чином, комутатору не передається фактичний MAC-адреса кластера, і в результаті вступні в кластер пакети даних доставляються на всі порти комутатора. Якщо вузли кластера підключені замість комутатора безпосередньо до концентратора, в одноадресному режимі служби БНМ маскування вихідної MAC-адреси можна відключити, щоб заблокувати лавинну маршрутизацію вихідних пакетів у комутаторі. Відключення маскування виробляється шляхом присвоєння в реєстрі параметру MaskSourceMAC служби БНМ значення 0. Використання комутатора вихідних даних за протоколом третього рівня дозволяє також уникнути лавинної маршрутизації.

В одноадресного режиму роботи служби БНМ є побічний ефект блокування безпосереднього зв'язка між мережними адаптерами вузлів кластера. Оскільки вихідні пакети, що відправляються іншому вузлу кластера, посилають на MAC-адресу, що збігається з адресою відправника, ці пакети вертаються відправникові мережним стеком і ніколи не потраплять у мережу. Це обмеження можна обійти, установивши в кожний сервер кластера другий мережний адаптер. У такому випадку служба БНМ обробляє дані тільки від мережного адаптера підмережі, від якого надходять клієнтські запити, а другий адаптер, як правило, підключається до окремої підмережі, призначеної для зв'язку з іншими вузлами й серверами файлів і баз даних. Для ритмічних повідомлень і віддаленого керування використовується тільки мережний адаптер кластера.

Слід зазначити, що обмеження одноадресного режиму служби БНМ не впливають на підключення вузлів кластера до вузлів, не включеним у кластер. Потік даних з мережі, що надходить на виділену IP-адресу (на адаптер кластера), одержують всі вузли кластера, оскільки вони використовують однакову MAC-адресу. Оскільки служба БНМ ніколи не передає розподілені дані на виділену IP-адресу кластера, вони доставляються безпосередньо стеку протоколів TCP/IP вузла-адресата. На інших вузлах служба БНМ обробляє ці пакети як уже

розподілений потік даних (оскільки IP-адреса одержувача не збігається з виділеними IP-адресами інших вузлів): вони можуть бути передані стеку TCP/IP, що їх відхилить. Передача надмірного обсягу даних на виділені IP-адреси при роботі в одноадресному режимі може привести до падіння продуктивності, оскільки стек TCP/IP змушений відхиляти велика кількість «чужих» пакетів.

Тому в службі БНМ є другий режим, що забезпечує розподіл вхідного потоку даних всім вузлам кластера. При використанні багатоадресного режиму замість зміни MAC-адреси адаптера кластера йому привласнюється широкомовна адреса другого рівня. Оскільки в кожного вузла зберігається унікальний MAC-адреса, у цьому режимі немає необхідності встановлювати другий мережний адаптер для зв'язку між вузлами кластера. У ньому також відсутнє падіння продуктивності при використанні виділених IP-адрес.

Для одночасної доставки вхідного потоку даних всім вузлам кластера в односпрямованому режимі роботи служби БНМ використовується лавинна маршрутизація (доставка пакетів в усі порти комутатора, крім вихідного). При використанні багатоадресного режиму комутатори також у багатьох випадках відправляють пакети в усі порти для доставки широкомовних даних. Проте, у багатоадресному режимі системний адміністратор має можливість обмежити лавинну маршрутизацію, настроївши в комутаторі віртуальну мережу для портів, що відносяться до вузлів кластера. Це може бути зроблене шляхом ручного настроювання комутатора або за допомогою протоколів IGMP (Internet Group Management Protocol, міжмережний протокол керування групами) і GARP. Поточна версія служби БНМ не має автоматичної підтримки протоколів IGMP і GMRP.

Вона підтримує протокол ARP (Address Resolution Protocol, протокол дозволу адрес), необхідний для дозволу основної IP-адреси й інших віртуальних IP-адрес у широкомовний MAC-адресу кластера. (Виділена IP-адрес по-колишньому дозволяється в MAC-адресі адаптера кластера.) Досвід показує, що маршрутизатори Cisco не приймають сигнал ARP від кластера, що

однонаправленні IP-адреси в широкомовні MAC-адреси. Цю проблему можна обійти, додавши в маршрутизатор статичний запис ARP для кожної віртуальної IP-адреси. Широкомовну MAC-адресу можна подивитися у вікні властивостей служби БНМ або за допомогою програми віддаленого керування Wlbs.exe. У використовуваному за замовчуванням односпрямованому режимі дана проблема відсутня, оскільки MAC-адреса кластера збігається з односпрямованою MAC-адресою.

Служба БНМ не управляє ніякими вхідними пакетами даних, крім даних по протоколах TCP, UDP і GRE (частина потоку даних PPTP) для зазначених портів. Вона не фільтрує дані по протоколах IGMP, ARP (за винятком зазначених вище), ICMP або іншим IP-протоколам. Всі дані по цих протоколах передаються стеку TCP/IP у незмінному виді на всіх вузлах кластера. У результаті для деяких TCP/IP-програм «точка-точка» (наприклад, ping) при використанні IP-адреси кластера можуть генеруватися відгуки, що дублюються. Завдяки надійності протоколу TCP/IP і його здатності обробляти дубльовані датаграми, інші протоколи продовжують правильно працювати в кластері. Щоб уникнути даної проблеми, для таких програм можна використовувати виділену IP-адресу.

Алгоритм балансування навантаження центру обробки даних SDDC

Для розподілу клієнтів, що підключаються, між вузлами кластера служба БНМ використовує повністю розподілений алгоритм фільтрації. Цей алгоритм був обраний для надання вузлам кластера можливості швидко й незалежно друг від друга ухвалювати рішення щодо розподілі навантаження для кожного вхідного пакета. Алгоритм був оптимізований для забезпечення статистично рівномірного розподілу навантаження по великій кількості клієнтських вузлів, що виконують постійні, відносно невеликі запити (такий тип навантаження типовий, наприклад, для серверів центру обробки даних SDDC). Коли кількість клієнтів невелика й/або обсяг запитів клієнтів міняється в широких межах, алгоритм розподілу навантаження служби БНМ менш ефективний. Проте, простота й швидкість роботи цього алгоритму дозволяють забезпечувати дуже високу

продуктивність, високу пропускну здатність і низький час відгуку для різних типів додатків клієнт/сервер.

Служба БНМ розподіляє вступні клієнтські запити, направляючи певний відсоток нових запитів кожному вузлу кластера. Відсоток розподіляється загрузки, що, задається у вікні «Властивості балансування навантаження центру обробки даних SDDC мережі» для кожного діапазону номерів портів, навантаження яких необхідно розподіляти. Алгоритм не здатний адаптуватися до змін навантаження кожного вузла кластера (на відміну від навантаження ЦП і використання пам'яті). Однак зіставлення міняється при зміні кількості вузлів кластера, відповідно до якого міняються відсотки розподілу навантаження.

При аналізі вступного пакета всі вузли одночасно виконують статичне зіставлення для швидкого вибору вузла, що буде обробляти пакет. Для зіставлення використовується функція випадкового вибору, що розраховує пріоритет вузла на підставі IP-адреси клієнта, номера порту й інших даних про стан, підтримуваних для оптимізації розподілу навантаження. Обраний вузол передає пакет з мережного стека в стек TCP/IP, а інші вузли відхиляють цей пакет. Схема зіставлення міняється тільки при змінах состава вузлів кластера; у результаті комбінація певної IP-адреси клієнта й номери порту завжди зіставлена одному вузлу кластера. Однак вузол кластера, якому зіставлена IP-адреса й порт клієнта, не можна визначити заздалегідь, тому що функція випадкового вибору враховує поточне й минуле членство в кластері для мінімізації змін схеми зіставлення.

В алгоритмі розподілу навантаження передбачається, що IP-адреси й номери портів клієнтів (коли не використовується режим прив'язки клієнтів) статистично не зв'язані. Таке припущення може перестати відповідати дійсності, якщо на стороні сервера центру обробки даних SDDC використовується брандмауер, що перетворить адреси клієнтів в одну IP-адресу, і одночасно включена прив'язка клієнтів. У такому випадку всі клієнтські запити будуть оброблятися одним вузлом кластера, а служба розподілу завантаження

функціонувати не буде. Проте, якщо прив'язка клієнтів не використовується, розкид номерів портів клієнтів навіть при використанні брандмауера звичайно здатний забезпечити гарний розподіл навантаження.

Як правило, якість розподілу статистично визначається кількістю клієнтів, що виконують запити. На практиці при включеному режимі прив'язки для рівномірного розподілу навантаження кількість клієнтів повинне значно перевищувати кількість вузлів.

Внаслідок стохастичного характеру набору клієнтів можуть спостерігатися незначні тимчасові зміни рівномірності розподілу навантаження. Слід зазначити, що досягнення абсолютно рівномірного розподілу навантаження між всіма вузлами кластера зажадало б значного збільшення продуктивності (пропускної здатності й часу відгуку), які витрачалися б на вимір і реакцію на коливання навантаження. Вартість ресурсів, які додатково потрібні були для цього, варто зіставити з потенційними перевагами максимально оптимального використання ресурсів вузлів кластера (в основному ресурсів ЦП і оперативної пам'яті). У кожному разі необхідно забезпечити надмірність ресурсів кластера для забезпечення обробки навантаження у випадку збоїти одного з вузлів. У службі БНМ використовується простий, але в теж час ефективний алгоритм розподілу навантаження, що забезпечує максимально можливу продуктивність і відказостійкість.

Параметри прив'язки клієнтів служби БНМ реалізовані за рахунок зміни вхідних даних алгоритму статичного зіставлення. При виборі прив'язки клієнтів у вікні «Властивості балансування навантаження центру обробки даних SDDC мережі» номера портів клієнтів не використовуються для зіставлення. Тому всі запити від одного клієнта завжди направляються на один вузол кластера. Необхідно відзначити, що для цього обмеження значення часу очікування не вказується (як це звичайно буває в рішеннях з диспетчеризацією), і це зіставлення буде діяти аж до зміни состава кластера. При виборі режиму прив'язки одного клієнта алгоритм зіставлення використовує повна IP-адресу клієнта. Однак при

виборі режиму прив'язки адрес класу С алгоритм використовує тільки частину адреси клієнта, що ставиться до класу С (перші 24 розряди). Таким чином, всі клієнти з одного адресного простору класу С зіставлені одному вузлу кластера.

При зіставленні клієнтів вузлам служба БНМ не може безпосередньо відслідковувати границі сеансів (таких, як сеанси за протоколом SSL), оскільки рішення про розподіл навантаження приймаються при встановленні підключень TCP до надходження пакетів, що містять дані мережних додатків. Крім того, служба не здатна відслідковувати границі потоків за протоколом UDP, оскільки границі логічних сеансів визначаються додатками. Замість цього параметри прив'язки служби БНМ використовуються для підтримки сеансів клієнтів. При збої або відключенні вузла від кластера всі підключення клієнтів до нього розриваються. Після визначення нового состава кластера за допомогою процедури сходження (див. нижче) клієнти, які були зіставлені вибулому вузлу, зіставляються одному з вузлів, що залишилися. При цьому збій не впливає на сеанси всіх інших клієнтів, яких кластер продовжує безперебійно обслуговувати. Таким чином, алгоритм розподілу навантаження мінімізує перерви в обслуговуванні клієнтів при збої.

При включенні в кластер нового вузла ініціюється процедура сходження для обліку нового члена кластера. Після завершення сходження новому вузлу зіставляється мінімум клієнтів. Служба БНМ відслідковує TCP-підключення до кожного вузла, і після завершення поточних підключень наступні підключення клієнтів, що зачіпаються, будуть оброблятися новим вузлом. Обробка UDP-потоків новим вузлом починається негайно. Це може привести до розриву деяких клієнтських сеансів, що включають кілька підключень або UDP-потоків. Отже, вузли необхідно додавати в кластер у моменти, коли це приведе до розриву мінімальної кількості сеансів. Щоб повністю усунути цю проблему, стан сеансів повинне управлятися серверними додатками, щоб мати можливість відновити або відновити сеанс із будь-якого вузла кластера. Наприклад, стан сеансу може передаватися серверу внутрішньої бази даних або зберігатися на стороні клієнта у

файлах cookie. Стан SSL-сеансів відновлюється автоматично шляхом повторної автентифікації клієнта.

GRE-потік усередині протоколу PPTP – це особливий випадок сеансу, на який не впливає додавання нового вузла в кластер. Оскільки GRE-потік обмежений у межах тривалості керуючого TCP-сеансу, служба БНМ відслідковує цей потік разом з керуючим сеансом. Таким чином, запобігає розрив тунельного підключення PPTP при додаванні нового вузла.

Сходження

Вузли кластера, що працює під управлінням служби БНМ, періодично обмінюються багатоадресними або широкомовними ритмічними повідомленнями. Це дозволяє контролювати стан кластера. При зміні стану кластера (збої, відключенні або включенні нового вузла) служба БНМ ініціює процес, називаний сходженням, при якому вузли обмінюються ритмічними повідомленнями для визначення нового вузла, поточний стан кластера й вибор вузла з найбільш високим пріоритетом у якості нового основного вузла. При досягненні вузлами угоди про поточний стан кластера зміни членства в кластері після сходження заносяться в журнал подій.

У процесі сходження вузли продовжують обробляти вхідний потік даних у звичайному режимі, за винятком запитів до вузла, що вийшов з ладу, які тимчасово не обслуговуються. Запити клієнтів до працюючих вузлів обробляються як звичайно. Процедура сходження закінчується, коли всі вузли повідомляють про однакове подання состава кластера протягом декількох циклів ритмічних повідомлень. При спробі включити в кластер вузол з неузгодженими правилами портів або зі значенням пріоритету, що вже використовується одним з вузлів, процедура сходження не буде завершено. Таким чином, запобігає передача частини потоку даних кластера неправильно настроєному вузлу.

Після завершення сходження потік даних клієнтів перерозподіляється між вузлами, що залишилися. Якщо вузол був доданий у кластер, процедура сходження дозволяє йому почати обробку своєї частини розподіленого

навантаження мережі. Розширення состава кластера не впливає на поточні операції й здійснюється повністю прозоро як для віддалених клієнтів, так і для серверних додатків. Однак воно може вплинути на підключення клієнтів, оскільки в процесі розширення між підключеннями клієнти можуть бути зіставлені іншим вузлам кластера.

В одноадресному режимі ритмічні повідомлення розсилаються кожним вузлом широкомовно, а в багатоадресному режимі – багатоадресно. Кожне ритмічне повідомлення займає один кадр Ethernet і позначено основною IP-адресою кластера, щоб забезпечити можливість роботи декількох кластерів усередині однієї підмережі. Ритмічним повідомленням призначається або значення, що набудовується, або шістнадцяткове значення 886F. За замовчуванням період відправлення ритмічних повідомлень дорівнює одній секунді. Це значення можна змінити за допомогою параметра реєстру AliveMsgPeriod. При виконанні процедури сходження для прискорення процесу період відправлення повідомлення знижується вдвічі. Навіть у великих кластерах смуга пропущення, займана ритмічними повідомленнями, украй невелика (наприклад, 24 КБ/с в 16-вузловому кластері).

При роботі служби БНМ передбачається, що вузол кластера функціонує правильно доти, поки він бере участь в обміні ритмічними повідомленнями між вузлами кластера. Якщо інші вузли не одержують від одного із серверів центру обробки даних SDDC ритмічні повідомлення протягом декількох періодів обміну повідомленнями, запускається процедура сходження. Кількість пропущених повідомлень, після якого починається сходження, за замовчуванням дорівнює п'яти, однак його можна змінити за допомогою параметра реєстру AliveMsgTolerance.

При одержанні ритмічного повідомлення від нового вузла або суперечливого повідомлення, що свідчить про проблеми в розподілі завантаження, процедура сходження запускається негайно. Після одержання ритмічного повідомлення від нового вузла кожний вузол перевіряє, чи не

обробляють інші вузли запити від тих же клієнтів. Така ситуація може виникати при об'єднанні підмережі кластера після поділу. Швидше за все, для нового вузла вже було виконане сходження у відділеному фрагменті й він не одержує даних від клієнтів. Така ситуація можлива, якщо комутатор вносить помітну затримку в підключення вузла до підмережі. Якщо вузол кластера визначає цю проблему, а інший вузол після останньої процедури сходження одержав більше число підключень клієнтів, він негайно припиняє обробку клієнтських даних для заданого діапазону портів. Оскільки обидва вузли обмінюються ритмічними повідомленнями, вузол, на який доводиться більша кількість підключень, продовжує обробляти запити, а інший вузол очікує завершення сходження, щоб почати обробку своєї частини навантаження. Такий евристичний алгоритм усуває можливі конфлікти розподілу завантаження при об'єднанні розділеного раніше кластера. Дана подія заноситься в журнал.

Віддалене керування

Механізм віддаленого керування службою БНМ використовує протокол UDP по порту 2504. Його датаграми відправляються на основну IP-адресу кластера. Оскільки датаграми обробляються драйверами служби балансування навантаження центру обробки даних SDDC на кожному вузлі кластера, вони повинні направлятися в підмережу кластера (а не в підмережу, до якої підключений кластер). Команди віддаленого керування, що віддаються в межах кластера, передаються в локальну підмережу кластера.

Продуктивність балансування навантаження центру обробки даних SDDC мережі

Вплив балансування навантаження центру обробки даних SDDC мережі на продуктивність відзначається в наступних чотирьох областях:

– Накладні витрати ресурсів ЦП на вузлах кластера, тобто відсоток обчислювальної потужності процесора, необхідний для аналізу й фільтрації мережних пакетів (переважно менше значення).

– Час відгуку на запит клієнта, що зростає пропорційно, що не перекривається частини, накладних витрат ресурсів ЦП, називається затримкою (переважно менше значення).

– Пропускна здатність до клієнтів, що збільшується при росту потоку даних від клієнтів, що кластер може обробити до досягнення максимального навантаження на вузли (переважно більше значення).

– Завантаження комутатора, що росте при збільшенні потоку даних від клієнтів (переважно менше значення). Завантаження не повинна критично впливати на пропускну здатність портів.

Крім того, масштабованість служби БНМ визначає ступінь росту продуктивності кластера при додаванні нових вузлів. Для масштабованості продуктивності необхідно, щоб накладні витрати ресурсів ЦП і час затримки збільшувалися повільніше, ніж число вузлів.

Алгоритм round-robin

Round-robin – алгоритм розподілу навантаження комп’ютерної мережі або розподіленої обчислювальної системи, методом перебору й упорядкування її елементів по круговому циклі.

Суть алгоритму

Нехай є N об'єктів, здатних виконати задану дію, і M завдань, які повинні бути виконані цими об'єктами. Мається на увазі, що об'єкти n рівні по своїх властивостях між собою, завдання m мають рівний пріоритет. Тоді перше завдання (m = 1) призначається для виконання першому об'єкту (n = 1), друге – другому й т.д., до досягнення останнього об'єкта (m = N). Тоді наступне завдання (m = N+1) буде призначене знову першому об'єкту й т.п. Простіше говорячи, відбувається перебір виконуючі завдання об'єктів по циклу, або по колу (round), і по досягненні останнього об'єкта наступне завдання буде також призначене першому об'єкту. Рішення завдань може бути додатково розбите на кванти часу, причому для продовження рішення в часі нумерація об'єктів (i, відповідно, призначені завдання) зрушується по колу на 1, тобто завдання першого об'єкта

віддається другому, другого – третьому, і т.д., а перший об'єкт одержує завдання останнього, або звільняється для прийому нового завдання. Таким чином, алгоритм round-robin стає алгоритмом розподілу часу або балансування навантаження центру обробки даних SDDC.

Застосування:

– балансування навантаження центру обробки даних SDDC обчислювальних мереж (Round robin DNS);

– керування завданнями в системах з розподілом часу (Round-robin scheduling).

Round robin DNS

Round robin DNS – один з методів розподілу навантаження, або відказостійкості за рахунок надмірності кількості серверів центру обробки даних SDDC, за допомогою керування відповідями DNS-сервера центру обробки даних SDDC відповідно до якоїсь статистичної моделі. Звичайно застосовується до таких Інтернет-протоколів, як сервери, FTP-сервери.

У найпростішому випадку round robin DNS працює, відповідаючи на запити не тільки однією IP-адресою, а списком з декількох адрес серверів центру обробки даних SDDC, що надають ідентичний сервіс. Порядок, у якому вертаються IP-адреси зі списку, заснований на алгоритмі round-robin. З кожною відповіддю, послідовність ip-адрес міняється. Як правило, прості клієнти намагаються встановлювати з'єднання з першою адресою зі списку, у такий спосіб різним клієнтам будуть видані адреси різних серверів центру обробки даних SDDC, що розподілить загальне навантаження між серверами.

Не існує стандартної процедури для визначення того, які адреси будуть використовуватися запитуючим додатком – деякі сервера центру обробки даних SDDC намагаються змінити порядок списку, приділяючи пріоритетну увагу чисельно більше «близьким» мережам. Деякі настільні клієнти намагаються одержати альтернативні адреси після того, як не вдалося встановити з'єднання протягом 30-45 секунд.

Кругова система DNS часто використовується для розподілу навантаження територіально-розподілених серверів центру обробки даних SDDC. Наприклад, у компанії є один домен і три ідентичних WEB-сайти, розташованих на трьох серверах із трьома різними адресами. Коли один користувач одержує доступ до головної сторінки, він буде спрямований на першу адресу IP. Другий користувач, що звертається до головної сторінки, буде відправлений на наступну адресу IP, а третій користувач буде відправлений на третю адресу IP. У кожному випадку, коли IP-адреса видається, він відправляється в кінець списку. Четвертий користувач, отже, буде відправлений знову на першу адресу IP, і так далі.

Хоча round robin DNS (RR DNS) легко реалізувати, все-таки цей алгоритм має кілька проблематичних недоліків, пов'язані з кешуванням запису в ієрархії RR DNS самого себе, а також з кешуванням на стороні клієнта, виданого адреси і його повторного використання, сполучення яких важко керовано. RR DNS не опирається на доступність послуг. Приміром, якщо сервіс на одній з адрес недоступний, RR DNS буде продовжувати роздавати цю адресу й клієнти будуть як і раніше намагатися добратися до непрацездатних послуг.

Крім того, воно не може бути кращим вибором для балансування навантаження центру обробки даних SDDC на самого себе, оскільки він лише заміняє порядок адрес щораз, коли ім'я сервера центру обробки даних SDDC запитується. Не існує обліку відповідності IP-адреси користувача і його географічного розташування, часу виконання, навантаження на сервер, перевантаження мережі й т.д. Кругова система DNS навантаження найкраще підходить для послуг з великою кількістю рівномірно розподілених з'єднань із серверами еквівалентної потужності. У противному випадку він просто робить розподіл навантаження.

Існують методи, щоб перебороти такі обмеження. Наприклад, модифіковані DNS-сервера центру обробки даних SDDC (такі, як lbnamed) можуть регулярно опитувати дзеркала серверів центру обробки даних SDDC для

перевірки їхньої доступності й навантаженності. Якщо сервер не відповідає в міру необхідності, сервер може бути тимчасово віддалений з пула DNS, поки він не повідомить, що знову працює у відповідності зі специфікацією.

Створений у результаті роботи над роботою, прототип центра, що засвідчує, – СКЗІ можна віднести до корпоративного рішення (варіант інсорсингу). Достоїнством цього програмно-апаратного комплексу є його масштабованість, що дозволяє включати розроблену систему як у жорстко задану ієрархічну структуру, так і будувати на його базі мережні моделі із кросс-сертифікацією.

Для рішення завдання вибору числа й розміщення елементів УЦ, представленого у вигляді інформаційно-обчислювального комплексу, запропоновано використовувати евристичні алгоритми, засновані на методах локальної оптимізації й теорії масового обслуговування. Альтернативний підхід до рішення завдання дискретного математичного програмування, заснований на теорії графів, неприйнятний через високу обчислювальну складність при великій кількості вузлів у мережі. Наприклад, у мережі з 10 вузлів існує 2^{45} варіантів розташування ліній зв'язку, включаючи безліч тривіальних випадків. Якщо припустити, що аналіз кожного варіанта становить 1 секунду, то на дослідження буде потрібно більш ніж $9*10^8$ років.

На змістовному рівні завдання побудови оптимальної мережі з УЦ формулюється в такий спосіб. Виходячи із заданих значень інтенсивності запитів абонентів з урахуванням вводимих допущень і обмежень визначити оптимальні за критерієм мінімуму наведених витрат структурні параметри мережі: число й розміщення програмно-апаратних модулів УЦ у пунктах мережі, співвіднести групи абонентів з обслуговуючими їх УК УЦ, ємність УК УЦ і каналів зв'язку. При цьому повинні дотримуватися обмеження на якість обслуговування: середній час обробки повідомлення й імовірність своєчасного обслуговування не повинні перевищувати граничних значень.

Як критерій оптимізації обрані наведені витрати на канал зв'язку й вузол мережі.

Як модель, що інтерпретує інформаційний потік у мережі з УЦ, розглядається дві мережі масового обслуговування (СеМО):

– багатофазна СеМО з відмовами й повторними викликами;

– багатофазна СеМО комбінованою комутацією.

Завдання пошуку оптимальної структури мережі успішно вирішуються з використанням евристичних алгоритмів, заснованих на методах локальної оптимізації зі спрямованим перебором варіантів структури мережі.

На першому етапі вся безліч абонентських пунктів розбивається на групи, розташовувані в окремих зонах обслуговування УК УЦ. У процесі роботи алгоритму число УК УЦ (відповідно й чисто зон) змінюється від мінімального (наприклад, рівного одиниці), до максимального. Для кожного значення шукається локальне оптимальне розміщення УК УЦ.

Абонентські місця приєднуються до найближчих УК УЦ.

Далі по черзі вибирається найкраще місце розташування кожного УК УЦ у межах своєї зони. Найкращий пункт для розміщення УК УЦ у межах своєї зони вибирається не із всіх можливих варіантів, а з пунктів підмножини A^*, число елементів якого істотно менше в порівнянні з безліччю A. Додатково прискорити процес перебору дозволяє уведення околиць, у які входять до десяти варіантів можливого розташування УК УЦ. Після УК УЦ $д$-й зони процес повторюється із УК УЦ першої зони. Якщо, починаючи з будь-якого УК УЦ, $д$ раз не вдалося поліпшити значення оптимизуємого функціонала, то процес вибору розміщення $д$ УК УЦ у фіксованих зонах припиняється, тому що досягається локальний екстремум.

При фіксованій кількості УК УЦ рішення завдання закінчується, коли перестановка УК УЦ не приводить до мінімізації оптимізуємого функціонала.

Після того як мережа із УК УЦ сформована (знайдене місце розташування всіх УК УЦ і зроблене прикріплення абонентів), обчислюється значення

параметрів оптимізації (середній час обслуговування T и ймовірність своєчасного обслуговування Q). Знайдені значення рівняються із граничними величинами (T_b, Q_L). Якщо обмеження не виконуються, то поточна структура виключається з розгляду.

Оптимальна структура мережі відповідає мінімальному значенню функціонала серед отриманих величин при різному числі УК УЦ.

У цьому розділі основним питанням є поліпшення якості функціонування мережі УЦ шляхом адаптивного балансування навантаження центру обробки даних SDDC між його керуючими компонентами.

Серед аналізованих підходів по керуванню навантаженням і динамічним керуванням потоками інформації (метод рельєфів, матричний і хвильовий алгоритми, ігровий і ймовірносно-ігровий методи) для імітаційного моделювання системи балансування навантаження центру обробки даних SDDC на УК УЦ обраний матричний метод.

Аналіз вироблявся по наступних ознаках:

– вид аналізованої інформації;

– тип реакції на вхідний потік;

– тип системи керування;

– критерій оптимізації;

– обсяг службової інформації, що пересилається;

– тимчасові витрати;

– складність реалізації.

Недоліком матричного методу для розглянутої прикладної області є критерій оптимізації у вигляді довжини шляхи комутації. Це можна усунути шляхом формування іншого критерію оптимізації, адаптованого під роботу мережі з УЦ.

Матричний метод балансування навантаження центру обробки даних SDDC орієнтований на алгоритми знаходження найкоротших шляхів. Даний метод дозволяє знайти довжини найкоротших шляхів між всіма вузлами мережі

одночасно, ґрунтуючись на застосуванні операцій над матрицями відстаней. У розробленій модифікації методу пропонується замінити параметр довжини зв'язку інтегральним критерієм, розрахованим раніше.

У цьому випадку структуру мережі можна представити у вигляді матриці адаптивного балансування навантаження центру обробки даних SDDC мережі, де показник оптимальності визначається як результат розподілу інтегрального критерію z-ого вузла (n_i) на показник доступності лінії зв'язку з h-ним вузлом.

Запропонована в роботі модифікація матричного підходу полягає в наступному:

– пропонується замінити параметр довжини зв'язку інтегральним критерієм;

– на кожному кроці побудови матриці балансування в додатковій матриці запам'ятовуються проміжні транзитні вузли. Це дозволяє по результатом багатокрокового перерозподілу навантаження видавати інструкції для УК УЦ по маршрутизації вступні на них повідомлень.

Запропоноване рішення по поліпшенню якості функціонування мережі з УЦ рівнялося з існуючими моделями якості обслуговування (Quality of Service). За результатами порівняння зроблений вивід про неможливість рішення завдання перерозподілу навантаження між УК УЦ винятково засобами цих моделей.

Результатом моделювання є вказівки для УК УЦ, сформовані за допомогою програмного модуля розробленої системи, засобами якого вузли можуть перенаправляти надлишкове навантаження на аналогічні компоненти. З наведених результатів моделювання можна зробити вивід, що розроблена модель динамічного балансування навантаження центру обробки даних SDDC мережі враховує основні характеристики роботи УЦ, дозволяючи оптимально перерозподілити вхідний інформаційних потік на її керуючих компонентах. У результаті балансування не тільки знімається надлишкове завантаження із УК УЦ, але й відбувається перерозподіл вхідного потоку з недоступних вузлів.

Матриця доступності, використовувана при моделюванні, дозволяє врахувати залежність від якості лінії зв'язку, що актуально, як при роботі через Інтернет-провайдерів у мережі загального користування, так і у випадку мережі корпоративного УЦ, що діє у ЛОМ або розподіленої VPN-мережі. Модифікація матричного підходу знаходження оптимальних для прийняття надлишкового навантаження вузлів і введення інтегрального показника, дозволяють сформувати нотації для перемикання навантаження не тільки на вигідний УК УЦ, але й організувати маршрут перерозподілу, що складає з декількох транзитних вузлів.

При побудові моделі (без апріорної прив'язки до «організації») дало можливість зв'язати її блоки на різних рівнях декомпозиції з об'єктами організаційно-технічної структури, що виступають як механізми. У цьому випадку, і це методично надто важливо, організаційно-технічна структура стає результатом функціонального моделювання. Саме цього вдалося досягти при декомпозиції механізмів впливу на систему керування навантаженням на вузли УЦ.

Створена функціональна модель мережі з УЦ дозволяє вирішити наступні завдання:

– сформувати наочне візуальне подання взаємодії основних процесів, що відбуваються в мережі з УЦ;

– зробити чітке визначення ролі системи поліпшення функціонування в загальній системі УЦ;

– точно визначити вхідні, вихідні й керуючі впливи на підсистему балансування навантаження центру обробки даних SDDC, що надалі дозволяє провести аналітичне й імітаційне моделювання інформаційного потоку;

– сценарій балансування навантаження центру обробки даних SDDC, розроблений у ході виконання роботи, є основою алгоритму імітаційного моделювання.

Функціональна схема розробленої системи зображена на рисунку 3.2.

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

– Блок розрахунку вектору значень інтегрального критерію оптимізації.

– Блок ініціалізації граничних значень параметрів моделювання.

– Блок формування матриці балансування навантаження центру обробки даних SDDC.

– Блок формування матриці транзитних вузлів мережі центру обробки даних SDDC.

– Блок формування сигналів для елементів мережі центру обробки даних SDDC.

– Згенеровані звіти по поточному завантаженню елементів мережі центру обробки даних SDDC.

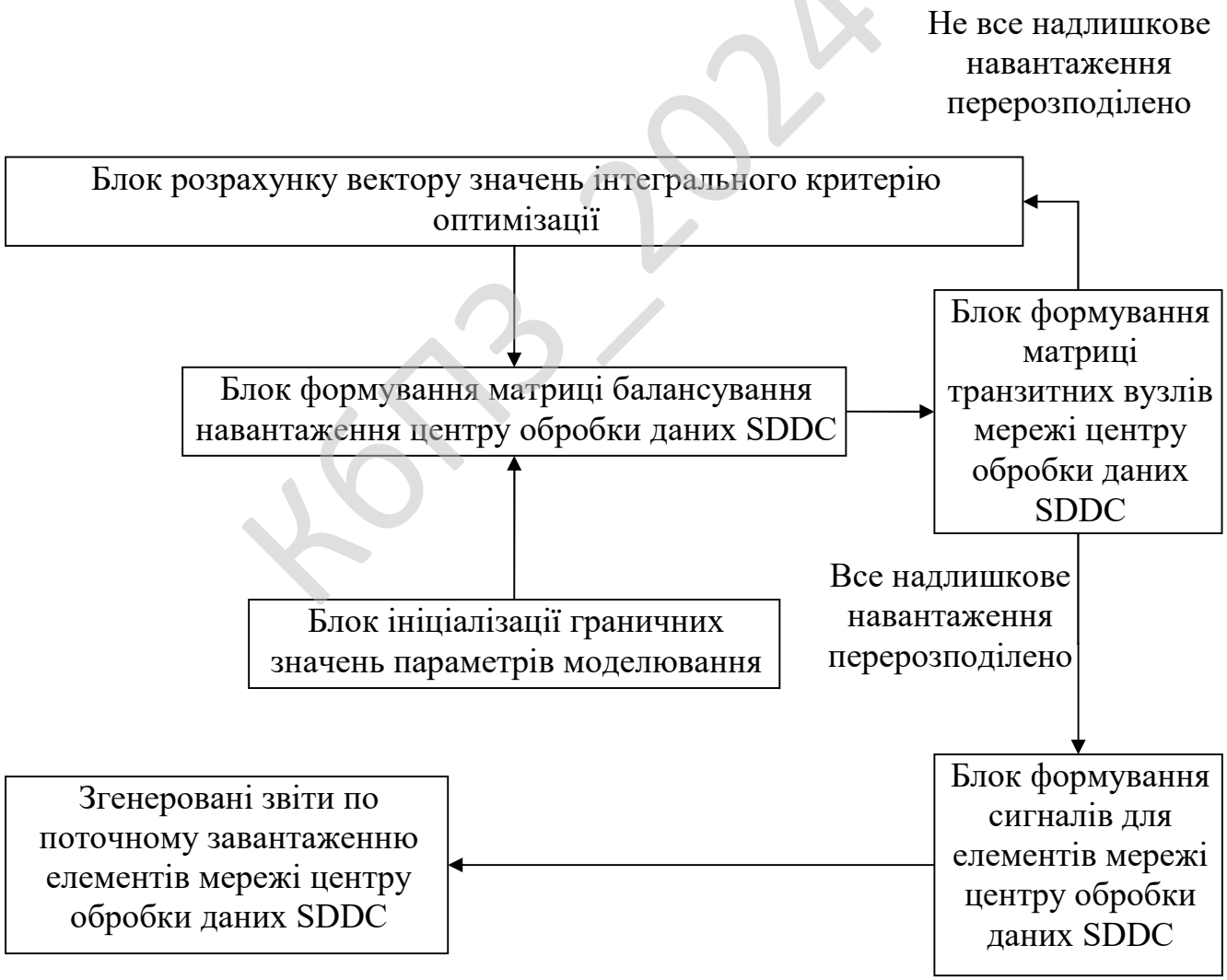

Рисунок 3.2 – Функціональна схема системи

Інтернет-технології прийняли широке поширення, і використовуються як основа побудови корпоративних і критично важливих додатків, таких як WEB-вузли, вузли потокового мультимедіа-віщання й сервери віртуальних приватних мереж.

Служба розподілу навантаження мережі центру обробки даних SDDC є оптимальним і ефективним рішенням, що забезпечує масштабованість і високу відказостійкість таких додатків як в Інтернеті, так і в інтрамережах.

Служба БНМ дозволяє системним адміністраторам створювати кластери, що включають до 32 вузлів мережі центру обробки даних SDDC, між якими будуть розподілятися вступні від клієнтів запити.

При цьому з погляду клієнтів кластер нічим не відрізняється від звичайного сервера центру обробки даних SDDC; серверні додатки також не вимагають адаптації для роботі в кластері.

Служба БНМ постачена всіма необхідним адміністраторам способами керування, у тому числі можливістю (після уведення пароля) віддалено управляти кластером з будь-якого комп'ютера в мережі центру обробки даних SDDC. Крім того, адміністратори мають можливість набудовувати кластер під спеціальні завдання, управляючи потоком даних на рівні портів. Вузли додаються й виключаються із кластера без припинення обслуговування. Крім того, програмне забезпечення на вузлах кластера можна обновляти без припинення обробки клієнтських запитів.

Служба БНМ використовує для розподілу навантаження між вузлами повністю розподілений алгоритм.

На відміну від рішень на основі диспетчеризації така архітектура забезпечує високу продуктивність і низькі накладні витрати ресурсів на розподіл потоку запитів від клієнтів.

Крім того, для цієї архітектури характерна висока відказостійкість (N-1) при числі вузлів мережі центру обробки даних SDDC N. Всі ці характеристики

досягаються без необхідності використовувати спеціальні апаратні або програмні рішення.

Служба БНМ періодично розсилає ритмічні повідомлення, призначені для інформування кожного члена кластера про наявність інших вузлів мережі центру обробки даних SDDC.

Збій будь-якого вузла виявляється протягом п'яти секунд, а відновлення обслуговування клієнтів виконується протягом десяти секунд. Як при відключенні працюючого вузла, так і при додаванні нового вузла в кластер навантаження автоматично й прозоро перерозподіляється між членами кластера.

Тести продуктивності демонструють, що використання програмної служби БНМ дає низькі накладні витрати на обробку потоку даних і чудові можливості масштабування продуктивності, обмежені тільки пропускною здатністю підмережі центру обробки даних SDDC.

Служба БНМ демонструє пропускну здатність більше 200 Мбіт/с у реальних рішеннях по обслуговуванню електронної торгівлі з більш ніж 800 млн. запитів протягом дня. Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

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. Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

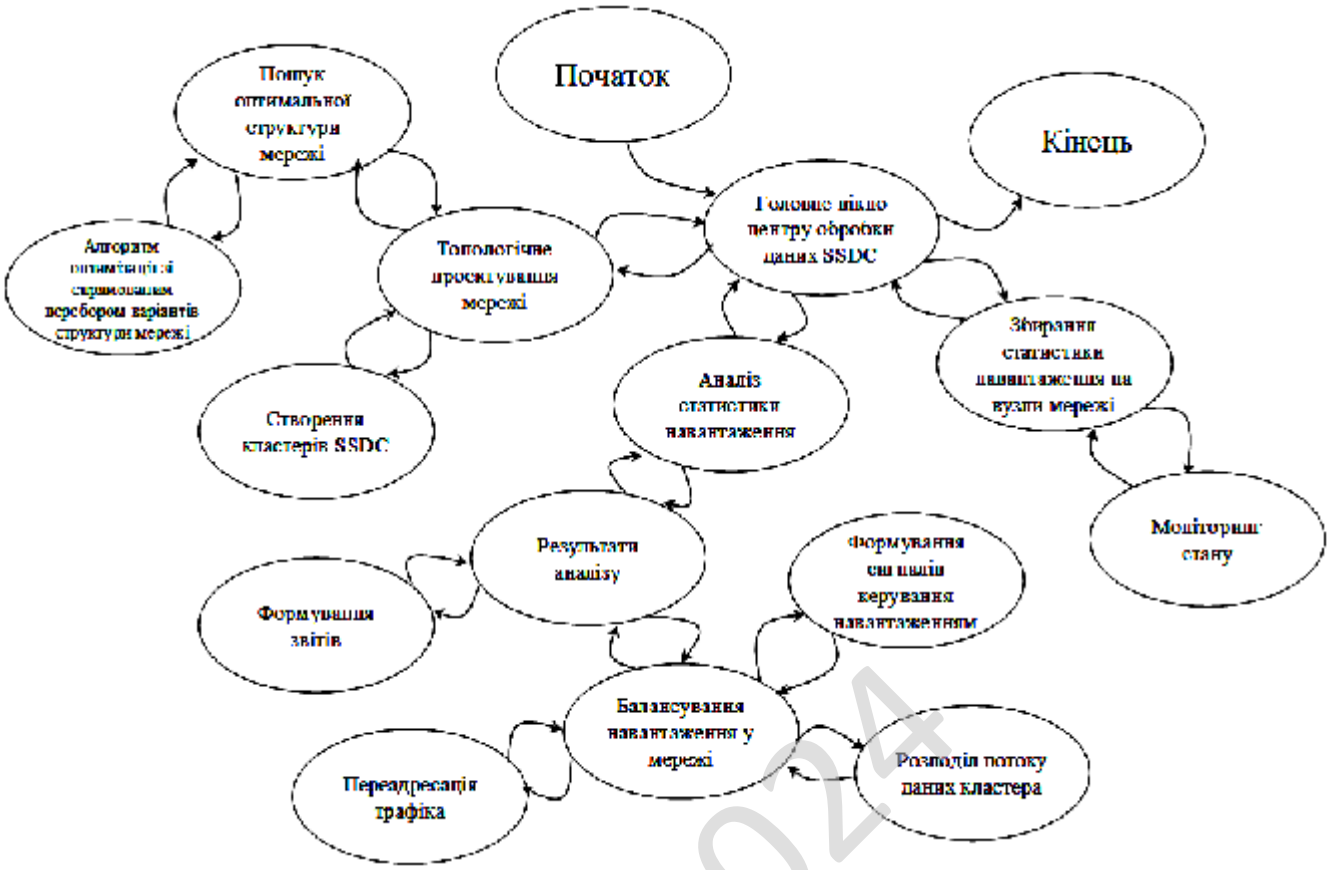

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграми потоків даних містять чотири типи елементів:

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

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схеми є основою ПЗ. Тому від точності і детальності проробки блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації, також те, що при розробці програми слід надати особливу увагу модулю центру обробки даних SSDC.

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

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

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірки поточного стану та поверненням на початок схеми чи з завершенням роботи розробленого ПЗ.

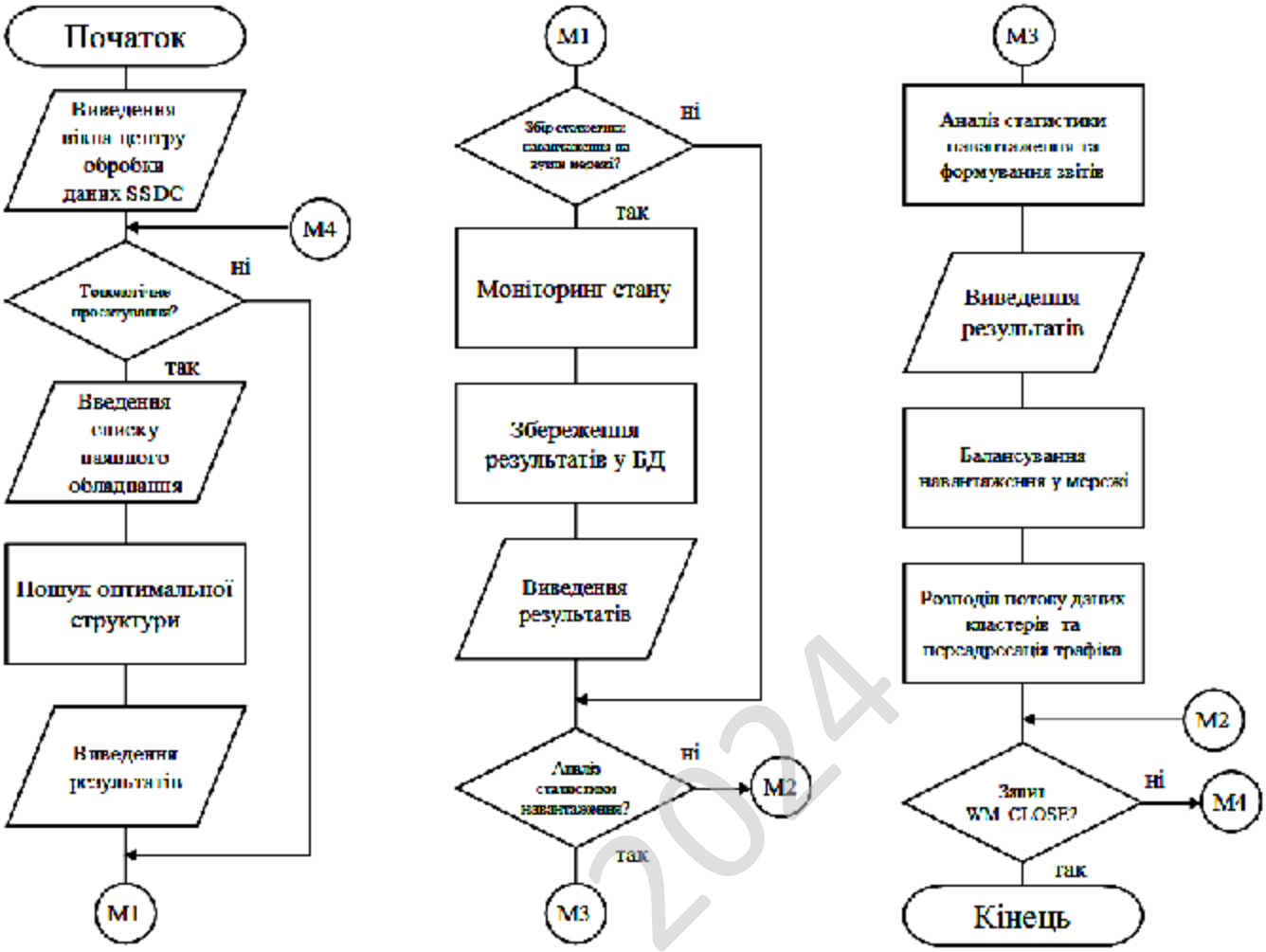

Рисунок 4.1 – Блок-схема основної програми

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.

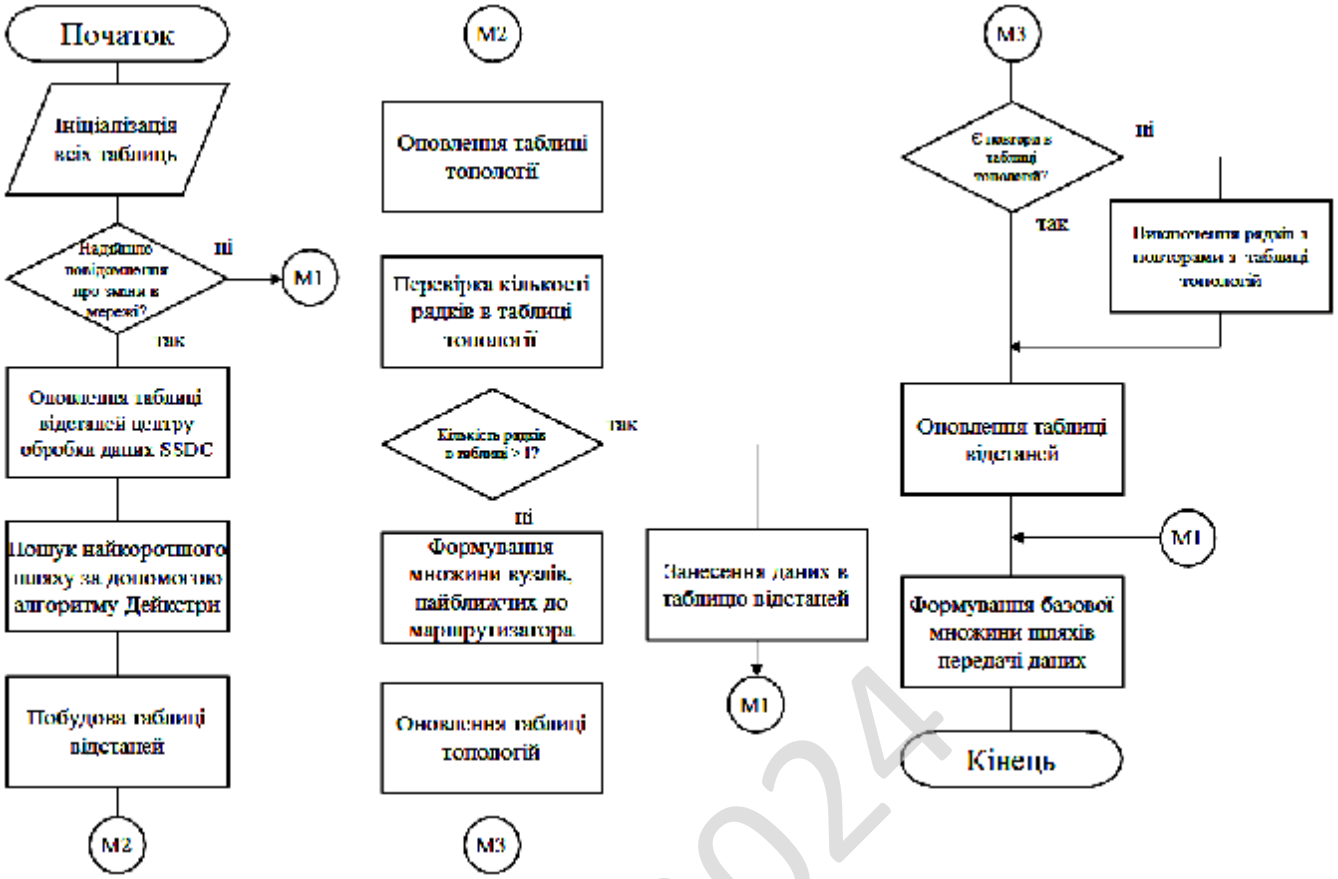

Рисунок 4.2 – Блок-схема роботи підпрограми

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.

Розглянемо використані технології та їх основні компоненти що підтверджують правильність використаних проектних рішень.

Незважаючи на те що я працював над ПЗ один в реалізації програми я використовував підходи пришвидшення розробки на основі методологій Agile – Extreme Programming.

Екстремальне програмування (Extreme Programming, далі XP) це методологія розробки програмного забезпечення, найпопулярніша серед так

званих гнучких методологій. Має на меті поліпшення якості програмного забезпечення та чутливість до змін у вимогах замовників.

Як вид гнучких методологій, XP радить часті "випуски" програми у коротких циклах розробки, що має на меті поліпшити продуктивність праці та покращити можливості виконання вимог замовника що змінюються. Авторами даної методології є Кент Бек, Ворд Каннінгем, Мартін Фаулер та інші.

Інші елементи екстремального програмування включають в собі: парне програмування, проведення обширної перевірки сирцевого коду, модульне тестування всього коду, уникання створення функціональності до того як вона дійсно необхідна, простота та ясність коду, очікування на зміну вимог замовників з плином часу та коли вимоги до продукту стають ясніші, досить часте спілкування із замовником та між самими програмістами.

Назва методології походить від ідеї застосувати корисні методи і практики розробки програмного забезпечення, піднявши їх до "екстремальних" рівнів.

Критики XP зауважують на потенційні недоліки цієї методології – нестабільні вимоги, незадокументовані компроміси конфліктів користувачів, відсутність загального документу дизайну програми.

Технологія екстремального програмування була розроблена Кентом Беком, Уардом Каннінґхемом та Роном Джеффріесом під час роботи над Chrysler Comprehensive Compensation System (C3). У 1996 Кент Бек став лідером проекту і почав вдосконалювати методи розробки, що застосовувалися в роботі над проектом. Свій метод він виклав у книзі «Extreme Programming Explained», котру було видано у жовтні 1999. Після купівлі Крайслера компанією Даймлер–Бенц проект C3 було скасовано у лютому 2000.

Хоча саме екстремальне програмування є відносно новим, багато її практик вже існували і використовувались протягом певного часу; однак, методологія підносить "найкращі практики" до екстремального рівня. Для прикладу, практика по плануванню і написанню тестів перед написанням кожної маленької частини коду було використано раніше в проекті НАСА "Меркурій".

Для зменшення часу на розробку ПЗ деякі формальні документи тестування (такі як приймальне тестування) писались паралельно (або й раніше) з написанням самого ПЗ. Незалежна група тестування НАСА може писати процедури тестування базуючись на формальних вимогах до продукту до того як програмне забезпечення розроблене та інтегроване в систему. В XP ця концепція піднесена до "екстремального рівня" завдяки написанню автоматичних тестів які перевіряють поведінку навіть малих частинок коду, а не тільки значних функціональних частин ПЗ.

Посібник Extreme Programming Explained: Embrace Change описує Екстремальне Програмування, як:

– Спроба примирити гуманність і продуктивність.

– Механізм для соціальної зміни.

– Шлях до удосконалення.

– Стиль розвитку.

Дисципліна розробки програмного забезпечення.

Головною метою Екстремального Програмування є скорочення вартості неочікуваних змін. У традиційних методах розробки (на кшталт SSADM) вимоги до розвитку системи визначаються на початку роботи над проектом, і часто виправляються пізніше. Це означає, що вартість проекту через зміни буде більшою за заплановану (традиційна особливість для програмного забезпечення, що проектується).

XP використовується для скорочення вартості змін, завдяки представленню простих значень, принципів і методів. При використанні екстремального програмування, проект повинен стати гнучкішим щодо змін.

Extreme Programming Explained описує екстремальне програмування як дисципліну розробки програмного забезпечення яка змушує людей створювати високоякісне ПЗ якомога швидше.

XP намагається зменшити ціну зміни вимог до ПЗ завдяки малим циклам розробки, а не одним довгим циклом. Екстремальне програмування сприймає

зміни до вимог як звичайні, неминучі та бажані аспекти розробки ПЗ, і ці зміни мають бути очікуваним. Основна ідея полягає в тому що неможливо розробити самодостатній пакет вимог до ПЗ, зміни в вимогах – неминучі.

Екстремальне програмування також вводить набір практик та принципів на основі методології гнучкої розробки програмного забезпечення.

Екстремальне програмування описує чотири базові активності що виконуються при розробці програмного забезпечення: написання коду, тестування, слухання та дизайн.

Написання коду. Прихильники XP заявляють що єдиним дійсно важливим результатом розробки ПЗ є код: без готового коду нема продукту.

Тестування. Методологія екстремального програмування заявляє, що якщо дрібне тестування може перевірити незначну частину функціональності, то багато дрібних тестів можуть перевірити набагато більше частинок і продукт в цілому.

Основні прийоми XP. Дванадцять основних прийомів екстремального програмування (за першим виданням книги Extreme programming explained) можуть бути об'єднані в чотири групи:

1. Короткий цикл зворотного зв'язку (Fine scale feedback).

1.1. Розробка через тестування (Test driven development).

1.2 Гра в планування (Planning game).

1.3. Замовник завжди поруч (Whole team, Onsite customer).

1.4 Парне програмування (Pair programming).

2. Безперервний, а не пакетний процес.

2.1 Безперервна інтеграція (Continuous Integration).

2.2 Рефакторинг (Design Improvement, Refactor).

2.3 Часті невеликі релізи (Small Releases).

3. Розуміння, що поділяється всіма учасниками.

3.1 Простота (Simple design).

3.2 Метафора системи (System metaphor).

3.3 Колективне володіння кодом (Collective code ownership) або обраними шаблонами проектування (Collective patterns ownership).

3.4 Стандарт кодування (Coding standard or Coding conventions).

4. Соціальна захищеність програміста (Programmer welfare), а саме 40 годинний робочий тиждень (Sustainable pace, Forty hour week).

```
# Основна система центру обробки даних SSDC
# Архітектура побудована на модульному підході

import random
import math
from datetime import datetime

# Модуль для управління обладнанням серверного центру
class Server:
    def __init__(self, server_id, cpu_capacity, ram_capacity,
ssd_capacity):
        self.server_id = server_id
        self.cpu_capacity = cpu_capacity
        self.ram_capacity = ram_capacity
        self.ssd_capacity = ssd_capacity
        self.current_load = 0

    def add_load(self, load):
        # Додаємо навантаження на сервер
        if self.current_load + load <= self.cpu_capacity:
            self.current_load += load
        else:
            raise ValueError("Перевищено обчислювальну потужність сервера")

    def clear_load(self):
# Очищаємо сервер від навантаження
        self.current_load = 0

# Модуль для моніторингу навантаження
class LoadBalancer:
    def __init__(self):
        self.servers = []

    def add_server(self, server):
```

```
# Додаємо сервер у систему
        self.servers.append(server)

    def distribute_load(self, load):
        # Розподіляємо навантаження по серверах
        for server in self.servers:
            if server.current_load + load <= server.cpu_capacity:
                server.add_load(load)
return f"Навантаження {load} додано до сервера {server.server_id}"
        return "Немає достатньо ресурсів для навантаження"

# Модуль для управління енергоспоживанням
class PowerManagement:
    def __init__(self, base_power, server_count):
        self.base_power = base_power
        self.server_count = server_count

    def calculate_power(self, active_servers):
        # Розраховуємо споживання енергії
        power_usage = self.base_power * active_servers
        return power_usage

# Основна система центру
class SSDCSystem:
    def __init__(self):
        self.servers = []
        self.load_balancer = LoadBalancer()
        self.power_management = None
        self.data_processed = 0

    def setup_servers(self, num_servers):
        # Ініціалізуємо сервери
        for i in range(num_servers):
            server = Server(
                server_id=f"S{i + 1}",
                cpu_capacity=random.randint(50, 100),
                ram_capacity=random.randint(128, 256),
                ssd_capacity=random.randint(1024, 2048)
            )
            self.servers.append(server)
            self.load_balancer.add_server(server)
self.power_management = PowerManagement(base_power=500,
```

```
server_count=num_servers)

            def process_data(self, data_load):
                # Обробляємо дані
                load_per_task = math.ceil(data_load / len(self.servers))
                for i in range(len(self.servers)):
                    self.load_balancer.distribute_load(load_per_task)
                self.data_processed += data_load

            def monitor_energy(self):
                # Моніторимо споживання енергії
                active_servers = sum(1 for server in self.servers if
server.current_load > 0)
                return self.power_management.calculate_power(active_servers)

        # Головна функція для запуску системи
        def main():
            # Ініціалізуємо систему SSDC
            ssdc = SSDCSystem()
            num_servers = 5
            ssdc.setup_servers(num_servers)

            # Симуляція обробки даних
            data_loads = [200, 400, 300]
            for load in data_loads:
                ssdc.process_data(load)
                print(f"Енергоспоживання після обробки {load} ГБ:
{ssdc.monitor_energy()} Вт")
            # Загальний звіт
            print(f"Загальна кількість оброблених даних: {ssdc.data_processed} ГБ")
```


Подійно-орієнтована архітектура (Event-driven architecture, надалі EDA) – шаблон архітектури програмного забезпечення, який призначений для створення подій, їх виявлення, споживання і реагування на них.

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

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

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

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

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

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

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

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

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

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

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

Подія може складатися з двох частин: заголовка події та тіла події. Заголовок події може включати в себе інформацію таку як, наприклад, назва події, часова мітка події і тип події. Тіло події – це частина, яка описує факт, що стався в дійсності. Тіло події не слід плутати з шаблоном або логікою, яка може бути застосована як реакція на саму події.

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

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

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

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

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

Механізм обробки подій. Механізм обробки подій (event processing engine) є місцем, де подія ідентифікується і вибирається відповідна реакція на нього, яка потім виконується. Це також може призвести до породження ряду тверджень. Якщо подія, яка надійшла до механізму обробки подій, є наприклад такою «Запаси продукту ID досягли нижнього допустимого рівня», це може ініціювати, наприклад, такі реакції як «Замовити продукт ID» і «Сповістити персонал».

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

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

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

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

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

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

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

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення захистимо за допомогою національного стандарту захисту інформації на основі алгоритму шифрування/дешифрування ДСТУ 4145-2002 з використанням еліптичних кривих над двійковим розширеним полем Галуа. У системі шифрування/дешифрування як параметри розглядається еліптична крива E_p (a,b) і точка G на ній. Учасник В вибирає закритий ключ n і обчислює відкритий ключ $P_B = n \times G$. Щоб зашифрувати повідомлення P_m використовується відкритий ключ

одержувача В P_B. Учасник А вибирає випадкове ціле позитивне число k і обчислює зашифроване повідомлення C_m, що є точкою на еліптичній кривій.

$$C_m = \{k \times G, P_m + k \times P_B\}. \quad (4.1)$$

Щоб дешифрувати повідомлення, учасник В множить першу координату точки на свій закритий ключ і віднімає результат від другої координати:

$$P_m + k \times P_B - n_B \times (k \times G) = P_m + k \times (n_B \times G) - n_B \times (k \times G) = P_m. \quad (4.2)$$

Учасник А зашифрував повідомлення P_m додаванням до нього $k \times P_B$. Ніхто не знає значення k, тому, хоча P_B і є відкритим ключем, ніхто не знає $k \times P_B$. Супротивнику для відновлення повідомлення доведеться обчислити k. Зробити це буде нелегко. Одержувач також не знає k, але йому як підказку посилається $k \times G$. Помноживши $k \times G$ на свій закритий ключ, одержувач одержить значення, що було додано відправником до незашифрованого повідомлення. Тим самим одержувач, не знаючи k, але маючи свій закритий ключ, може відновити незашифроване повідомлення.

Нехай задано просте число $p > 4$. Тоді еліптичною кривою E, визначеною над розширеним двійковим полем полем F_{2^m}, називається безліч пар чисел $(x, y), x, y \in F$, що задовольняють тотожності:

$$y^2 \equiv x^3 + a \cdot x + b (\mathrm{mod}\, 2^m), \quad (4.3)$$

де $4 \cdot a^3 + 27 \cdot b^2$ не рівно з нулю по модулю 2^m.

Інваріантом еліптичної кривої називається величина $J(E)$, що задовольняє тотожності:

$$J(E) \equiv 1728 \frac{4a^3}{4a^3 + 27b^2} (\mathrm{mod}\, 2^m). \quad (4.4)$$

Коефіцієнти a, b еліптичної кривої E, по відомому інваріанту $J(E)$, визначаються таким чином:

$$\begin{cases} a \equiv 3k (\mathrm{mod}\, 2^m) \\ b \equiv 2k (\mathrm{mod}\, 2^m) \end{cases} \text{де } k \equiv \frac{J(E)}{1728 - J(E)} (\mathrm{mod}\, 2^m), J(E) \neq 0 \text{ або } 1728. \quad (4.5)$$

Пари (x,y), що задовольняють тотожності (4.1), називаються точками еліптичної кривої E, x та y – відповідно x– та y-координатами точки.

Точки еліптичної кривої позначатимемо $Q(x,y)$ або просто Q. Дві точки еліптичної кривої рівні, якщо рівні їх відповідні x– і y-координати.

На безлічі всіх точок еліптичною кривою Е введемо операцію додавання, яку позначатимемо знаком "+". Для двох довільних точок $Q_1(x_1, y_1)$ та $Q_2(x_2, y_2)$ еліптичної кривої Е, розглянемо декілька варіантів.

Нехай координати точок Q_1 та Q_2 задовольняють умові $x_1 \neq x_2$. В цьому випадку їх сумою називатимемо точку $Q_3(x_3, y_3)$ координати якої визначаються порівняннями:

$$\begin{cases} x_3 \equiv \lambda^2 - x_1 - x_2 (\mathrm{mod}\, 2^m), \\ y_3 \equiv \lambda(x_1 - x_3) - y_1 (\mathrm{mod}\, 2^m), \end{cases} \text{де } \lambda \equiv \frac{y_2 - y_1}{x_2 - x_1} (\mathrm{mod}\, 2^m). \quad (4.6)$$

Якщо виконана рівність $x_1 = x_2$ та $y_1 = y_2 \neq 0$, то визначимо координати точки Q_3 таким чином:

$$\begin{cases} x_3 \equiv \lambda^2 - 2x_1 (\mathrm{mod}\, 2^m), \\ y_3 \equiv \lambda(x_1 - x_3) - y_1 (\mathrm{mod}\, 2^m), \end{cases} \text{де } \lambda \equiv \frac{3x_1^2 + a}{2y_1} (\mathrm{mod}\, 2^m). \quad (4.7)$$

У разі, коли виконана умова $x_1 = x_2$ та $y_1 = -y_2 (\mathrm{mod}\, p)$, суму точок Q_1та Q_2 називатимемо нульовою точкою О, не визначаючи її х– і у-координати. В цьому випадку, точка Q_2 називається запереченням точки Q_1. Для нульової точки О виконана рівність:

$$Q + 0 = 0 + Q = Q, \quad (4.8)$$

де Q – довільна точка еліптичної кривої Е.

Щодо введеної операції складання безліч всіх точок еліптичною кривою Е, разом з нульовою точкою, утворюють кінцеву абельову (комутативну) групу порядку т, для якого виконана нерівність:

$$p + 1 - 2\sqrt{p} \leq m \leq p + 1 + 2\sqrt{p} \quad (4.9)$$

Точка Q називається точкою кратності k, або просто – кратною точкою еліптичної кривої E, якщо для деякої точки P виконана рівність:

$$Q = P + ... + P = kP \quad (4.10)$$

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ центру обробки даних SSDC яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Файл; Кластер; Хост; Параметри; Довідка.

– Розділу виведення поточних даних.

– Розділу виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

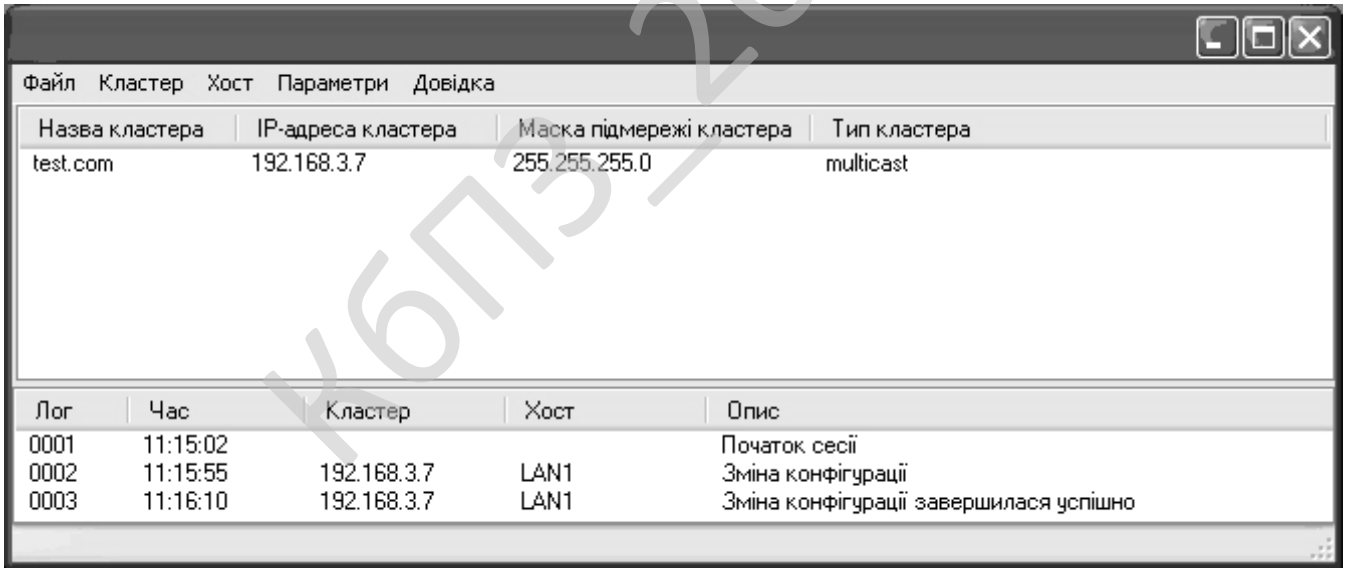

Рисунок 5.1 – Головне вікно ПЗ

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

середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий.

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

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

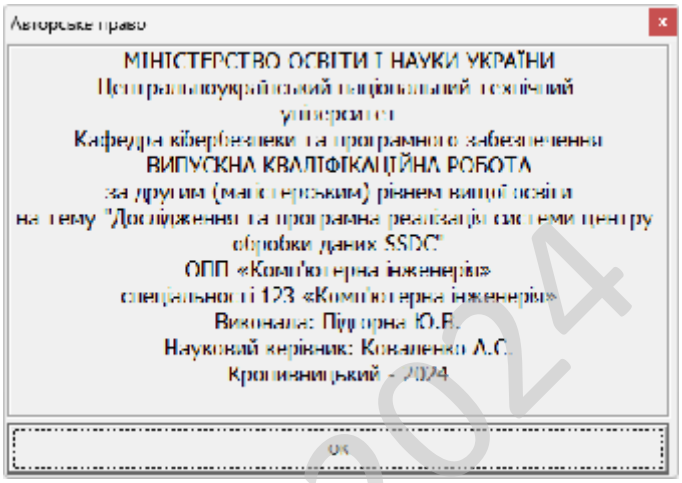

Рисунок 5.2 – Авторське право

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 чорної скриньки.

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

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

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

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

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

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

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

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

Будь-який спосіб тестування «чорної скриньки» повинен:

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

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

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

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій.

– Помилок інтерфейсу.

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

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

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – 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.

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

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи центру обробки даних SSDC.

Метою розробки є дослідження та програмна реалізація системи центру обробки даних SSDC.

Об'єктом дослідження є процес центру обробки даних SSDC.

Предметом дослідження є методи центру обробки даних SSDC.

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

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

– Удосконалено метод центру обробки даних SSDC.

– Розроблено вітчизняний продукт центру обробки даних SSDC, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмної реалізації системи центру обробки даних (ЦОД) з технологією SSDC (Software-Defined Data Center) можуть зацікавити кілька груп професіоналів та організацій (рис. 7.1).

ІТ-компанії – для них важливою є можливість гнучкого масштабування, автоматизації процесів, зниження операційних витрат і підвищення надійності ІТ-інфраструктури, яку забезпечують SSDC. Вони можуть зацікавитися впровадженням таких рішень для оптимізації роботи ЦОД.

Інтегратори систем і постачальники хмарних рішень – ці компанії можуть впроваджувати SSDC як частину своїх сервісів або рішень для клієнтів, що потребують програмно визначених ресурсів.

Підприємства, які потребують масштабованої ІТ-інфраструктури – для великих підприємств, що працюють з великими обсягами даних, важлива стабільність та ефективне управління інфраструктурою. SSDC дозволяє автоматизувати та оптимізувати їхні процеси.

Фінансові та страхові установи – завдяки SSDC вони можуть підвищити безпеку даних та зменшити час на управління ІТ-інфраструктурою, що є критичним для швидкої обробки транзакцій та захисту чутливих даних.

Наукові та дослідницькі інститути – для них результати досліджень у галузі SSDC можуть бути корисними для розвитку нових рішень у високопродуктивних обчисленнях, обробці великих даних та машинному навчанні.

Виробники обладнання та програмного забезпечення – вони можуть використовувати результати дослідження для розробки нових продуктів або покращення існуючих, що відповідатимуть вимогам сучасного ринку.

Організації, що займаються автоматизацією та віртуалізацією – їм може бути цікавим використання SSDC для управління віртуалізованими середовищами та впровадження гнучких рішень на основі програмного забезпечення.

Рисунок 7.1 – Цільова аудиторія

Результати дослідження можуть сприяти зниженню витрат, підвищенню ефективності та створенню нових можливостей для цих груп, що стимулюватиме подальший розвиток SSDC.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Для оцінки привабливості програмної реалізації системи центру обробки даних SSDC можна застосувати метод експертних оцінок, наприклад, метод вагових коефіцієнтів із використанням експертної групи.

Спочатку визначимо ключові критерії, за якими оцінюватиметься привабливість проєкту (рис. 7.2).

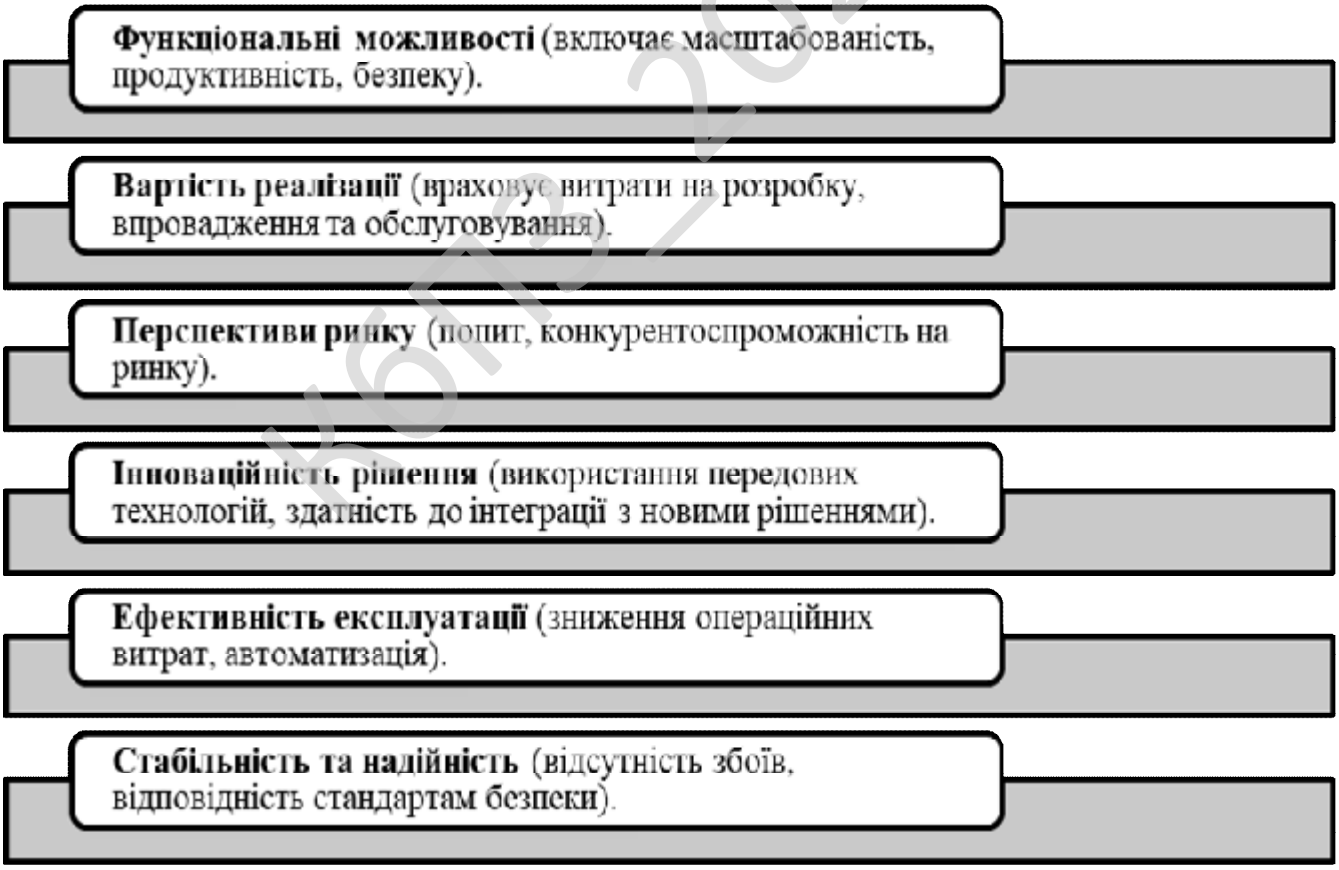

Рисунок 7.2 – Критерії оцінювання

До експертної групи можна залучити фахівців у сфері IT-інфраструктури, фінансових спеціалістів, інженерів із системної інтеграції та спеціалістів із розвитку продуктів.

Експерти оцінюють вагу кожного критерію за шкалою від 0 до 1, де сума ваг дорівнює 1: функціональні можливості – 0.25, вартість реалізації – 0.15, перспективи ринку – 0.20, інноваційність рішення – 0.15, ефективність експлуатації – 0.15, стабільність та надійність – 0.10.

Експерти виставляють оцінки від 1 до 10 для кожного критерію, де 1 – мінімальна привабливість, 10 – максимальна. Далі виводимо середні бали: функціональні можливості – 8, вартість реалізації – 7, перспективи ринку – 9, інноваційність рішення – 8, ефективність експлуатації – 7, стабільність та надійність – 9.

Підсумкова оцінка привабливості розраховується як сума добутків оцінки за критерієм на ваговий коефіцієнт цього критерію:

Оцінка привабливості=(0.25×8)+(0.15×7)+(0.20×9)+(0.15×8)+(0.15×7)+(0.10×9) = 8.0

Після обчислення отримаємо підсумкову оцінку привабливості. Чим вищий результат, тим привабливішим є проєкт для реалізації. Якщо отримане значення є високим (у нашому випадку 8 балів), це свідчить про доцільність інвестування та розробки SSDC з огляду на високу оцінку функціональності, ринкових перспектив і надійності.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації системи центру обробки даних SSDC можна розглянути такий метод як Constructive Cost Model.

Суть методу: COCOMO II – це модель розрахунку вартості, що враховує розмір програмного коду та фактори складності (технологічні, кадрові, інфраструктурні). У рамках COCOMO II використовуються різні рівні деталізації.

Переваги: детальний метод, що враховує численні фактори, як-от досвід команди, складність технологій і середовище проєкту.

Особливості для SSDC: модель дозволяє отримати точнішу оцінку, враховуючи технологічні складності, такі як інтеграція з віртуальними машинами та автоматизоване управління ресурсами.

Рекомендація: підходить для великих проєктів SSDC, де потрібна висока точність.

Це дозволить отримати комплексну оцінку, враховуючи як програмну складність, так і вимоги до функціональних можливостей системи SSDC.

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

Впровадження системи центру обробки даних SSDC може дати значний економічний ефект завдяки оптимізації процесів, зниженню витрат на обслуговування та підвищенню продуктивності (рисунок 7.3).

Загальний економічний від впровадження SSDC гарантуватиме: зниження витрат на обладнання, економія на енергоспоживанні, скорочення операційних витрат, прискорений вихід на ринок, зменшення втрат від простоїв та свідчить ймовірну про високу ефективність від впровадження SSDC.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Алгоритм просування проєкту програмної реалізації системи центру обробки даних SSDC включає кілька ключових етапів, спрямованих на привернення уваги цільової аудиторії, розбудову інтересу, стимулювання попиту та утримання клієнтів (рис. 7.4).

Рисунок 7.3 – Економічна ефективність від реалізації проєкту для клієнта

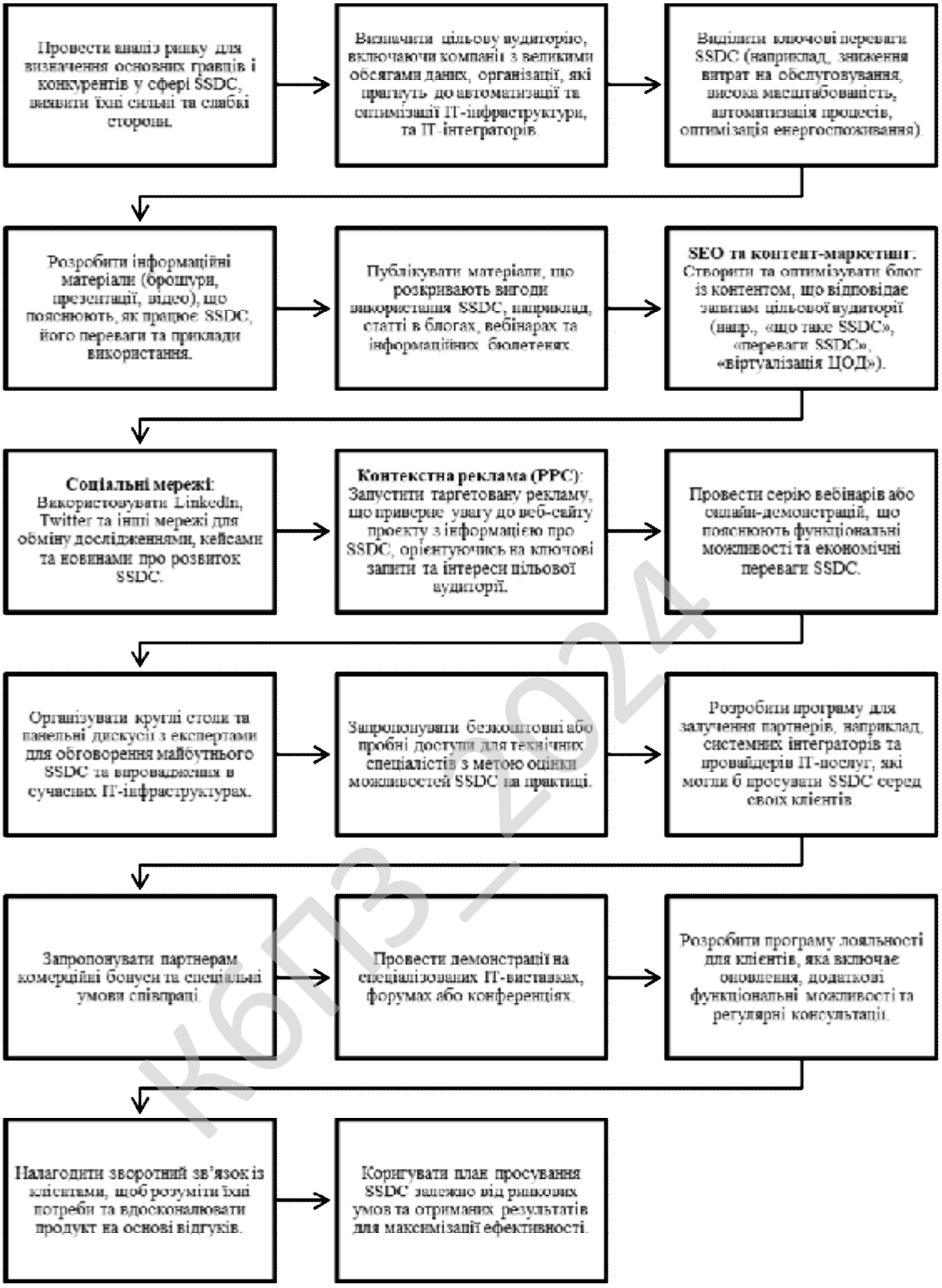

Рисунок 7.4 – Алгоритм просування проєкту

Цей алгоритм дозволить планомірно розвивати проєкт SSDC, залучати нових клієнтів та підвищувати впізнаваність і попит на продукт на ринку.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту та шляхів реалізації проєкту програмної реалізації системи центру обробки даних SSDC можна розглянути кілька стратегічних підходів, спрямованих на залучення широкого кола клієнтів, підвищення ефективності каналів продажу та спрощення процесу реалізації (рис. 7.5).

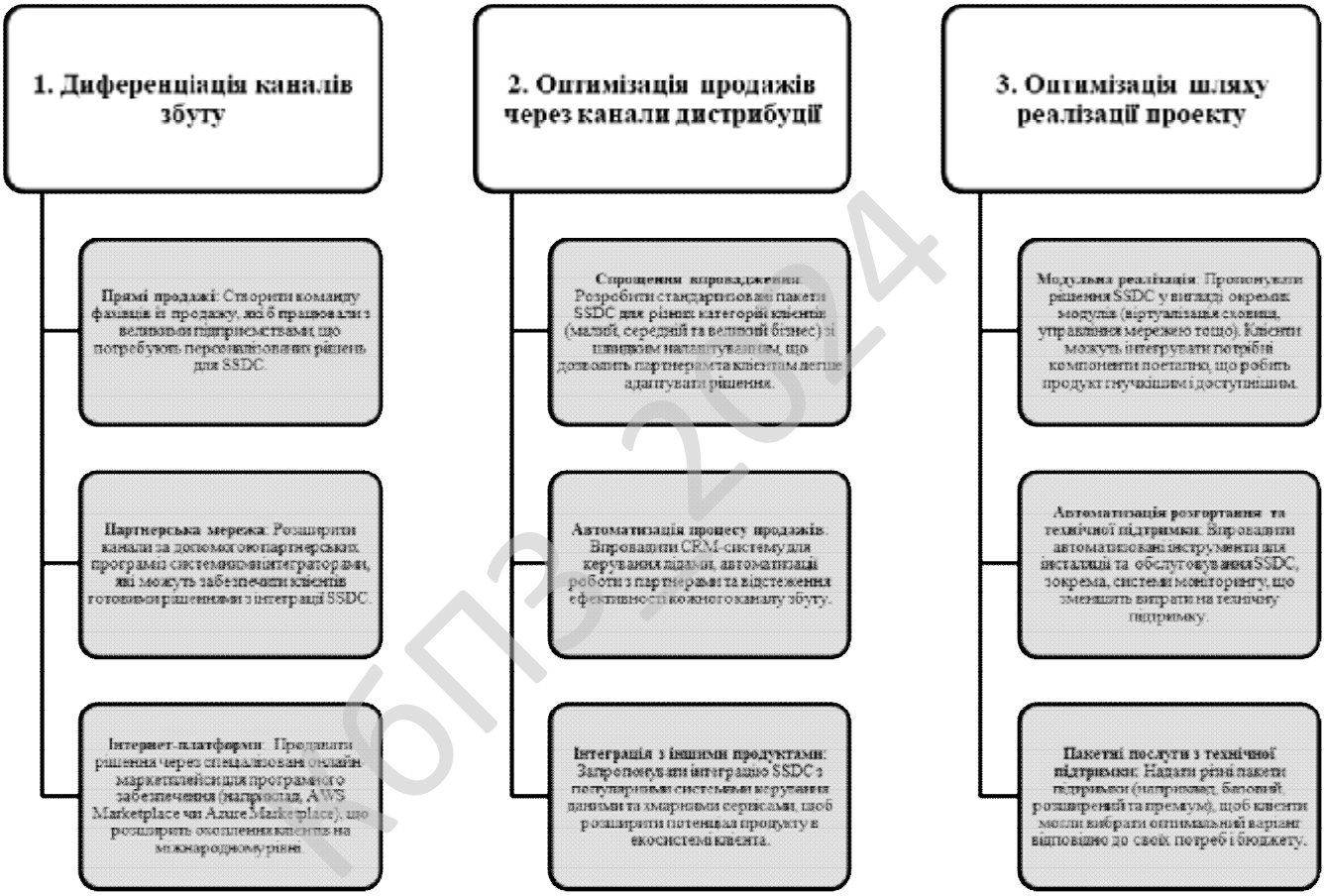

Рисунок 7.5 – Оптимізація каналів збуту

Оптимізація каналів збуту та шляхів реалізації допоможе SSDC ефективніше виходити на ринок, підвищити залученість клієнтів та зменшити витрати на впровадження й обслуговування.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовими факторами успіху проєкту програмної реалізації системи центру обробки даних SSDC є ті аспекти, які забезпечують ефективність, надійність і конкурентоздатність рішення (рис. 7.6).

Рисунок 7.6 – Ключові фактори успіху проєкту

Ці ключові фактори дозволяють SSDC-системі відповідати високим вимогам ринку, забезпечувати економічну та операційну ефективність для користувачів і підвищувати привабливість для потенційних клієнтів.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Характерною ознакою сучасного науково-технічного прогресу практично у всіх сферах діяльності людини є широке застосування комп'ютерних технологій, заснованих на використанні електронно-обчислювальних машин (ЕОМ). Сьогодення, а тим більше, майбутнє, вже важко уявити без комп'ютерів та іншої електронної техніки. Адже саме завдяки їм стала можливою швидка обробка величезних обсягів інформації, виконання необхідних розрахунків та інших видів робіт, пов'язаних з обробкою текстових даних та ілюстраційних зображень, організація оперативного отримання та передачі інформації, збереження її значних обсягів електронним способом.

Стрімке впровадження комп'ютерів не тільки в сфері управління виробництвом, в банківській систему, бізнесі, системі освіти, але також на транспорті, сфері обслуговування призвело до того, що десятки мільйонів людей у всьому світі виявились втягнутими у взаємодію людини з комп'ютером. Природно виникає запитання: настільки безпечною є ця взаємодія для людини? Адже відома аксіома про те, що будь-яка взаємодія людини та засобів праці двостороння. Людина впливає на удосконалення засобів праці, а останні – на працюючу людину. Отже, навіть сучасні технології та техніка, до яких безперечно, залежать комп'ютерні технології та ЕОМ несуть у собі певні потенційні небезпеки. У зв'язку з цим набуває актуальності адекватна оцінка конкретних умов і характеру праці, яка сприяє обґрунтованому розробленню та впровадженню комплексу заходів і засобів, спрямованих на збереження здоров’я і працездатності людини в процесі праці за рахунок поліпшення параметрів виробничого середовища, зменшення важкості, напруженості трудового процесу та збереження здоров'я працівників на комп'ютеризованих робочих місцях.

Законодавством України чітко врегульовано норми та вимоги до використання комп’ютерної техніки на підприємстві, безпосередньо й охорона праці на підприємстві при роботі за комп’ютером., зокрема «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», затверджені наказом Мінсоцполітики від 14.02.2018 № 207 [1], «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98. [2].

Загальні вимоги пожежної безпеки під час експлуатації комп’ютерної техніки визначають «Правила пожежної безпеки в Україні» (затверджені наказом МВС від 30.12.2014 № 1417) [3], комп’ютерних класів — пункт 3 розділу VIII «Правил пожежної безпеки для навчальних закладів та установ системи освіти України» (затверджені наказом МОН від 15.08.2016 № 974). [4] та інші державні стандарти, що регламентують експлуатування комп’ютерної техніки як радіоелектронної апаратури.

8.2 Шкідливі і небезпечні фактори при роботі з комп’ютером

Можна виділити наступні основні фактори, що впливають на стан здоров'я людей, які працюють за комп'ютером:

– сидяче положення на протязі тривалого періоду;

– вплив електромагнітного випромінювання монітора;

– втома очей, навантаження на зір;

– перевантаження суглобів кистей;

– стрес при втраті інформації або виникненні критичних помилкою.

У кожному з цих випадків ступінь ризику прямо пропорційний часу, що проводиться за комп’ютером і поблизу від нього. В сучасних умовах взаємодія людини з технікою значно ускладнилась, що вимагає комплексного підходу, який передбачає розгляд людини, технічних засобів праці та виробничого середовища, як взаємозв'язаних елементів єдиної системи. Все вищесказане в повній мірі

відноситься й до системи «людина–комп'ютер–середовище». Вагомий вплив на працездатність та здоров'я користувачів комп'ютерів здійснює виробниче середовище. Це середовище у виробничих приміщеннях (офісах), в основному, визначається мікрокліматом, освітленням, наявністю шкідливих речовин у повітрі, рівнем шуму та випромінювання.

Для того, щоб об'єктивно проаналізувати відповідність умов праці діючим нормативно-правовим актам та запропонувати заходи щодо зменшення негативного впливу комп'ютера на організм людини необхідно скласти санітарно-гігієнічну характеристику умов працівника, який працює з програмним продуктом.

8.3 Аналіз санітарно-гігієнічних умов праці на робочому місці користувача ПК

Розглянемо приміщення в якому працює користувач ПК з даним програмним продуктом. Приміщення має одностороннє природне освітлення і загальне штучне освітлення. Стіни і стеля обклеєні світлими шпалерами, підлога вкрита темним ламінатом. У приміщенні відсутні сильні вібрації та шкідливі речовини. Склад повітря відповідає нормі. У кімнаті знаходиться ПК з 4-ядерним процесором і 23-дюймовим IPS-монітором, а також меблі. Приміщення має довжину 4м, ширину 3,5м, висоту стелі 2,7м. Кількість робочих місць – одне. Площа – 14 $м^2$, об'єм – 37,8 $м^3$. Виходячи з цього, отримано дані, наведені в таблиці 8.1.

Таблиця 8.1 – Фактичні та нормативні значення параметрів приміщення

Параметр	Норма *	Реальні параметри
Площа, S	не менше 6 $м^2$	14 $м^2$
Об'єм, V	не менше 20 $м^3$	37,8 $м^3$

*Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин).

За даними, які наведено у табл. 8.1, можна зробити висновок, що отримані показники, площа та об’єм приміщення у розрахунку на одно робоче місце користувача ПК відповідає чинним нормам і вимогам.

Щодо мікроклімату, то згідно з ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» [5] роботу з ПК можна віднести до категорії легка - 1а. Джерелами тепла в цьому приміщенні є люди, електроустаткування, освітлювальні прилади в темний час доби і система опалювання взимку. Оператором виділяється до 120 ккал теплової енергії за годину. Оптимальні та фактичні значення параметрів мікроклімату приведені в таблиці 8.2.

Таблиця 8.2 – Значення параметрів мікроклімату

Період року	Параметр	Оптимальний*	Фактичний
Теплий	Температура	23 – 25 ^{0} С	24^0 С
	Вологість	40 – 60%	50%
	Швидкість повітря	< 0,1м/с	
Холодний	Температура	22 - 24^0 С	23^0 С
	Вологість	40 – 60%	55%
	Швидкість повітря	< 0,1м/с	

*ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень»

По отриманим замірам параметрів мікроклімату можна зробити висновок, що всі показники задовольняють вимогам, зазначеним для робіт категорії легка 1а і є задовільними для здоров'я людини.

Щодо освітлення , то згідно з ДБН В.2.5-28:2006 «Природне і штучне освітлення» [6] ця робота відноситься до Va розряду зорових робіт. Передбачається використання природного, штучного і змішаного освітлення.

Природне освітлення здійснюється за допомогою вікна, площа якого складає S' = 1,8х1,5 = 2,7 м^2 і є бічним освітленням. У світильниках місцевого і загального освітлення використовуються світлодіодні лампи потужністю 20 Вт із світловим потоком однієї лампи 900 лм. Згідно замірів рівень освітлення в даному приміщенні і на робочому місці складає в межах 350 -500 лк, що відповідає нормованому значенню.

Джерелом шуму в приміщенні є комп'ютер. Вентилятори (кулери) системного блоку, процесора, відеокарти і блоку живлення є сучасними і мають низький рівень шуму. Згідно з технічною документацією шум, зумовлений кулером в блоці живлення складає 25 дБ, кулером процесора - 30 дБ, загальний - 34 дБ. Враховуючи незначний рівень шуму від персонального комп'ютера і незначний рівень фонового шуму від іншого устаткування, можна стверджувати, що сумарний рівень шумового забруднення приміщення не перевищує максимально допустимий рівень коригованої звукової потужності і складає не більше 50 дБА, що відповідає рівню шуму для приміщень з комп’ютерною технікою згідно Державних санітарних правил і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

У приміщенні відсутні джерела інфрачервоного, ультрафіолетового і електромагнітного випромінювання, бо монітор ПК вироблений на основі рідкокристалічної матриці, підсвітка якої здійснюється неоновою лампою, які не мають сильного електромагнітного випромінювання і сертифіковані в Україні.

Блок живлення є екранованим і не випускає вищезазначених видів випромінювання.

8.4 Розробка заходів з умов поліпшення охорони праці

Перерахуємо проведені заходи щодо забезпечення умов праці на робочому місці користувача ПК. З точки зору забезпечення електробезпеки до цих заходів

можна віднести: устаткування розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв; періодична перевірка всіх приладів і пристроїв; щорічна здача іспитів з охорони праці. З точки зору забезпечення оптимальних умов мікроклімату, рівня звуку і освітленості до цих заходів можна віднести: організацію природної вентиляції, за допомогою дефлектора, для забезпечення необхідного повітрообміну в приміщенні вузла; організацію системи центрального опалювання, для підтримки оптимальної температури в холодний період року; організацію штучного загального освітлення, для забезпечення необхідних умов зорової роботи, що відповідають, оформлення паспорта на приміщення вузла, з занесенням в нього вимірювань освітленості і рівня звуку, проведених відділом охорони праці.

Крім рекомендацій щодо конкретного приміщення, де було проведено дослідження умов праці, існують загальні вимоги, які зарекомендовані відповідними нормативними документами.

Правильна організація робочих місць запобігає передчасній втомлюваності користувача і сприяє збереженню здоров'я. Організація робочого місця передбачає:

– правильне розміщення робочого місця у виробничому приміщенні;

– вибір ергономічного обґрунтованого робочого положення, виробничих меблів з урахуванням характеристик людини;

– раціональне компонування обладнання на робочих місцях;

– урахування характеру й особливостей трудової діяльності. Стосовно робочих місць користувача ВДТ, то організація робочого має забезпечуватися відповідно до ДСанПіН 3.3.2-007-98. Для запобігання перевтомленню необхідно виконувати вправи для очей та дотримуватись розпорядку роботи та відпочинку. На робочому місці реалізовувався режим відпочинку: кожні дві години – перерва для виконання фізичних вправ для м‘язів очей.

8.5 Протипожежний захист

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.

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 якому встановлено комп'ютери і де немає необхідності влаштування систем автоматичного пожежогасіння, необхідно оснащувати переносними вуглекислотними вогнегасниками з розрахунку 2 шт. на кожні 20 $м^2$ в приміщеннях.

Звуковбирне облицювання стін, стель приміщень треба виконувати з негорючих та важко горючих матеріалів.

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

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

8.6 Розрахункова частина

В приміщенні (де відсутні джерела виділення шкідливих речовин) працює одна людина. Робота пов'язана з використанням ПЕОМ. Розміри приміщення: А = 4 м, В = 3.5 м, Н = 2.8 м, устаткування займає 15% об'єму. Визначимо найменшу необхідну кількість повітря для вентиляції.

Для приміщень, в яких відсутні виділення шкідливих речовин у повітрі, розрахунок вентиляції здійснюється залежно від кількості працюючих.

Необхідна кількість повітря ($м^3$/год.), яка забезпечує відповідність параметрів повітря робочої зони нормованим значенням, визначається за наступною формулою:

$$L = L' \cdot N,$$

де L' - нормативна кількість повітря на одного працюючого, яка залежить від питомого об'єму приміщення, $м^3$/(год.-люд.);

N - кількість працюючих.

Питомий об'єм приміщення $V_п$, ($м^3$ /люд.), визначається за формулою:

$$V_п = V/N,$$

де V - об'єм приміщення, $м^3$.

Визначаємо вільний об'єм приміщення

$$V = A \cdot B \cdot H \cdot 0,85 = 4 \cdot 3,5 \cdot 2,8 \cdot 0,85 = 33,3 \text{ м}^3.$$

Питомий вільний об'єм складає

$$V' = V / N = 33,2 / 1 = 33,2 \text{ м}^3/\text{люд.} > 20 \text{ м}^3/\text{люд.}$$

Нормована кількість повітря на одну людину при V' > 20 $м^3$ /люд. становить 30 $м^3$ /(год.·люд.).

Висновки до розділу

У даному розділі магістерської роботи проведено аналіз умов працівника робота якого пов'язана з комп'ютерною технікою. Проведено аналіз основних санітарно – гігієнічних показників в заданому приміщені , де працівник зайнятий постійною роботою за комп'ютером.. Створені умови повинні забезпечувати

комфортну роботу. На підставі вивченої літератури з даної проблеми, були зазначені оптимальні параметри мікроклімату, освітлення, допустимі рівні шуму та іонізуючого випромінювання при роботі з ПЕОМ, а також розраховано найменшу необхідну кількість повітря для вентиляції.

Дотримання умов, що визначають оптимальну організацію робочих місць працівників, дозволить зберегти гарну працездатність протягом усього робочого дня, підвищить як в кількісному, так і в якісному відносинах продуктивність їх праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи центру обробки даних SSDC.

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 центру обробки даних SSDC.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем центру обробки даних SSDC.

– Досліджена система центру обробки даних SSDC.

– На основі отриманих результатів досліджень створена програмна реалізація системи центру обробки даних SSDC.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання центру обробки даних SSDC.

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

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

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

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 4145-2002.

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. Підгорна Ю.В. Дослідження та програмна реалізація системи центру обробки даних SSDC // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2024.

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

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

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

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

6. 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.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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.

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

13. 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.

14. 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.

15. 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.

16. 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.

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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

37. 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.

38. 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.

39. 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.

40. 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.

41. 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.

42. 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.

43. 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).

44. 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

45. 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.

46. 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.

47. 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.

48. 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.

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

50. 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.

51. 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.

52. 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.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи центру обробки даних SSDC.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 21-13 від 07.08.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи центру обробки даних SSDC.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи центру обробки даних SSDC;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати маркетингове та економічне обґрунтування ІТ-проєкту з урахуванням цін на 3 вересня 2024 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці користувача ПК.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 113 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Маркетингове та економічне обґрунтування ІТ-проєкту.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 02.12.2024 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 18.12.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Коваленко А.С.

Дослідження та програмна реалізація системи центру обробки даних SSDC

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 19

Літера: РП

Кропивницький – 2024 року

Основна програма

```
# Основна система центру обробки даних SSDC
# Архітектура побудована на модульному підході

import random
import math
from datetime import datetime

# Модуль для управління обладнанням серверного центру
class Server:
    def __init__(self, server_id, cpu_capacity, ram_capacity, ssd_capacity):
        self.server_id = server_id
        self.cpu_capacity = cpu_capacity
        self.ram_capacity = ram_capacity
        self.ssd_capacity = ssd_capacity
        self.current_load = 0

    def add_load(self, load):
        # Додаємо навантаження на сервер
        if self.current_load + load <= self.cpu_capacity:
            self.current_load += load
        else:
            raise ValueError("Перевищено обчислювальну потужність сервера")

    def clear_load(self):
        # Очищаємо сервер від навантаження
        self.current_load = 0

# Модуль для моніторингу навантаження
class LoadBalancer:
    def __init__(self):
        self.servers = []

    def add_server(self, server):
        # Додаємо сервер у систему
        self.servers.append(server)

    def distribute_load(self, load):
        # Розподіляємо навантаження по серверах
        for server in self.servers:
            if server.current_load + load <= server.cpu_capacity:
                server.add_load(load)
                return f"Навантаження {load} додано до сервера
{server.server_id}"
        return "Немає достатньо ресурсів для навантаження"

# Модуль для управління енергоспоживанням
class PowerManagement:
    def __init__(self, base_power, server_count):
        self.base_power = base_power
        self.server_count = server_count

    def calculate_power(self, active_servers):
        # Розраховуємо споживання енергії
        power_usage = self.base_power * active_servers
        return power_usage

# Основна система центру
class SSDCSystem:
    def __init__(self):
        self.servers = []
        self.load_balancer = LoadBalancer()
        self.power_management = None
        self.data_processed = 0

    def setup_servers(self, num_servers):
        # Ініціалізуємо сервери
        for i in range(num_servers):
```

```
            server = Server(
                server_id=f"S{i + 1}",
                cpu_capacity=random.randint(50, 100),
                ram_capacity=random.randint(128, 256),
                ssd_capacity=random.randint(1024, 2048)
            )
            self.servers.append(server)
            self.load_balancer.add_server(server)
        self.power_management = PowerManagement(base_power=500,
server_count=num_servers)

    def process_data(self, data_load):
        # Обробляємо дані
        load_per_task = math.ceil(data_load / len(self.servers))
        for i in range(len(self.servers)):
            self.load_balancer.distribute_load(load_per_task)
        self.data_processed += data_load

    def monitor_energy(self):
# Моніторимо споживання енергії
   active_servers = sum(1 for server in self.servers if server.current_load > 0)
        return self.power_management.calculate_power(active_servers)

# Головна функція для запуску системи
def main():
    # Ініціалізуємо систему SSDC
    ssdc = SSDCSystem()
    num_servers = 5
    ssdc.setup_servers(num_servers)

    # Симуляція обробки даних
    data_loads = [200, 400, 300]
    for load in data_loads:
        ssdc.process_data(load)
print(f"Енергоспоживання після обробки {load} ГБ: {ssdc.monitor_energy()} Вт")

    # Загальний звіт
    print(f"Загальна кількість оброблених даних: {ssdc.data_processed} ГБ")

# Запускаємо головну функцію
if __name__ == "__main__":
    main()
```

backup_manager.py

```
# Модуль для управління резервним копіюванням даних

import os
import shutil
from datetime import datetime

class BackupManager:
    def __init__(self, backup_dir="backups"):
        # Ініціалізація менеджера резервного копіювання
        self.backup_dir = backup_dir
        if not os.path.exists(backup_dir):
            os.makedirs(backup_dir)

    def create_backup(self, data_dir):
        # Створюємо резервну копію даних
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_path = os.path.join(self.backup_dir, f"backup_{timestamp}")
        shutil.copytree(data_dir, backup_path)
        return backup_path

    def list_backups(self):
        # Список доступних резервних копій
        return os.listdir(self.backup_dir)

    def restore_backup(self, backup_name, restore_dir):
        # Відновлюємо дані з резервної копії
        backup_path = os.path.join(self.backup_dir, backup_name)
        if os.path.exists(backup_path):
            shutil.copytree(backup_path, restore_dir, dirs_exist_ok=True)
        else:
            raise FileNotFoundError("Резервна копія не знайдена")

# Приклад використання
if __name__ == "__main__":
    backup_manager = BackupManager()
    backup_manager.create_backup("data")  # Каталог даних, який потрібно
резервувати
    print("Список резервних копій:", backup_manager.list_backups())
```

Файл autoscaling.py

```
# Модуль для автоматичного масштабування серверів

class AutoScaler:
    def __init__(self, max_servers=20):
        # Ініціалізація системи масштабування
        self.max_servers = max_servers
        self.active_servers = 0
        self.server_pool = []

    def add_server(self):
        # Додаємо новий сервер
        if self.active_servers < self.max_servers:
            self.server_pool.append(f"Server_{self.active_servers + 1}")
            self.active_servers += 1
            return f"Додано сервер Server_{self.active_servers}"
        else:
            return "Досягнуто максимальну кількість серверів"

    def remove_server(self):
        # Видаляємо сервер
        if self.active_servers > 0:
            server = self.server_pool.pop()
            self.active_servers -= 1
            return f"Видалено сервер {server}"
        else:
            return "Немає серверів для видалення"

# Приклад використання
if __name__ == "__main__":
    autoscaler = AutoScaler()
    print(autoscaler.add_server())
    print(autoscaler.add_server())
    print(autoscaler.remove_server())
```

Файл monitoring_integration.py

```
# Модуль для інтеграції з системами моніторингу

class ExternalMonitoring:
    def __init__(self, system_name, api_key):
        # Ініціалізація зовнішньої системи моніторингу
        self.system_name = system_name
        self.api_key = api_key

    def send_metrics(self, metrics):
        # Імітація відправки метрик до зовнішньої системи
        print(f"Відправляємо метрики в {self.system_name}: {metrics}")
        return "Метрики успішно передані"

    def receive_alerts(self):
        # Імітація отримання сповіщень від зовнішньої системи
        return f"Отримано нові сповіщення з {self.system_name}"

# Приклад використання
if __name__ == "__main__":
    monitoring = ExternalMonitoring("Prometheus", "test-api-key")
    metrics = {"cpu_usage": 75, "memory_usage": 60}
    print(monitoring.send_metrics(metrics))
    print(monitoring.receive_alerts())
```

Файл ai_optimizer.py

```
# Модуль для оптимізації розподілу навантаження за допомогою ШІ

import random

class AILoadOptimizer:
    def __init__(self, servers):
        # Ініціалізація оптимізатора
        self.servers = servers

    def optimize_distribution(self, load):
        # Розподіл навантаження за допомогою алгоритму
        recommendations = []
        for server in self.servers:
            if server.cpu_capacity - server.current_load >= load:
                recommendations.append(server.server_id)
        return recommendations or ["Немає серверів з достатньою потужністю"]

# Приклад використання
if __name__ == "__main__":
    from ssdc_system import Server
    server1 = Server("S1", 100, 128, 1024, 400)
    server2 = Server("S2", 80, 128, 2048, 300)
    optimizer = AILoadOptimizer([server1, server2])
    print(optimizer.optimize_distribution(50))
```

Файл load_prediction.py

```
# Модуль для прогнозування навантаження системи

import random
from datetime import datetime, timedelta

class LoadPredictor:
    def __init__(self, history_data):
        # Ініціалізація модуля прогнозування
        self.history_data = history_data

    def predict_next_hour(self):
        # Прогноз навантаження на наступну годину
        avg_load = sum(self.history_data[-5:]) / len(self.history_data[-5:])
        predicted_load = avg_load + random.randint(-10, 10)
        return max(predicted_load, 0)

    def predict_next_day(self):
        # Прогноз на наступні 24 години
        return [self.predict_next_hour() for _ in range(24)]

# Приклад використання
if __name__ == "__main__":
    history = [random.randint(50, 100) for _ in range(10)]
    predictor = LoadPredictor(history)
    print("Прогноз на наступну годину:", predictor.predict_next_hour())
    print("Прогноз на 24 години:", predictor.predict_next_day())
```

Файл cybersecurity_integration.py

```
# Модуль для інтеграції системи з рішеннями кібербезпеки

class CyberSecurity:
    def __init__(self, threat_database):
        # Ініціалізація модуля кібербезпеки
        self.threat_database = threat_database

    def scan_for_threats(self, logs):
        # Сканування журналів на наявність загроз
        threats_found = [threat for threat in logs if threat in
self.threat_database]
        return threats_found

    def report_threats(self, threats):
        # Формування звіту про виявлені загрози
        if threats:
            print("Виявлено загрози:", threats)
        else:
            print("Загроз не виявлено")

# Приклад використання
if __name__ == "__main__":
    database = ["SQL Injection", "DDoS", "Ransomware"]
    logs = ["SQL Injection", "Phishing", "Data Breach"]
    security = CyberSecurity(database)
    threats = security.scan_for_threats(logs)
    security.report_threats(threats)
```

Файл cloud_integration.py

```
# Модуль для інтеграції з хмарними технологіями

class CloudIntegration:
    def __init__(self, cloud_provider, api_key):
        # Ініціалізація модуля хмарної інтеграції
        self.cloud_provider = cloud_provider
        self.api_key = api_key

    def upload_data(self, data):
        # Імітація завантаження даних у хмару
        print(f"Дані завантажено у {self.cloud_provider}: {data}")
        return "Дані успішно збережено"

    def download_data(self, data_id):
        # Імітація завантаження даних з хмари
        print(f"Завантажуємо дані {data_id} з {self.cloud_provider}")
        return f"Дані {data_id} успішно завантажено"

# Приклад використання
if __name__ == "__main__":
    cloud = CloudIntegration("AWS", "cloud-api-key")
    cloud.upload_data("Файли системи")
    cloud.download_data("backup_12345")
```

Файл dmin_gui.py

```
# Модуль графічного інтерфейсу для адміністрування

import tkinter as tk

class AdminGUI:
    def __init__(self):
        # Ініціалізація графічного інтерфейсу
        self.window = tk.Tk()
        self.window.title("Адміністрування SSDC")

    def run(self):
        # Створення елементів інтерфейсу
        tk.Label(self.window, text="Виберіть дію:").pack()
        tk.Button(self.window, text="Переглянути звіти",
command=self.view_reports).pack()
        tk.Button(self.window, text="Моніторинг серверів",
command=self.monitor_servers).pack()
        tk.Button(self.window, text="Вихід", command=self.window.quit).pack()
        self.window.mainloop()

    def view_reports(self):
        # Обробка вибору перегляду звітів
        print("Відображення звітів")

    def monitor_servers(self):
        # Обробка вибору моніторингу серверів
        print("Моніторинг серверів")

# Приклад запуску
if __name__ == "__main__":
    gui = AdminGUI()
    gui.run()
```

Файл virtualization_support.py

```
# Модуль для підтримки платформ віртуалізації

class VirtualMachine:
    def __init__(self, vm_id, cpu, memory, storage):
        # Ініціалізація віртуальної машини
        self.vm_id = vm_id
        self.cpu = cpu
        self.memory = memory
        self.storage = storage
        self.status = "stopped"

    def start(self):
        # Запуск віртуальної машини
        if self.status == "stopped":
            self.status = "running"
            return f"Віртуальна машина {self.vm_id} запущена"
        else:
            return f"Віртуальна машина {self.vm_id} вже працює"

    def stop(self):
        # Зупинка віртуальної машини
        if self.status == "running":
            self.status = "stopped"
            return f"Віртуальна машина {self.vm_id} зупинена"
        else:
            return f"Віртуальна машина {self.vm_id} вже зупинена"

class VirtualizationManager:
    def __init__(self):
        # Ініціалізація менеджера віртуалізації
        self.vms = {}

    def create_vm(self, vm_id, cpu, memory, storage):
        # Створення нової віртуальної машини
        if vm_id not in self.vms:
            self.vms[vm_id] = VirtualMachine(vm_id, cpu, memory, storage)
            return f"Віртуальна машина {vm_id} створена"
        else:
            return "Віртуальна машина з таким ID вже існує"

    def manage_vm(self, vm_id, action):
        # Управління віртуальною машиною
        if vm_id in self.vms:
            vm = self.vms[vm_id]
            if action == "start":
                return vm.start()
            elif action == "stop":
                return vm.stop()
            else:
                return "Невідома дія"
```

```
        else:
            return "Віртуальна машина не знайдена"

# Приклад використання
if __name__ == "__main__":
    manager = VirtualizationManager()
    print(manager.create_vm("vm1", 4, 16, 200))
    print(manager.manage_vm("vm1", "start"))
    print(manager.manage_vm("vm1", "stop"))
```

Файл energy_forecasting.py

```
# Модуль для прогнозування енергоспоживання

class EnergyForecaster:
    def __init__(self, energy_data):
        # Ініціалізація модуля енергопрогнозування
        self.energy_data = energy_data

    def forecast_next_hour(self):
        # Прогноз енергоспоживання на наступну годину
        avg_consumption = sum(self.energy_data[-5:]) / len(self.energy_data[-
5:])
        forecast = avg_consumption * 1.05  # Додаємо невелике зростання
        return forecast

    def forecast_next_day(self):
        # Прогноз на наступні 24 години
        return [self.forecast_next_hour() for _ in range(24)]

# Приклад використання
if __name__ == "__main__":
    energy_usage = [100, 105, 110, 95, 102, 98, 104]
    forecaster = EnergyForecaster(energy_usage)
    print("Прогноз на наступну годину:", forecaster.forecast_next_hour())
    print("Прогноз на 24 години:", forecaster.forecast_next_day())
```

Файл data_analytics.py

```
# Модуль для аналітики даних

class DataAnalyzer:
    def __init__(self, data):
        # Ініціалізація модуля аналітики даних
        self.data = data

    def calculate_average(self):
        # Розрахунок середнього значення
        return sum(self.data) / len(self.data) if self.data else 0

    def find_peak_values(self):
        # Пошук пікових значень
        return max(self.data), min(self.data)

# Приклад використання
if __name__ == "__main__":
    analytics_data = [120, 150, 130, 170, 160]
    analyzer = DataAnalyzer(analytics_data)
    print("Середнє значення:", analyzer.calculate_average())
    print("Пікові значення:", analyzer.find_peak_values())
```

Файл api_integration.py

```
# Модуль для інтеграції через API

class APIIntegration:
    def __init__(self, base_url, api_key):
        # Ініціалізація модуля інтеграції API
        self.base_url = base_url
        self.api_key = api_key

    def send_request(self, endpoint, payload):
        # Імітація запиту до API
        print(f"Запит до {self.base_url}{endpoint} з даними {payload}")
        return {"status": "success", "response": "Дані отримано"}

# Приклад використання
if __name__ == "__main__":
    api = APIIntegration("https://api.example.com", "key123")
    response = api.send_request("/data", {"query": "test"})
    print(response)
```

Файл disaster_recovery.py

```
# Модуль для реалізації плану відновлення після аварій

class DisasterRecovery:
    def __init__(self, backup_manager, critical_services):
        # Ініціалізація плану відновлення
        self.backup_manager = backup_manager
        self.critical_services = critical_services

    def simulate_failure(self):
        # Симуляція відмови критичних служб
        print("Симуляція аварії. Критичні служби недоступні:",
self.critical_services)

    def recover_from_failure(self, backup_name):
        # Відновлення системи з резервної копії
        try:
            self.backup_manager.restore_backup(backup_name, "restored_data")
            print(f"Система відновлена з резервної копії {backup_name}")
        except FileNotFoundError as e:
            print(e)

# Приклад використання
if __name__ == "__main__":
    from backup_manager import BackupManager

    backup_manager = BackupManager()
    recovery = DisasterRecovery(backup_manager, ["Database", "Web Server",
"Cache"])
    recovery.simulate_failure()
    recovery.recover_from_failure("backup_20241112_103000")
```

Файл resource_management.py

```
# Модуль для управління ресурсами системи

class ResourceManager:
    def __init__(self):
        # Ініціалізація менеджера ресурсів
        self.resources = {}

    def add_resource(self, resource_name, capacity):
        # Додавання ресурсу
        self.resources[resource_name] = {"capacity": capacity, "used": 0}
        return f"Ресурс {resource_name} додано"

    def allocate_resource(self, resource_name, amount):
        # Виділення ресурсу
        if resource_name in self.resources:
            resource = self.resources[resource_name]
            if resource["used"] + amount <= resource["capacity"]:
                resource["used"] += amount
                return f"{amount} одиниць ресурсу {resource_name} виділено"
            else:
                return "Недостатньо доступного ресурсу"
        else:
            return "Ресурс не знайдено"

    def release_resource(self, resource_name, amount):
        # Звільнення ресурсу
        if resource_name in self.resources:
            resource = self.resources[resource_name]
            resource["used"] = max(0, resource["used"] - amount)
            return f"{amount} одиниць ресурсу {resource_name} звільнено"
        else:
            return "Ресурс не знайдено"

# Приклад використання
if __name__ == "__main__":
    manager = ResourceManager()
    print(manager.add_resource("CPU", 100))
    print(manager.allocate_resource("CPU", 40))
    print(manager.release_resource("CPU", 20))
```

Файл network_optimization.py

```
# Модуль для оптимізації мережі

class NetworkOptimizer:
    def __init__(self, bandwidth_data):
        # Ініціалізація оптимізатора мережі
        self.bandwidth_data = bandwidth_data

    def optimize_bandwidth(self):
        # Оптимізація використання пропускної здатності
        avg_bandwidth = sum(self.bandwidth_data) / len(self.bandwidth_data)
        optimal_allocation = [max(10, value - 5) for value in
self.bandwidth_data]
        return {"average_bandwidth": avg_bandwidth, "optimized":
optimal_allocation}

# Приклад використання
if __name__ == "__main__":
    bandwidth = [100, 120, 80, 150, 110]
    optimizer = NetworkOptimizer(bandwidth)
    print(optimizer.optimize_bandwidth())
```