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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Грицай Г.Г.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук
_____ Лисенко І.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

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

Грицаю Глібу Геннадійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale*

2. Керівник роботи *Лисенко Ірина Анатоліївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 46-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи передачі даних між дата-центрами на основі технології Web Scale*

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

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

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

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

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Лисенко І.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Грицай Г.Г.
(прізвище та ініціали)

АНОТАЦІЯ

Грицай Г.Г. Програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи передачі даних між дата-центрами на основі технології Web Scale.

Метою розробки є програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale.

Результат роботи – програмна реалізація системи передачі даних між дата-центрами на основі технології Web Scale.

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, Web Scale

ABSTRACT

Hrytsay H.H. Software for a data transfer system between data centers based on Web Scale technology. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for a data transfer system between data centers based on Web Scale technology.

The purpose of the development is software for a data transfer system between data centers based on Web Scale technology.

The result of the work is a software implementation of a data transfer system between data centers based on Web Scale technology.

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

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

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

The program was developed in the Python environment.

Keywords: computer engineering, Web Scale

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи..... 5
1.2 Область застосування..... 7
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 23
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти..... 23
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 23
2.3 Розгорнута постановка завдання 24
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 26
3.1 Опис функціонування системи 26
3.2 Розробка структурної схеми..... 29
3.3 Розробка функціональної схеми 41
3.4 Розробка діаграми процесів..... 44
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 46
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 46
4.2 Захист розробленого програмного забезпечення..... 62
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 64
6 ОСНОВНІ ВИСНОВКИ..... 69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 71

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

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

ВСТУП

Актуальність теми. Поширення хмарних послуг і віртуалізації привело до появи безлічі ЦОДів, результатом чого став ефект Web Scale – масштабування Web. Це стало стимулом для постачальників інтернет-контента й хмарних послуг запропонувати своїм клієнтам різноманітні додатки й надати доступ до контенту величезної безлічі користувачів. Основою цих перетворень є мережа. Доступність ЦОДа у світі Web Scale прямо залежить від операційної простоти, масштабованості й ефективній бізнесу-моделі.

Сьогодні ЦОДи піддаються радикальної трансформації, націленої на забезпечення ефективної роботи в умовах постійно змінюваного трафіку, збільшення потреби в пропускній здатності й широкому поширенні додатків. Ще ніколи одержання миттєвого доступу до інтернет-контенту не було так важливо для нас. Сьогодні він затребуваний більше, ніж коли-або, і це стало одним з найважливіших факторів, що спонукали постачальників послуг зайнятися забезпеченням повсюдного доступу до інформації, що цікавить клієнтів, причому в будь-який час і на будь-якому пристрої.

Через особливості сучасних послуг і додатків користувальницький контент повинен доставлятися миттєво, з дотриманням високих стандартів якості. Безпрецедентне поширення мобільних пристроїв, насамперед смартфонів і планшетів, і нових накладених послуг (Over-The-Top, OTT), таких як потокове відео й онлайн-ігри, стимулювало будівництво додаткових ЦОДів як у пригородах, так і в сільських районах.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale.

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

– Огляд існуючих систем передачі даних між дата-центрами на основі технології Web Scale.

– Дослідження системи передачі даних між дата-центрами на основі технології Web Scale.

– Програмна реалізація системи передачі даних між дата-центрами на основі технології Web Scale.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі передачі даних між дата-центрами на основі технології Web Scale.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Додатковий імпульс тенденції будівництва додаткових ЦОДів надає популярність засобів віртуалізації: корпоративні IT-підрозділи сьогодні переходять від власних замовлених IT-додатків до стандартних хмарних рішень, контент яким зберігається в ЦОДе. З урахуванням росту кількості центрів обробки даних і обсягу запитуваної інформації, а також все більшого поширення віртуалізації серверів і обчислювальних пристроїв, міжз'єднання ЦОДів (Data Center Interconnet, DCI) стає одним із ключових факторів забезпечення успішної реалізації додатків верхнього рівня. Інакше кажучи, якщо до ЦОДу не можна підключитися, його не можна використовувати.

Більше того, віртуалізація торкнулася усіх компонентів ЦОДа. Сучасні технології дозволяють використовувати два або декілька ЦОДів як один – з поділом навантаження й завдань із метою відомості до мінімуму експлуатаційних витрат і досягнення максимальної продуктивності. Наприклад, ресурси двох відносно близько розташованих центрів обробки даних, що перебувають у великому місті, де ціни на нерухомість високі, можна об'єднати для доступу й зберігання даних, усунувши необхідність побудови нових ЦОДів.

Типи ЦОДів

Існує два типи центрів обробки даних. До першого ставляться великі ЦОДи, побудовані у віддалених районах з низькими цінами на землю й енергоресурси. Вони розташовуються в безпосередній близькості до магістральних волоконно-оптичним каналам. Витрати на оренду й електроенергію й сейсмічні ризики тут невисокі. У таких ЦОДах найчастіше розміщаються тисячі серверів. Підприємства використовують їх для резервування й дзеркалювання даних у реальному часі в рамках програм по забезпеченню

безперебійного функціонування / післяаварійного відновлення. Ці ЦОДи можуть бути розташовані в сотнях і навіть тисячах кілометрів від корпоративних офісів.

ЦОДи другого типу розміщаються в міському середовищі, ближче до кінцевих користувачів. Вони забезпечують максимальну якість обслуговування клієнтів, зводячи до мінімуму затримки за рахунок локалізації доставки контента. У багатьох випадках такі ЦОДи необхідно зв'язати між собою, щоб наблизити контент до користувача й одночасно підвищити продуктивність додатків. Такі ЦОДи, як правило, розміщаються недалеко від центра міста, проникаючи згодом у пригороди й прилягаючі до них райони.

Мережний обмін даними – найважливіший аспект основних функцій стандартного ЦОДа (незалежно від його типу). Він вирішує наступні завдання:

– Організація з'єднань між серверами для комп'ютерів і обчислювальних пристроїв з високою швидкодією: необхідна додаткам для розрахунку різних функцій, таких як пошук швидкого маршруту між двома кінцевими крапками в додатках навігації й GPS, для хмарної обробки даних, біллинга й керування документами.

– Доступ до пристроїв зберігання: до високоємних дисків, які містять даними, збереженими або використовувані додатками (наприклад, додатками для роботи з електронною поштою, онлайн-фотографіями й відео). Пристрою зберігання також служать для резервного копіювання корпоративних даних (при їхньому дублюванні або дзеркалюванні) з метою захисту корпоративних інформаційних активів від стихійних лих або ушкоджень.

– З'єднання між різними ЦОДами, а також між ЦОДами й зовнішнім миром: дані необхідно відправляти в ЦОД, переміщати між серверами й пристроями зберігання даних або в їхньому середовищі. Крім того, необхідно з'єднати ЦОД із зовнішнім миром, у тому числі з мережею компанії й іншими ЦОДами – як прилеглими, так і, що перебувають на значній відстані друг від друга.

Мережна взаємодія дозволяє серверу й системам зберігання обслуговувати додатка. Якщо якісний зв'язок недоступний, додаток не зможе працювати! Оскільки саме ЦОДи є основою хмарних середовищ, їхня ефективність прямо залежить від міжз’єднання.

1.2 Область застосування

Проблеми міжз’єднання ЦОДів

Ефективна робота ЦОДів у сучасних умовах, коли постійно міняється картина трафіку, росте попит на пропускну здатність і поширюються різні додатки, вимагає радикальної трансформації ЦОДів. Міжз’єднання стало одним із ключових факторів забезпечення успішної реалізації додатків верхнього рівня. Не можна, однак, не відзначити й ряд пов'язаних з ним проблем.

Дистанційні обмеження. Для ЦОДа нерідко потрібні з'єднання з мінімальною затримкою, щоб підтримувати необхідний потік інформації й синхронізацію між передавальну інформацію сервером і пристроєм зберігання даних. Якщо ЦОДи перебувають далеко друг від друга, затримки при обміні даними на мережному встаткуванні, за допомогою якого виробляється їх міжз’єднання, зростають. Затримку, обумовлену характеристиками оптичного волокна, можна звести до мінімуму, вибравши найкоротший фізичний маршрут. Для зменшення затримок, обумовлених характеристиками ПЗ й устаткування, необхідно використовувати передові практики проектування.

Ємність. Дуже часто сукупний обсяг масивів даних додатків, що надходять у ЦОД або виходять із нього, досить великий і досягає сотень гігабайт або навіть терабайт. Телекомунікаційне устаткування повинне забезпечувати надійні високоємні з'єднання з можливістю подальшого масштабування в міру необхідності.

Безпека. Корпоративні дані, особисті записи й дані фінансових транзакцій найчастіше носять конфіденційний характер і мають найважливіше значення для бізнесу, тому всі мережні з'єднання ЦОДа повинні бути надійно захищені.

Керування. Мережні операції, виконувані вручну, трудомісткі, складні, повільні й спричиняють помилки. Надто важливо звести їхня кількість до мінімуму за рахунок автоматизації виконання регулярних і численних завдань адміністрування. Процес розгортання з'єднання між двома ЦОДами повинен бути простий і надійний без необхідності постійно виконувати експлуатаційні операції вручну.

Витрати. Очікуваний середньорічний темп росту трафіку становить близько 30%. Витрати на зміст мережі повинні рости набагато повільніше (у противному випадку ЦОДи можуть виявитися нежиттєздатними з фінансової точки зору).

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Огляд рішення Cisco UCS (Unified Computing System)

Рішення Cisco United Computing System (UCS) – це платформа для ЦОД наступного покоління, що поєднує обчислювальні й мережні ресурси, доступ до систем зберігання даних, а також засобу віртуалізації в єдину систему. Рішення Cisco UCS дозволяє значно знизити витрати на експлуатацію ЦОД і збільшує операційну ефективність бізнесу.

Рішення Cisco UCS поєднує в собі універсальну високошвидкісну мережну інфраструктуру (United fabric) на базі технологій 10-Gigabit Ethernet і широко використовувану x-86 архітектуру серверів компанії Intel. Рішення Cisco UCS дозволяє створити інтегровану й масштабовану платформу, у якій всі елементи рішення управляються централізовано.

Рішення Cisco UCS має у своєму составі єдину систему керування, що дозволяє управляти комплексами до 160 фізичних серверів, на кожному з яких можуть функціонувати десятки віртуальних машин, забезпечуючи чудове масштабування ЦОД без збільшення складності керування.

Архітектура системи Cisco UCS поліпшує переносимість профілів фізичних машин, тому що профіль сервера, конфігурація його LAN/SAN з'єднань і I/O, убудованого ПЗ й профілі мережних з'єднань можуть бути динамічно привласнені будь-якому фізичному серверу в системі. Така високодинамічне середовище з підтримкою принципу Stateless Computing можуть бути легко адаптована для задоволення швидко мінливих вимог сучасних центрів обробки даних. Це має на увазі впровадження в потрібний момент необхідних

обчислювальних ресурсів і переміщення робочого навантаження. Cisco United Computing System поліпшує доступність, безпеку й продуктивність завдяки інтегрованому дизайну системи.

Компоненти системи UCS:

– Центральні комутатори UCS 6200 Series Fabric.

– Шасі блейд-серверів UCS 5100 Series.

– Мережні модулі UCS 2200 Series Fabric Extender.

– Блейд-сервери UCS B-Series.

– Стійкові-сервери UCS C-Series.

– Мережні адаптери UCS.

– Модуль керування UCS Manager.

Центральні комутатори UCS серії 6200 Fabric Interconnect

Конвергентні комутатори, що сполучать функції комутації трафіку ethernet і fibre channel з керуванням системою UCS. Архітектура комутаторів передбачає комутацію трафіку на швидкостях до 10 Гбіт/с, без втрат пакетів і із украй низькими затримками. Комутатори поставляються в корпусі 1RU з 48 портами або в корпусі 2RU з 96 портами. Підтримують модулі розширення, що забезпечують підключення Fibre Channel і 10 Gigabit Ethernet.

Основні функції:

– 10 Gigabit Ethernet порти SFP+, підтримка FCoS.

– Варіанти 48 і 96 убудованих портів зі слотами розширення для додавання портів Fibre Channel і 10 GE.

– Убудована система керування UCS Manager.

– До 1.04 Tbps продуктивності.

– Зарезервовані блоки живлення й вентилятори з «гарячою заміною».

– Керування до 40 шасі на систему UCS.

Мережний модуль UCS серії 2200 Fabric Extender

Модулі для забезпечення зв'язку між центральними комутаторами й блейд-серверами. З їхньою допомогою спрощуються процеси діагностики, підключення кабелів і керування системою.

Основні функції:

– Підключення блейд-шасі UCS до центральних комутаторів (Fabric Interconnect).

– 8 зовнішніх порту 10 Gigabit Ethernet SFP+ з підтримкою FCoS.

– До двох модулів на шасі для забезпечення відказостійкості й до 80 Гбіт/с повнодуплексної продуктивності.

– Убудоване керування шасі.

– Управляється UCS Manager через центральний комутатор.

Шасі для блейд-серверів UCS серії 5100

Являє собою серверний кошик, що підтримує до восьми блейд-серверів і до двох мережних модулів у корпусі 6RU, не вимагаючи додаткових модулів керування.

Основні функції:

– 4 блоки живлення (резервування за схемою: N+1 або N+N).

– 8 вентиляторів (за замовчуванням).

– Охолодження « попереду-назад».

– Висота шасі 6U.

– Усередину можна встановити до 8 блейд-серверів половинної ширини або 4 блейд-сервери повної ширини.

– Всі блейд-сервери одинарної висоти.

– Установка до 2-х мережних модулів.

– Індикаторна ідентифікація.

Блейд-сервери UCS (серія B)

Блейд-сервери UCS – це блейд-сервери архітектури x86, на базі процесорів Intel Xeon, які адаптуються під вимоги додатків, регулюють

використання електроенергії й забезпечують кращу віртуалізацію серед пристроїв свого класу. Унікальна технологія розширення пам'яті Cisco значно збільшує обсяг пам'яті, що підвищує продуктивність і пропускну здатність для ресурсномістких додатків віртуалізації й обробки великих наборів даних. Крім того, ця технологія пропонує більше економічний варіант пам'яті для менш вимогливих робочих навантажень.

Стійкові сервери UCS (серія C)

Стійкові сервери UCS – сервери, призначені для роботи в автономних середовищах і в складі середовища уніфікованих обчислень Cisco, виконані в стандартному конструктивному виконанні. Вони підтримує модель покрокового розгортання з можливістю майбутнього переходу на уніфіковані обчислення. Мережні адаптери Cisco UCS

У цей час, всі мережні адаптери для блейд-систем умовно можна поділити на три типи: традиційні, конвергентні й віртуалізовані:

– Традиційні адаптери – класичні адаптери з апаратною підтримкою протоколу Ethernet і програмною підтримкою конвергентного протоколу Fibre Channel over Ethernet, призначені для передачі й обміну даними в мережі. Найбільш відомий виробник таких адаптерів – компанія Intel, Broadcom.

– Конвергентні адаптери – адаптери з інтегрованими чипами протоколів 10Gb і Fibre Channel або які мають конвергентний чип протоколу FCoS. На даний момент це найбільш використовуваний тип адаптерів в існуючих серверних системах. Найпоширеніші виробники цих типів – компанії Emulex і Qlogic.

– Віртуалізовані адаптери – адаптери, які дозволяють створити кілька логічних адаптерів як у динаміку (наприклад, при інтеграції із системами віртуалізації), так і статично. Використання віртуалізированого адаптера в середовищі гіпервізора дозволяє перенести завдання комутації трафіку між мережними машинами на мережне устаткування, що дозволяє визволити процесорні ресурси й надати їхнім віртуальним машинам, тим самим підвищити їхня продуктивність у цілому.

Компанія Cisco Systems розробила власний віртуалізированый двопортовий конвергентний адаптер Cisco Virtualized Interface Card з винятковими характеристиками, що дозволяє мати перевагу перед іншими виробниками:

– Здатний створити до 58 віртуалізированих Ethernet або FC адаптерів (на одному фізичному адаптері з використанням мітки VNTag проекту стандарту IEEE 802.1Qbh).

– Підтримка стандарту PCIe.

– Робота у віртуалізированому і невіртуалізированому середовищу.

– Підтримка Hypervisor Bypass.

– Відказостійкість на апаратному рівні.

– Висока продуктивність (2x 10Gb, >600K IOPS).

У стійкові сервери UCS використовуються найбільш затребувані мережні адаптери у вигляді карт PCIe, таких виробників як QLogic, Broadcom, Emulex і Intel, а також віртуалізированый PCIe адаптер Cisco VIC.

Програмний модуль UCS manager

UCS Manager – це убудований програмний модуль, за допомогою якого, здійснюється керування всіма компонентами блейд-системи.

Інтерфейс UCS Manager розділений на п'ять зон: адміністрування, устаткування, сервери, ЛОМ і мережа зберігання. Під адмініструванням маються на увазі операції, вироблені за допомогою UCS Manager. Устаткування – це, як правило, задіяна в цей момент фізична основа UCS. Серверами в загальному випадку йменуються логічні сервери, створені й використовувані за допомогою UCS Manager. У зону ЛОМ включається все, що ставиться до локальних мереж, а в зону мережі зберігання – усе, що стосується ресурсів зберігання.

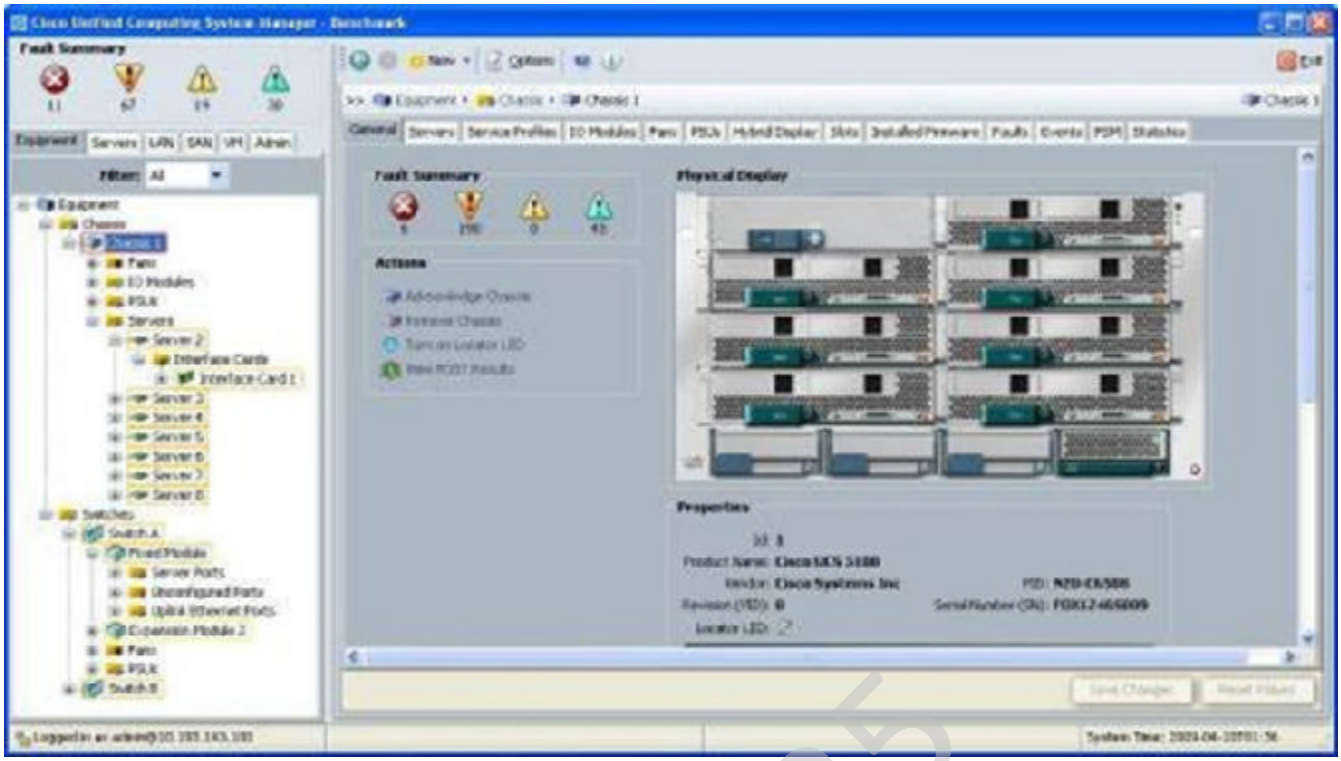

Рисунок 2.1 – Інтерфейс користувача UCS manager

Переваги United Computing System (UCS)

Убудоване керування системою

Кожний з компонентів об'єднаної системи обчислень Cisco UCS поставляється з убудованим мікропрограмним забезпеченням, що дозволяє управляти роботою пристрою за допомогою Cisco UCS Manager. Адміністратори мережі, системи зберігання даних і серверів можуть працювати із графічним користувальницьким інтерфейсом або інтерфейсом командного рядка системи Cisco UCS Manager або, використовуючи документований набір функцій XML API, з існуючої корпоративної системи керування ЦОД.

Рольова модель контролю спрощує рішення завдань керування, до яких повинні залучатися групи адміністраторів серверів, мережі й системи зберігання даних, дозволяючи обмежити область поширення спеціальної інформації рамками кожної групи. Це дозволять експертам по конкретних питаннях випливати звичайним робочим процедурам і інтегрувати всі дані про

конфігурацію в рамках єдиної системи керування, а не в розрізнених системах, як це нерідко відбувається в сучасних ЦОД.

Впровадження додатків з використанням сервісних профілів

ПЗ UCS Manager реалізує концепцію керування на базі ролей і політик з використанням сервісних профілів і шаблонів. Інформація про параметри системи електроживлення, охолодження, фізичної безпеки, а також про стан устаткування, конфігурації мережного середовища й мережі зберігання даних утримується в сервісному профілі. Використання сервісних профілів дозволяє IT-персоналу ЦОД знизити час впровадження додатків у ЦОД від днів до хвилин.

Об'єднаний транспорт

Розроблена Cisco Systems технологія консолідованої мережі (United fabric) на базі наборів стандартів Data Center Bridging і Fibre Channel over Ethernet (FCoS) дозволяє значно знизити витрати на елементи мережної інфраструктури (безліч мережних адаптерів, комутатори LAN, SAN, різні мережні кабелі й т.п.). Мережні модулі шасі дозволяють відмовитися від використання комутаторів у складі блейд-серверів шляхом транзиту всього трафіку від серверів через централізовану фабрику комутації, де трафік буде оброблятися й комутируватися по призначенню. Уніфікована фабрика комутації будується на базі технології 10-Gigabit Ethernet зі стандартними кабельними з'єднаннями. Тепер у випадку зміни типу підключення сервера до мережі немає необхідності в установці додаткових адаптерів і прокладці нових кабелів.

Підтримка технології віртуалізації (VN-Link)

Технологія Cisco VN-Link розширює границю мережі до віртуальної машини. Ця технологія стирає розходження по керуванню мережною інфраструктурою для фізичних і віртуальних серверів. Тепер всі мережні з'єднання налаштовуються й управляються централізовано, без виділення додаткового рівня комутації для віртуальних середовищ. Конфігурації портів уведення/виводу й мережних політиків можуть переміщаються між віртуальними серверами, що збільшує ефективність і зменшує складність їхньої експлуатації.

Віртуалізированый адаптер Cisco VIC

Віртуалізированый адаптер Cisco VIC дозволяє одержати на одному двопортовому конвергентному адаптері динамічно або статично до 58 віртуальних адаптерів, кожний їх яким представлений PCIe-Функцією. У такий спосіб операційна система «бачить» кожний з віртуальних адаптерів як фізичний адаптер. Даним віртуальним адаптерам можна гарантувати смугу пропущення; невикористовувана в сучасний момент частина смуги пропущення будь-якого віртуального адаптера може бути динамічно розподілена між іншими віртуальними адаптерами, яким у той же момент потрібна більша смуга, чим їм гарантував системний адміністратор.

Технологія розширення пам'яті Cisco

Технологія розширення пам'яті Cisco дозволяє збільшити в 4 рази кількість рознімань для установки модулів пам'яті DIMM (до 96) у порівнянні із класичними двопроцессорними серверами архітектури x86, при збереженні швидкості частоти 1866Мгц. Збільшення оперативної пам'яті дозволяє збільшити продуктивність роботи серверів, особливо при роботі у віртуальних середовищах.

Сучасна продуктивність

У рішенні Cisco UCS використовуються блейд-сервера, побудовані на базі процесорів серії Intel Xeon E5 v2 і E7 v2. Ці багатоядерні процесори інтелектуально й автоматично регулюють продуктивність серверів відповідно до вимог додатків, збільшують продуктивність у необхідний момент і істотно заощаджують енергоспоживання в період простою. Для більше точного керування серверами всі параметри продуктивності й економії електроенергії можуть бути настроєні вручну.

Енергетична ефективність

Компоненти рішення Cisco UCS були спроектовані з урахуванням вимог по енергетичній ефективності. Спрощена архітектура системи дозволила скоротити кількість елементів, для яких необхідне електроживлення й охолодження зразкове на 50% у порівнянні із класичними середовищами блейд-

серверів. Шасі блейд-серверів у рішенні Cisco UCS зроблено таким чином, щоб значно збільшити теплообмін з навколишнім середовищем. Рішення Cisco UCS використовує більше ефективні, чим раніше, блоки живлення для своїх компонентів.

Гарантія й сервісні контракти

Компанія Cisco Systems на всі продукти UCS крім стандартної гарантії, за бажанням партнерів, надає сервісні контракти.

United Computing Warranty (3-year) – безкоштовна стандартна гарантія строком на 3 роки, містить у собі:

– Завантаження прошивань BIOS і драйверів.

– Авансова заміна устаткування строком NBD (next business day).

– 90 денна гарантія на програмний носій.

– Цілодобовий доступ у службу технічної підтримки Cisco TAC для оформлення RMA.

United Computing Warranty Plus – платне розширення гарантії строком на 1 рік (за замовчуванням), включає:

– Усе вище перераховане.

– Більше гнучкий час реакції на заміну устаткування.

– Можливість заміни устаткування на площадці в замовника.

United Computing Support Service – платний сервісний контракт строком на 1 рік (за замовчуванням), включає:

– Усе вище перераховане.

– Цілодобова підтримка в TAC на ПЗ (всі ОС, VMWare, BMC BladeLogic).

– Поділ зони відповідальності між Cisco і виробником ПЗ.

– Можливість проактивної діагностики устаткування.

– Цілодобовий доступ у службу Cisco TAC з будь-яких питань.

– Відновлення UCS Manager.

IBM PureApplication System

– IBM PureApplication System – це готова система хмарних обчислень із убудованим апаратним і програмним забезпеченням, призначена для розгортання й рішення завдань у хмарі – усе, що потрібно корпоративному центру обробки даних для додавання середовища приватної хмари. Хоча система (по більшій частині) автономна, її потрібно підключити до мережі ЦОД, щоб зовнішні користувачі могли підключатися до системи й управляти нею, а також розгортати свої додатки, підключатися до цих додатків і використовувати їх.

При цьому виникають наступні питання:

– Як приєднати систему до мережі?

– Як підключати до мережі додатка?

– Як виконати деякі спеціальні вимоги по підключенню до мережі?

Щоб використовувати систему PureApplication System, потрібно підключити її до мережі центра обробки даних, де вона встановлена. Мережні з'єднання дозволяють встановлювати зв'язок із системами, щоб ними можна було управляти й щоб клієнти за межами системи могли підключатися до розгорнутого в цій системі додаткам. Перш ніж виконати логічне налаштовування мережі центра обробки даних для підключення до системи, необхідно фізично підключити її до мережі ЦОД. Щоб продемонструвати, як підключити систему до мережі, ми спочатку покажемо, як улаштовані з'єднання усередині системи, потім скористаємося цією архітектурою як приклад оптимальних зовнішніх з'єднань.

Внутрішня мережа

PureApplication System призначена для забезпечення відказостійкості, так щоб вся система продовжувала функціонувати навіть у випадку відмови важливого компонента. Щоб уникнути єдиної крапки відмови кожний важливий компонент апаратного й програмного забезпечення реалізована як надлишкова пара компонентів.

Як і інша частина системи, її внутрішнє мережне устаткування націлене на відказостійкість із використанням надлишкових компонентів. Схема мережного устаткування являє собою деревоподібну структуру з парою головних комутаторів (Top of Rack – TOR) у корені дерева. Комутатори TOR, кожний з яких – це комутатор IBM System Networking RackSwitch™ G8264 з 64 портами й пропускною здатністю 10 Гбіт/с, – з'єднує корпуси системи один з одним і з мережею центра обробки даних. Комутатори TOR виконують функції точки підключення локальної мережі між системою й центром обробки даних.

TOR з'єднані один з одним і діють як один комутатор, розподіляючи свій трафік пакетів. Вони з'єднані чотирма кабелями 40 Гбіт/с. Комутатори використовують це з'єднання для налаштовування мережі Inter Switch Link (ISL), що поєднує два комутатори так, що вони поводяться як один.

– Галузі дерева – це внутрішні комутатори. У кожному корпусі перебувають два комутатори Ethernet – IBM Flex System Fabric EN4093R 10Gb Scalable Switch з 66 портами кожний, – які з'єднують комутатори TOR із внутрішніми обчислювальними вузлами. Усередині корпуса комутатори з'єднані вісьма кабелями 10 Гбіт/с. Як і комутатори TOR, пари внутрішніх комутаторів діє як один комутатор, використовуючи з'єднання між ними для налаштовування групи Virtual Link Aggregation Group (vLAG), що дозволяє двом комутаторам діяти як один.

– З'єднання між комутаторами TOR і кожною парою внутрішніх комутаторів також розраховано на забезпечення надмірності й відказостійкості. Кожний TOR з'єднаний з кожним внутрішнім комутатором чотирма кабелями 10 Гбіт/с, так що чотири комплекти з'єднань містять у цілому шістнадцять кабелів. Комутатори використовують протокол Link Aggregation Control Protocol (LACP) для об'єднання портів у загальний канал, що підсумує пропускну здатність кабелів. Шістнадцять кабелів між комутаторами TOR і парою комутаторів у корпусі агрегируются за допомогою LACP, який створює магістральну лінію із пропускною здатністю 160 Гбіт/с.

Ізоляція мереж

– PureApplication System віртуалізує свої мережні зв'язки у двох логічних сегментах мережі; ізольовані широкомовні домени, створені в рамках фізичного зв'язку, називається віртуальними локальними мережами (VLAN). Система використовує ці VLAN для логічної ізоляції мережного трафіку додатків від трафіку керування системою й логічної ізоляції додатків, що працюють у мережі, друг від друга. Ці логічні VLAN повинні відповідати мережному встаткуванню системи.

Як пояснювалося вище, зв'язок між комутаторами TOR і парою корпусних комутаторів реалізується як LACP-агрегований канал, у якому шістнадцять кабелів по 10 Гбіт/с діють як один кабель 160 Гбіт/с. Цей канал настроєний у комутаторах як магістраль (trunk – протокол, що дозволяє узагальнювати фізичні лінії зв'язку, пропускаючи пакети з різних VLAN) яка пропускає трафік декількох VLAN, зберігаючи пакети кожної з VLAN ізольованими від інших. Система розподіляє мережі VLAN додатків між мережним устаткуванням, зв'язуючи їх за допомогою цих магістральних каналів; всі VLAN входять у конфігурацію каналу.

Це робить їх надлишковими й відказостійкими. Комутатори можуть передавати пакети будь-який конкретної VLAN по кожному з кабелів, пов'язаних з магістраллю, і розподіляти трафік між кабелями. Це не тільки оптимізує пропускну здатність, але й дозволяє зберегти трафік VLAN у магістралі навіть у випадку відмови устаткування.

От кілька можливих сценаріїв відмови устаткування без переривання трафіку.

– У випадку відмови одного з кабелів магістралі інші продовжують переносити трафік між комутаторами. Тому що всі кабелі обслуговують всі VLAN, зв'язок з VLAN, підключеними до кабелю, що відмовив, не перерветься, вона просто перейде на інші кабелі.

– При відмові одного з комутаторів магістралі кабелі цього комутатора перестануть працювати, але весь їх трафік піде по кабелях другого комутатора, так що зв'язок збережеться до відновлення комутатора, що відмовив.

– Навіть якщо одночасно відмовлять два комутатори – по одному на кожному кінці магістралі (TOR і внутрішній комутатор), – і три чверті кабелів перестануть працювати, трафік всіх VLAN перейде в кабелі, що залишилися, і зв'язок збережеться до відновлення комутаторів.

Внутрішня мережа керування

При виборі ідентифікаторів VLAN для приєднання до мережі ЦОД майте на увазі, що в PureApplication System є внутрішня мережа керування, за якої зарезервовані ідентифікатори VLAN. VLAN, що підключаються до мережі ЦОД, не повинні використовувати ці зарезервовані ідентифікатори.

Система не призначає ці VLAN портам, використовуваним для підключення до мережі ЦОД, тому їх трафік не виходить за межі системи. Для надійності рекомендується зарезервувати ці ідентифікатори VLAN у мережі ЦОД, щоб вузли мережі не намагалися їх використовувати.

Підключення до мережі ЦОД

Ми показали, як конфігурація мережного устаткування й сегменти логічної мережі PureApplication System виключають єдину крапку відмови. Давайте візьмемо конкретний приклад і використовуємо його для проектування з'єднання між системою й мережею ЦОД, так щоб це з'єднання було таким же відказостійким, як внутрішня мережа системи.

Кожне з'єднання між комутаторами TOR і мережею ЦОД повинне бути надлишковим, як у внутрішній мережі PureApplication System. Для цього мережа ЦОД повинна з'єднуватися із системою за допомогою не одного, а двох комутаторів. Ці два комутатори з'єднані з використанням методу агрегування, такого як vPC, vSS, Stacking, ISL або vLAG. Наявність двох агрегованих комутаторів забезпечує безперервну готовність навіть при відмові одного з компонентів (комутатора, трансівера, кабелю й т.п.).

Зв'язок у системі PureApplication реалізується двома окремими мережами:

– мережею керування системою – ця мережа надає клієнтам доступ до функцій керування системою: консолі із графічним інтерфейсом користувача, інтерфейсу командного рядка (CLI) і інтерфейсу REST. Ця мережа також використовується для резервного копіювання й відновлення системи і її компонентів і для керування декількома системами;

– робоча мережа – ця мережа надає клієнтам доступ до додатків, розгорнутим у системі, а додаткам – доступ до корпоративних ресурсів.

Спочатку розглянемо, як настроїти мережа керування системою.

Зовнішні з'єднання повинні бути настроєні багато в чому як і внутрішні з'єднання між комутаторами TOR і парою корпусних комутаторів. Тобто з'єднання між комутаторами TOR і агрегований парою комутаторів ЦОД повинне являти собою агреговану магістраль, що проходить по двох кабелях (LACP або Etherchannel). Два кабелі підвищують пропускну здатність з'єднання, а магістраль створює надмірність у рамках з'єднання на випадок відмови одного кабелю. Лінія для мережі керування системою проходить окремо від лінії для робочої мережі, щоб фізично ізолювати підмережі, так щоб можна було налагоджувати, контролювати й підтримувати лінії мережі передачі даних без втрати мережного з'єднання системного керування. Тому що є тільки одна VLAN – VLAN керування системою, що проходить через цю лінію, – лінія настроєна за принципом портів доступу – таким чином, щоб тип пакетів, переданих по цій лінії, обмежувався тільки пакетами цієї VLAN.

Тепер давайте подивимося, як настроїти робочу мережу.

Ця мережа надає додаткам, що працюють у системі, доступ до ресурсів мережі ЦОД для наступних цілей:

– У додатків повинна бути можливість підключатися до ресурсів ЦОД, таким як база дані реєстрації подій.

– Клієнтським додаткам, що працюють за межами системи, таким як веб-браузер, потрібно підключитися до додатка, що працює на серверах системи.

Як і лінія мережі керування системою, лінія зв'язку між комутаторами TOR і агрегованою парою комутаторів ЦОД повинна являти собою агрегований канал. Для цього каналу використовуються чотири кабелі, об'єднані в одну магістраль – по одному від кожного комутатора TOR до кожного комутатора ЦОД. Це забезпечує вчетверо більшу фізичну пропускну здатність одного кабелю й означає, що навіть якщо комутатор TOR і один комутатор ЦОД відмовлять одночасно, у лінії однаково залишиться один діючий кабель. (Якщо вийде з ладу один комутатор ЦОД або один комутатор TOR, а не обоє одночасно, те з'єднання буде продовжувати роботу із двох кабелів.) Фізично це з'єднання ізольоване від мережі керування системи. Оскільки робоча мережа складається з декількох мереж VLAN, з'єднання налаштовується за принципом магістрального порту.

Система містить конфігурацію мережі, що вона використовує для налаштовування комутаторів TOR з VLAN і ліній з'єднання системи з мережею ЦОД. Ця конфігурація повинна відповідати тої, котра буде застосовуватися до комутаторів ЦОД.

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 першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи передачі даних між дата-центрами на основі технології Web Scale.

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

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

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

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

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

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

Всі сучасні додатки й послуги, від Інтернету речей і мобільних додатків до послуг OTT, впливають як на поводження кінцевих користувачів, так і на мережу ЦОДа. Однак концепція побудови й експлуатації мереж (і пропонованих на їхній базі послуг) змінилася не тільки із цієї причини. Віртуалізація поширюється із ЦОДів і IT-інфраструктур на мережі. Нові підходи, такі як програмно обумовлені мережі (SDN) і віртуалізація мережних функцій (NFV), міняють динаміку розвитку галузі.

У телекомунікаційній галузі NFV по праву розглядається як наступний етап на шляху до створення більше гнучкої й економічної мережної інфраструктури. NFV – це спосіб знизити витрати й прискорити розгортання послуг мережних операторів шляхом відділення систем мережного захисту й маршрутизації від спеціалізованого устаткування й переміщення їх на віртуальні сервери.

Замість установки дорогого пропрієтарного устаткування постачальники послуг можуть придбати недорогі комутатори, пристрої зберігання й сервери, на яких будуть розміщатися віртуальні машини, що виконують мережні функції. Це забезпечить виконання відразу декількох завдань на одному фізичному сервері, що буде сприяти зниженню витрат і зведе до мінімуму кількість виїздів фахівців. Якщо клієнтові буде потрібно нова мережна функція, постачальник послуг зможе додати її, запустивши нову віртуальну машину. Віртуалізація мережних функцій знижує залежність постачальників послуг від спеціалізованих апаратних пристроїв, а також забезпечує можливість ефективного масштабування й налаштовування в масштабі всієї мережі.

Технологічні інновації

Завдяки недавнім технологічним проривам в області апаратного й програмного забезпечення вдалося перебороти наступні пов'язані з DCI проблеми.

Усунення дистанційних обмежень за рахунок цифрової обробки сигналу (DSP): недоліки оптичного волокна, такі як хроматична й поляризаційна дисперсія, які раніше перешкоджали реалізації з'єднань із високою пропускною здатністю на далеких відстанях, тепер не представляють проблеми. Нові досягнення в області технології DSP дозволили постачальникам мережного устаткування розробити пакетні оптичні платформи, здатні автоматично компенсувати ці недоліки. Великі потоки даних можна передавати на кілька тисяч кілометрів по волокну різного типу без шкоди для швидкості й продуктивності. У сучасних оптичних інтерфейсах програмуються оптимальні схеми модуляції для різних сценаріїв розгортання.

Усунення обмежень по ємності за рахунок когерентної оптики. Завдяки появі когерентної оптики створений фундамент для успішної передачі даних на швидкості до 17,6 Тбіт/с і вище практично на будь-яку відстань. Когерентне детектування значно збільшило пропускну здатність, що є ключовою умовою успішного використання сучасних рішень DCI. *Зниження затримок за рахунок високопродуктивної й ультрашвидкісної оптоелектроніки.* Складна структура устаткування, оптимізовані програмні механізми, інноваційні схеми прямого виправлення помилок (FEC) і високопродуктивна оптоелектроніка значно знизили затримки, обумовлені характеристиками мережного устаткування. *Перехід від виконуваних вручну операцій до програмувальної автоматизації.* Мережі ЦОДів постійно міняються, тому прогнозування тенденцій трафіку ускладнюється. Виконання операційних завдань можна автоматизувати за допомогою API і відповідних додатків. Створювані самими користувачами додатка здатні самостійно відправляти запит на збільшення пропускної здатності, установлювати нові з'єднання між двома кінцевими крапками, міняти існуючі

підключення, а також виконувати безліч інших необхідних повсякденних операцій без якого-небудь втручання з боку людини.

Скорочення витрат за рахунок використання платформ, оптимізованих для додатків. При розробці новітніх оптичних платформ особлива увага приділяється підтримці завдань DCI. Спрощені процеси планування, замовлення й установки дозволяють швидше реалізувати міжз'єднання. Завдяки повної програмуємості можна проектувати й розробляти додатка для рішення конкретних завдань. Висока швидкість із мінімальними вимогами до площі забезпечує мінімальну вартість передачі на біт при з'єднанні ЦОДів. За рахунок компактності конструкції й низького енергоспоживання знижуються експлуатаційні витрати, а модульна структура забезпечує масштабування до декількох терабітів транспортної ємності без значного збільшення капітальних вкладень і поточних витрат. Сьогодні ми можемо розгортати гнучкі, економічно ефективні рішення DCI, що гарантують ефективність і високу швидкість виконання завдань. Значна ємність і ефективна масштабованість дозволяють повною мірою задовольнити зростаючі вимоги відносно обробки більших обсягів трафіку й підготуватися до роботи в більше складних умовах без значних капітальних витрат і переривання обслуговування для нарощування ємності.

Можливості миру Web Scale

Технологічні інновації дозволяють мережним операторам і операторам ЦОДів прискорити розгортання, знизити експлуатаційні витрати, збільшити рівень гнучкості й ефективності. Для успішної реалізації DCI необхідні відповідні платформи, спроектовані спеціально для масштабування додатків DCI до рівня Web Scale, що підтримують взаємодію із хмарними середовищами й іншими ЦОДами. Вимоги нової операційної парадигми Web Scale можна задовольнити тільки при впровадженні рішень DCI, споконвічно орієнтованих на реалізацію операцій Web Scale, таких як Ciena Waveserver, що містить нові функції й інструменти, що допомагають додаткам DCI забезпечити глобальний перехід до з'єднань Web Scale.

3.2 Розробка структурної схеми

Центри обробки даних (ЦОД) або дата-центри забезпечують безперервну роботу критичних бізнес-додатків, а також зниження вартості зберігання й обробки інформації шляхом більше ефективного використання систем зберігання даних і консолідації обчислювальних потужностей. Створення ЦОД оптимізує також витрати на експлуатацію займаних обчислювальним устаткуванням приміщень і на обслуговуючий персонал.

Центр обробки даних – це відказостійкая комплексна централізована система, що покликана забезпечувати безперебійну автоматизовану роботу бізнес-процесів з високим рівнем продуктивності і якістю надаваних сервісів.

ЦОД являє собою окреме спорудження (площадку), обладнане відповідними інженерними системами кондиціювання, пожежогасіння, безперебійного електроживлення та ін., у якому розміщаються обчислювальні ресурси (сервери, системи зберігання й т.д.), створена телекомунікаційна інфраструктура, що забезпечує віддалений доступ до ресурсів ЦОД, створена система інформаційної безпеки та ін.

Визначення поняття "ЦОД" є в базовому стандарті на вимоги до інфраструктури – ANSI/EIA/ TIA-942 (Telecommunications Infrastructure Standard for Data Centers), затвердженому в 2005 році в США. У ньому все дата-центри діляться на чотири рівні (класу). Існує також стандарт дослідницької організації Uptime Institute, у якому дата-центри теж розподіляються на чотири класи.

Зразковий набір загальних вимог до дата-центру включає:

– Високий гарантуємий рівень доступності виділеного каналу зв'язку й інтернет з'єднання.

– Гарантована швидкість відновлення працездатності при аваріях устаткування й збоях ПЗ.

– Наявність системи резервного копіювання, що відповідає вимогам за рівнем схоронності даних і швидкості їхнього відновлення.

– Наявність системи моніторингу апаратно-програмних засобів (включаючи налаштовування, діагностування й оперативний контроль).

– Цілодобова кваліфікована технічна підтримка устаткування й систем.

– Наявність засобів виявлення й протидії мережним вторгненням.

– Наявність системи захисту від несанкціонованого доступу у внутрішні приміщення ЦОД.

– Охорона зовнішнього периметра.

– Наявність системи енергоживлення з резервуванням енерговводa, наявність джерел безперебійного живлення й дизель-генератора.

– Електростатичний захист усього устаткування й пола.

– Наявність резервованої системи кондиціювання й вентиляції.

– Наявність системи виявлення загоряння й газового пожежогасіння.

Таким чином, вимоги пред'являються не тільки до інженерної інфраструктури, але й до організації ефективного керування IT-ресурсами, захисту систем, до каналів зв'язку, які повинні бути дубльовані й забезпечувати доступність даних і додатків. Всі ці вимоги спрямовані на досягнення високого рівня надійності й доступності корпоративних інформаційних систем, схоронності даних, підвищення керованості IT-інфраструктури.

Послуги

Клієнтам дата-центрів звичайно надається наступний типовий набір послуг:

Colocation (розміщення устаткування) – клієнт використовує технічні ресурси дата-центра для розміщення своїх серверів, систем зберігання даних, телекомунікаційного й іншого устаткування. Клієнт здійснює підтримку своїх ресурсів самостійно, а установку устаткування в стійки й підключення роблять фахівці дати-центра. Дана послуга рятує орендарів або власників офісних площ від необхідності нарощування власних IT-потужностей: установки серверного устаткування, організації локальної мережі й зовнішніх каналів зв'язку й проведення інших робіт, а також від змісту кваліфікованого обслуговуючого

персоналу. У цьому випадку всі питання, пов'язані з експлуатацією корпоративної інформаційної системи, клієнт віддає на аутсорсинг спеціалізованим компаніям, розміщаючи свої сервери на технічних площадках оператора дати-центра. Постачальники послуги, у свою чергу, гарантують постійну, надійну й безперебійну роботу інформаційної системи, безпека збережених даних, швидкісний Інтернет та інші необхідні сервіси.

Dedicated server (виділений сервер) – надання серверів в оренду. Клієнт може орендувати сервер у власника дати-центра. Послуга широко поширена серед клієнтів, які прагнуть уникнути великих разових вкладень в устаткування на початковому етапі розвитку IT-системи. Цей сервіс дозволяє на етапі становлення компанії не ризикувати інвестиціями, необхідними для створення власної IT-інфраструктури.

Telehousing – послуги з розміщення телекомунікаційної й IT-інфраструктури замовника. Оператори зв'язку можуть розмістити свій вузол зв'язку в дата-центрі, установлюючи устаткування повністю або частково, організувати підключення мережі до мереж інших операторів. Особливістю даної послуги є розміщення специфічного телекомунікаційного устаткування й наявність великої кількості прямих міжоператорських стиків і зв'язків.

Shared – оренда дискового простору на встаткуванні власника дата-центра із заздалегідь погодженою платформою й конфігурацією операційної системи. Дата-Центр забезпечує підтримку (технічних і програмну) додатків клієнта, наприклад підтримку скриптів. Корпоративні клієнти можуть використовувати послуги для повного або часткового розміщення устаткування, систем і інфраструктури. Також можлива організація підключення устаткування замовника до мереж операторів зв'язку.

На базі дата-центрів можуть надаватися й більше складні послуги, такі, наприклад, як:

а) **аутсорсинг інформаційних систем** – послуга, надавана за довгостроковою згодою, по якому постачальник послуг одержує в повне

керування або у власність всю IT-інфраструктуру клієнта або значну її частину, включаючи устаткування й установлене на ньому програмне забезпечення. Це проекти із залученням у роботу виконавця, які припускають відповідальність за системи, мережу й окремі додатки, що входять в IT-інфраструктуру. IT-аутсорсинг оформляється у вигляді довгострокового контракту;

б) **хостинг обслуговування й адміністрування ПЗ** – має на увазі централізоване керування тиражуємим програмним забезпеченням за умови, що додатки перебувають на території постачальника послуг, а замовник має віддалений доступ до ПЗ. Основні розходження між хостингом обслуговування й адмініструванням ПЗ й традиційним керуванням додатками полягають у наступному:

– додатка замовника працюють, перебуваючи в інформаційному центрі постачальника послуг;

– інформаційний центр постачальника послуг працює за схемою "один – до багатьох";

– постачальник послуг не несе відповідальності за апаратні засоби замовника.

в) **хостинг інфраструктурних послуг** – це сервіс по наданню стандартних елементів IT-інфраструктури у віддалене користування на певний період. Має на увазі керування серверами й мережними рішеннями в інформаційному центрі іншої компанії для керування виділеним або поділюваним веб-доступом, підтримки електронної комерції, керування змістом і системами безпеки. Включає послуги з розгортання системи з її наступним віддаленим керуванням.

Показники рівня надійності ЦОД (Tiers)

Ідея визначити рівні надійності дата-центра народилася в надрах асоціацій підприємств, зібраної під прапором Uptime Institute (UI), що займається збором і обробкою інформації, видачею рекомендацій і детальних вимог до рівнів надійності центрів обробки даних.

Розроблена кілька років назад компанією Uptime Institute система стандартів на базі рівнів відказостійкості Tier знайшла відбиття в стандарті TIA/ EIA-942 і широко використовується в усьому світі. Класифікація Tier ураховує дрібні деталі: відповідно до неї рівень надійності ЦОД визначається топологією, потужністю, схемами резервування інфраструктури й безліччю інших факторів.

Стандарт визначає чотири рівні відказостійкості ЦОД – чим вище рівень, тим вище вимоги. Доступність рівня Tier I, концепція якого була розроблена в 60-е роки, становить 99,671%; це означає, що запланований час простою не перевищує 28,8 години на рік. Рівень Tier IV, що з'явився в 90-е роки, – 99,995%; регламентні роботи можуть проводитися без відключення устаткування, а припустима тривалість зупинки – 0,4 години на рік. На відміну від першого рівня, четвертий припускає повне резервування, у тому числі фідерів електроживлення й вхідних магістралей від провайдерів послуг зв'язку. Мова йде не тільки про можливість проведення запланованих профілактичних робіт без припинення роботи площадки. Об'єкт може витримати й незаплановані інциденти, операційні помилки, вихід з ладу компонента або перебій у живленні без нанесення втрати IT. Архітектура Tier IV дозволяє автоматично нейтралізувати наслідку збоїти й стримувати його подальший вплив. Вартість ЦОД четвертого рівня надзвичайно висока, що обумовлено критичністю розв'язуваних завдань. Нерідко четвертий рівень доступності забезпечується шляхом створення резервного ЦОД.

Для зниження ризиків в Uptime Institute було уведено таке поняття, як операційна стійкість – це можливість сайту протягом усього строку життя надавати плановані результати в абсолютно всіх можливих умовах роботи. Цей параметр складається, у тому числі з якості менеджменту й рівня підготовки обслуговуючого персоналу. Площадки, де дуже добре поставлений менеджмент, будуть набагато більше продуктивні й надійні, чим сайти з більше сильним дизайн-компонентом, але з погано організованою експлуатацією, ненавченим персоналом.

Таблиця 3.1 – Рівні відказостійкості ЦОД

Параметр/Клас ЦОД (рівень)	Tier I	Tier II	Tier III	Tier IV
Тип будинку	З сусідами	З сусідами	Окремо варте	Окремо варте
Кількість енергопроводів	1	1	Один активний, другий резервний	Два активних
Схема резервування компонентів	N	N+1	N+1	2 (N+1) або S+S
Первісна потужність із розрахунку Вт на $м^2$	215-323	430-537	430-645	537-860
Максимальна потужність Вт на $м^2$	215-323	430-537	1075-1615	1615+
Безперебійне кондиціювання	Немає	Немає	Можливо	Є
Нормативне навантаження на фальшпол, кг на кв.м.	415	488	732	732+ (по стандарті 2005 р. – 1000+)
Загальна тривалість відмов за рік, годин	28,8	22	1,6	0,4
Доступність ЦОД	99,671%	99,749%	99,982%	99,995%
Обслуговування	ЦОД повинен повністю зупинятися для регламентних робіт	Допускаються перерви в роботі на технічне обслуговування й ремонт	Цілодобова зміна по буднях, у вихідні по виклику	Цілодобова чергова зміна

Важливу роль у зниженні ризиків грає можливість ізоляції. Вона означає рознесення інформаційних каналів і контурів подачі живлення. Основний і резервний елемент ізолюються, розносяться фізично, завдяки чому одиничний інцидент не нанесе втрати всьому комплексу ЦОД. Існує також можливість резервування надлишкових компонентів, з використанням множинних каналів розподілу: якщо на одній лінії відбувається збій, устаткування може перемкнутися на іншу ділянку живлення або кондиціювання.

Варто підкреслити, що стандарт TIA/ EIA-942 і стандарти, пропоновані Uptime Institute, мають принципово різний підхід до проблематики, хоча рівні Tier у стандарті 942 запозичені з ранніх публікацій Uptime. Представники Uptime іронізують, що дуже влещено, але стандарт Uptime все-таки не має нічого загального з TIA 942.

Якщо говорити про фактичну різницю, то стандарт 942 – це набір практичних рекомендацій, досить об'ємний (150-сторінковий) документ, що детально описує, як розвертати інженерну інфраструктуру, із чого вона повинна складатися, з конкретним перерахуванням систем, вимог, із числовими викладеннями.

Стандарт UI – це класифікація, що легко зрозуміти менеджерам, що приймають фінансові рішення про інвестиції. Це більше глобальний погляд, що не опускається до деталей. У цьому стандарті немає переліку систем; основна увага приділена топології й експлуатаційним перевагам, які ця топологія забезпечує. Наприклад, топологія сайту Tier III повинна допускати обслуговування будь-якого елемента інфраструктури на лету, без шкоди для устаткування.

Глобальні вимоги до площадки, для того щоб вона відповідала Tier III або Tier IV, зовсім проста, і їх небагато. Але вони дуже ефективні й зрозумілі менеджменту, що приймає рішення про будівництво велика, дорогих дата-центрів, які будуть експлуатуватися десятки років. Саме для менеджменту (який приймає рішення про інвестиції й піклується, про те, щоб вкладені кошти

працювали ефективно, і через п'ять років не довелося вкладати ще стільки ж на переробки або модернізацію) і розроблена така проста класифікація.

По даним Uptime Institute, кількість дата-центрів у Росії, сертифікованих Tier, становить 14. По цьому показнику нашу країну обганяють тільки 4 інших – Індія (16 сертифікованих ЦОДів), Канада (19 сертифікованих ЦОДів), Бразилія (23 сертифікованих ЦОДа) і США (67 сертифікованих ЦОДа).

Можливо, ви вже чули досить новий для ринку не_онлайн проектів термін – Web-Scale IT, що на думку Gartner в 2018 році займе не менш 50% ринку корпоративного IT.

Web-Scale

Для початку, варто спробувати взагалі визначитися, що мається на увазі під цим настільки одіозним терміном, причому бажано без маркетингового марнослів'я.

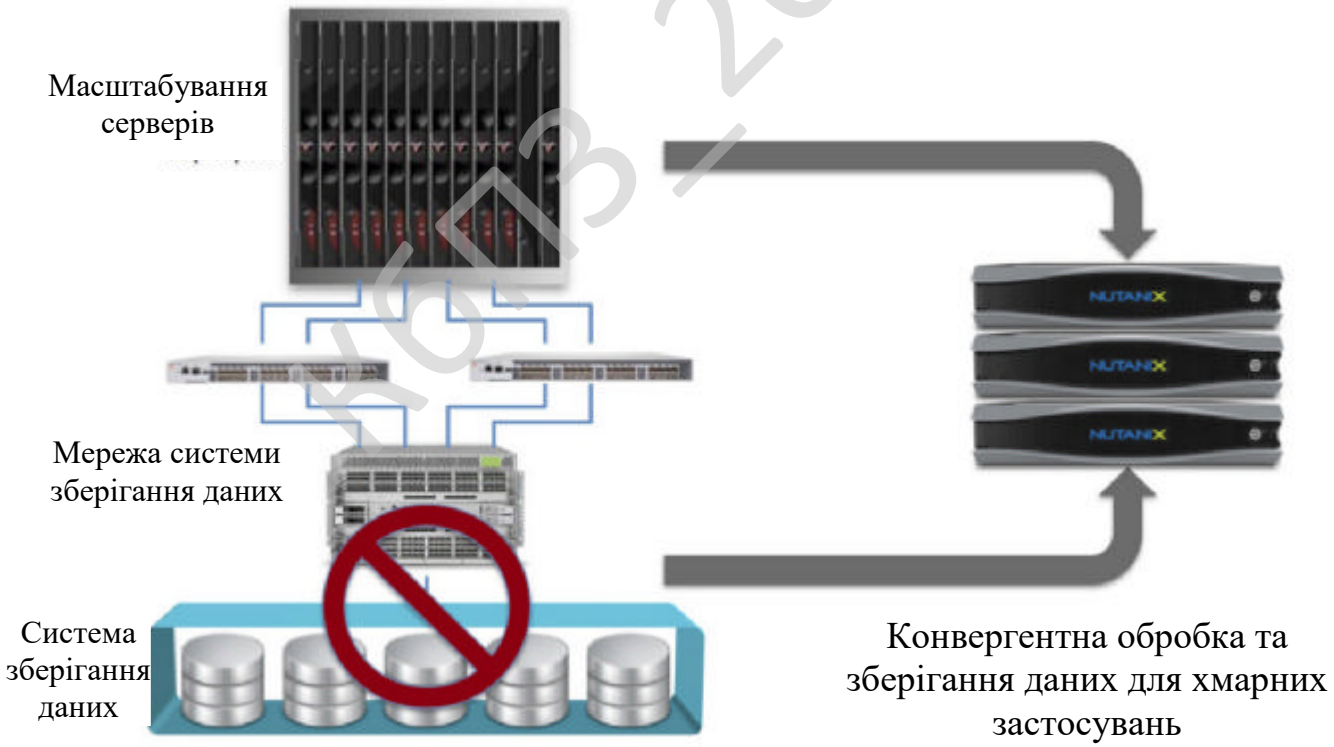

Рисунок 3.1 – Структурна схема системи

Базові принципи:

– Гіпер-конвергентність.

– Чисто програмна реалізація (Software Defined).

– Розподілені й повністю самодостатні системи.

– Лінійне розширення з дуже точною гранулярністю.

Додатково:

– Потужна автоматизація й аналітика за допомогою API.

– Самолікування після відмов устаткування й дата-центрів (катастрофостійкість).

Гіпер-конвергентність

У спрощеному змісті – нативне об'єднання двох або більше різних компонентів (мережа, віртуалізація й т.д.) в один юніт.

Нативність – ключове слово в цьому випадку, тому що ми не говоримо просто про впакування різних компонентів у єдиному пакеті, але маємо через повну й споконвічну інтеграцію.

Нативна інтеграція двох або більше компонентів у випадку Web-scale дає лінійне (горизонтальне) масштабування без яких-небудь лімітів.

Як наслідок, ми одержуємо істотні переваги:

– Масштабування по одній одиниці рішення (як кубики Лего).

– Локалізація уведення-виводу (надто важливо для масивного прискорення операцій I/O).

– Усунення традиційних СЗД як рудиментарних, з інтеграцією в єдине рішення (немає СЗД – немає проблем з нею зв'язаних).

– Підтримка всіх основних технологій віртуалізації на ринку, включаючи open-source (ESXi, Hyper, KVM).

Чисто програмна реалізація

Перенос всієї логіки з (украй складного) пропієтарного устаткування (спец-процесори, ASIC / FPGA) в 100% програмну реалізацію.

Як приклад, в Nutanix ми виконуємо будь-які (і найчастіше унікальні – такі як дедуплікація RAM кеша, розподілена map/reduce відкладена дедуплікація даних на кластері й т.д.) операції програмно.

Багато хто запитають – це ж повинне дико “гальмувати”? Жодним чином.

Сучасні процесори Intel уміють дуже багато чого, причому дуже швидко.

Як приклад – замість застосування апаратних спец. адаптерів для компресії й дедуплікації (як роблять деякі архаїчні вендори) – ми просто використовуємо апаратні інструкції процесора Intel для підрахунку sha1 контрольних сум.

Компресія даних (відкладена й “на льоту”)

Легко, вільний алгоритм snappy, використовуваний Google.

Резервування даних

Проблема RAID навіть не тільки швидкість його роботи, але час відновлення після відмов апаратних вузлів (дисків, полиць, контролерів). Наприклад, у випадку з Nutanix, після відмови жорсткого диска на 4TB, відновлення цілісності системи (кількості реплік даних) для 32 вузлів у кластері під важким навантаженням, займає всього 28 хвилин.

Скільки часу буде перебудовуватися великий масив (скажемо на сотні терабайт) і RAID 6 – думаємо, ви самі в курсі (багато годин, іноді доба).

З огляду на, що вже зараз готово гелієві диски на 10TB, для традиційних СЗД наступають зовсім лихоліття.

Насправді ж, «все геніальне – просто». Замість використання складних і повільних RAID систем, дані повинні розбиватися на блоки (так звані extent groups у нашім випадку) і просто “розмазуватися” по кластері з потрібною кількістю копій, причому в peer to peer режимі (у російськомовному просторі приклад торрентів відразу всім зрозумілий).

До речі, звідси випливає новий для багатьох (але не для ринку) термін – **RAIN** (**R**edundant/Reliable **A**rray of Inexpensive/**I**ndependent **N**odes) – резервуємий / надійний масив з недорогих / незалежних нодів.

Звучить складно? Скорочуємо: виносимо всю логіку роботи із заліза в чисто програмну реалізацію на стандартному X 86-64 устаткуванні.

Рівно так-же як робить Google / Facebook / Amazon та інші.

Які переваги?

– Швидкий (дуже швидкий) цикл розробки й апдейтів.

– Відвязка від залежностей до пропрієтарного устаткування.

– Використання стандартного (“ширвжиток”) устаткування для рішення завдань будь-яких масштабів.

Розподілені й повністю самодостатні системи

Ідея проста й лежить на поверхні – ідемо від концепції виділених керуючих систем (контролери, центральні вузли метаданих, мозок людини, і т.д.) до концепції рівномірного розподілу однакових ролей між безліччю елементів (кожний вузол – сам собі контролер, немає виділених елементів, бджолиний рій).

Перефразовуючи – чисто розподілена система.

Традиційні виробники завжди виходять із того, що устаткування повинне бути надійним, що в загальному-те в цілому можливо (але якою ціною?).

Тим часом як для розподілених систем підхід принципово відрізняється – завжди виходять із того що будь-яке устаткування в підсумку відмовить і обробка цієї ситуації повинна бути повністю автоматичної, без впливу на життєздатність системи.

Ми говоримо про «» системи, що самовиліковують, причому лікування повинне відбуватися максимально швидко.

Якщо логіка керування вимагає координації (так звані «master» вузли), то вибір їх повинен бути повністю автоматичним і будь-яким учасником кластера може стати таким майстром.

Що це все означає в реаліях?

– Розподіл ролей і відповідальностей усередині системи (кластера).

– Використання BigData принципів (такі як MapReduce) для розподілу завдань.

– Процес «публічних виборів» для позначення поточного майстра.

Лінійне розширення з дуже точної гранулярністю

Послідовне й лінійне (горизонтальне) розширення позначає можливість стартувати з певної кількості ресурсів (у нашім випадку – 3 вузли / нода / сервера) і масштабуватися лінійно для одержання лінійного-же приросту продуктивності. Всі пункти, які ми обговорювали вище, є критичними для одержання такої можливості.

Як приклад, звичайно ви маєте трьохрівневу архітектуру (сервера, система зберігання даних, мережа), кожний елемент якої масштабується незалежно. Якщо збільшили число серверів – то СЗД і мережа при цьому залишилися старі.

С гіпер-конвергентною платформою типу Nutanix, при додаванні кожного нода, ви одержите збільшення:

– Кількості гіпервізоров / комп'ютингових нодів.

– Кількості контролерів СЗД (3 нода = 3 контролери, 300 нодів = 300 контролерів, і т.д.).

– Процесорні потужності і ємність СЗД.

– Кількість нод завдань, що беруть участь у рішенні, кластера.

Спрощуємо:

– Можливість масштабувати сервера й СЗД по одному мікро-вузлі з відповідним лінійним збільшенням продуктивності, починаючи з 3-х і нескінченно.

Переваги:

– Можливість стартувати з мінімального розміру.

– Усунення будь-яких «пляшкових горлечок» і крапок відмови (так, SLA 100% реальний).

– Постійна й гарантована продуктивність при розширенні.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.2. Створена функціональна модель мережі передачі даних між дата-центрами на основі технології Web Scale дозволяє вирішити наступні завдання:

– сформувати наочне візуальне подання взаємодії основних процесів, що відбуваються у мережі передачі даних між дата-центрами на основі технології Web Scale;

– зробити чітке визначення ролі системи поліпшення функціонування в загальній системі передачі даних між дата-центрами на основі технології Web Scale;

– точно визначити вхідні, вихідні й керуючі впливи на підсистему динамічного перерозподіл трафіку мережі передачі даних між дата-центрами на основі технології Web Scale, що надалі дозволяє провести аналітичне й імітаційне моделювання інформаційного потоку;

– сценарій динамічного перерозподілу трафіку мережі передачі даних між дата-центрами на основі технології Web Scale, розроблений у ході виконання магістерської роботи, є основою алгоритму імітаційного моделювання.

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

– Блок формування матриці транзитних вузлів.

– Блок формування сигналів для елементів мережі передачі даних між дата-центрами на основі технології Web Scale.

– Згенеровані звіти по поточному завантаженню елементів мережі передачі даних між дата-центрами на основі технології Web Scale.

– Блок розрахунку вектору значень інтегрального критерію оптимізації.

– Блок ініціалізації граничних значень параметрів моделювання.

– Блок формування матриці динамічний перерозподіл трафіку мережі передачі даних між дата-центрами на основі технології Web Scale.

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 динамічного перерозподілу трафіку мережі передачі даних між дата-центрами на основі технології Web Scale (ДПТМ) є оптимальним і ефективним рішенням, що забезпечує масштабованість і високу відказостійкість таких додатків як в Інтернеті, так і в інтрамережах. Служба ДПТМ дозволяє системним адміністраторам створювати кластери, що включають до 32 вузлів, між якими будуть розподілятися вступні від клієнтів запити. При цьому з погляду клієнтів кластер нічим не відрізняється від звичайного сервера; серверні додатки також не вимагають адаптації для роботі в кластері.

Рисунок 3.2 – Функціональна схема системи

Служба ДПТМ постачена всіма необхідним адміністраторам способами керування, у тому числі можливістю (після уведення пароля) віддалено управляти кластером з будь-якого комп'ютера в мережі передачі даних між дата-центрами на основі технології Web Scale. 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 ДПТМ 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 N. Всі ці характеристики досягаються без необхідності використовувати спеціальні апаратні або програмні рішення.

Тести продуктивності демонструють, що використання програмної служби ДПТМ дає низькі накладні витрати на обробку потоку даних і чудові можливості масштабування продуктивності, обмежені тільки пропускною здатністю підмережі передачі даних між дата-центрами на основі технології Web Scale.

Служба ДПТМ демонструє пропускну здатність більше 200 Мбіт/с у реальних рішеннях по обслуговуванню електронної торгівлі з більш ніж 800 млн. запитів протягом дня.

Служба ДПТМ періодично розсилає ритмічні повідомлення, призначені для інформування кожного члена кластера про наявність інших вузлів. 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.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі передачі даних між дата-центрами на основі технології Web Scale.

3.4 Розробка діаграми процесів

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.

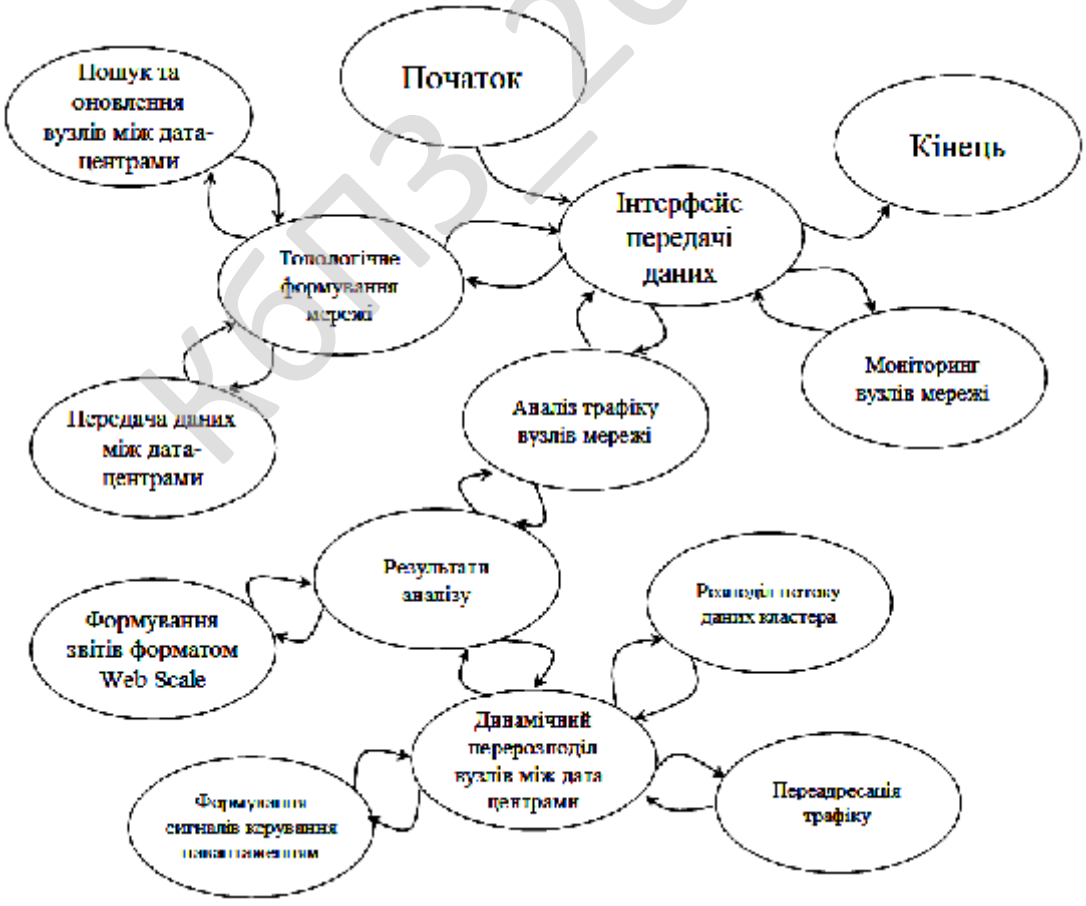

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

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

Діаграми потоків даних містять чотири типи елементів:

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

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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 передачі даних між дата-центрами на основі технології Web Scale.

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.

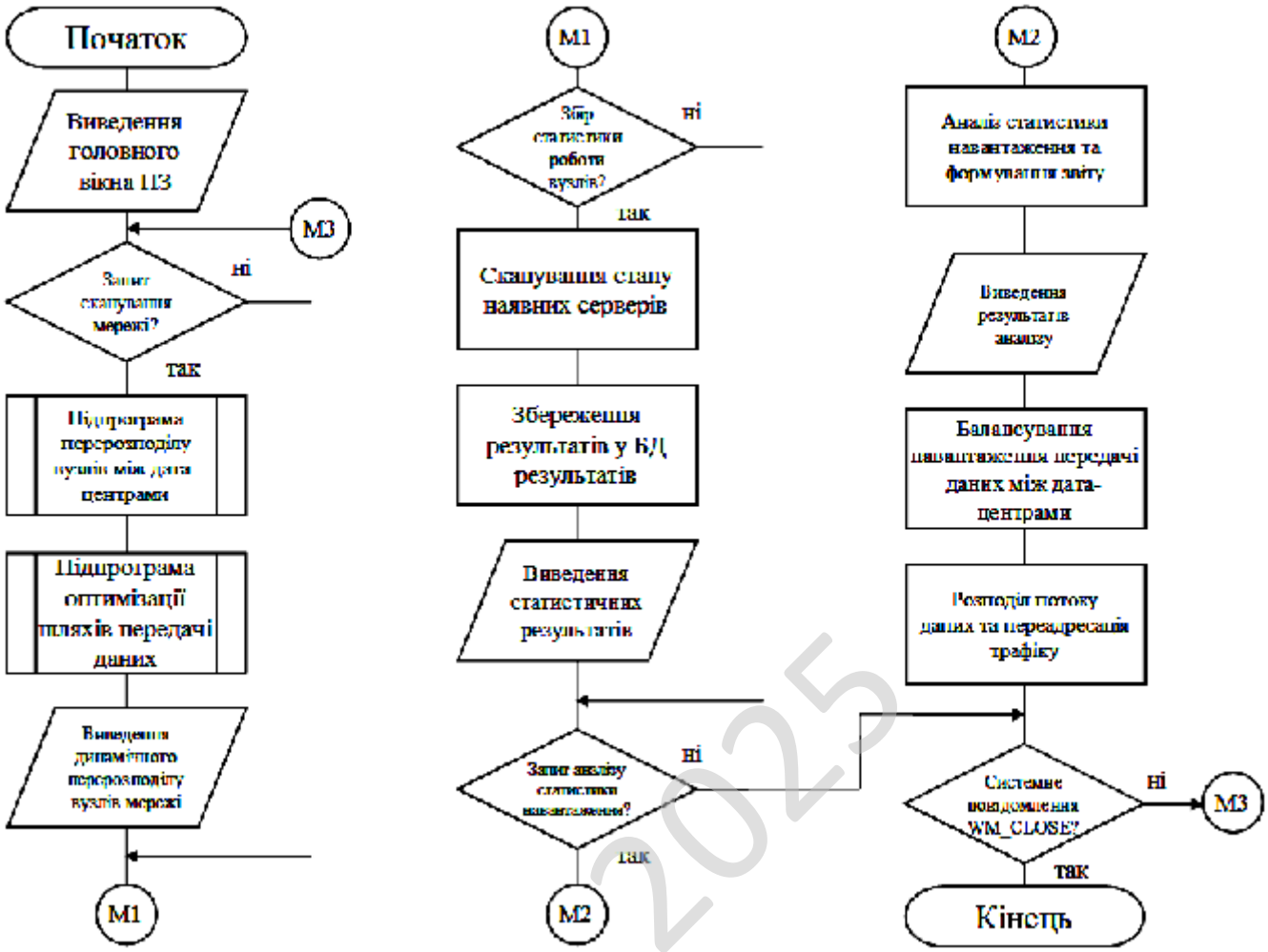

Рисунок 4.1 – Блок-схема основної програми

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. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем.

UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

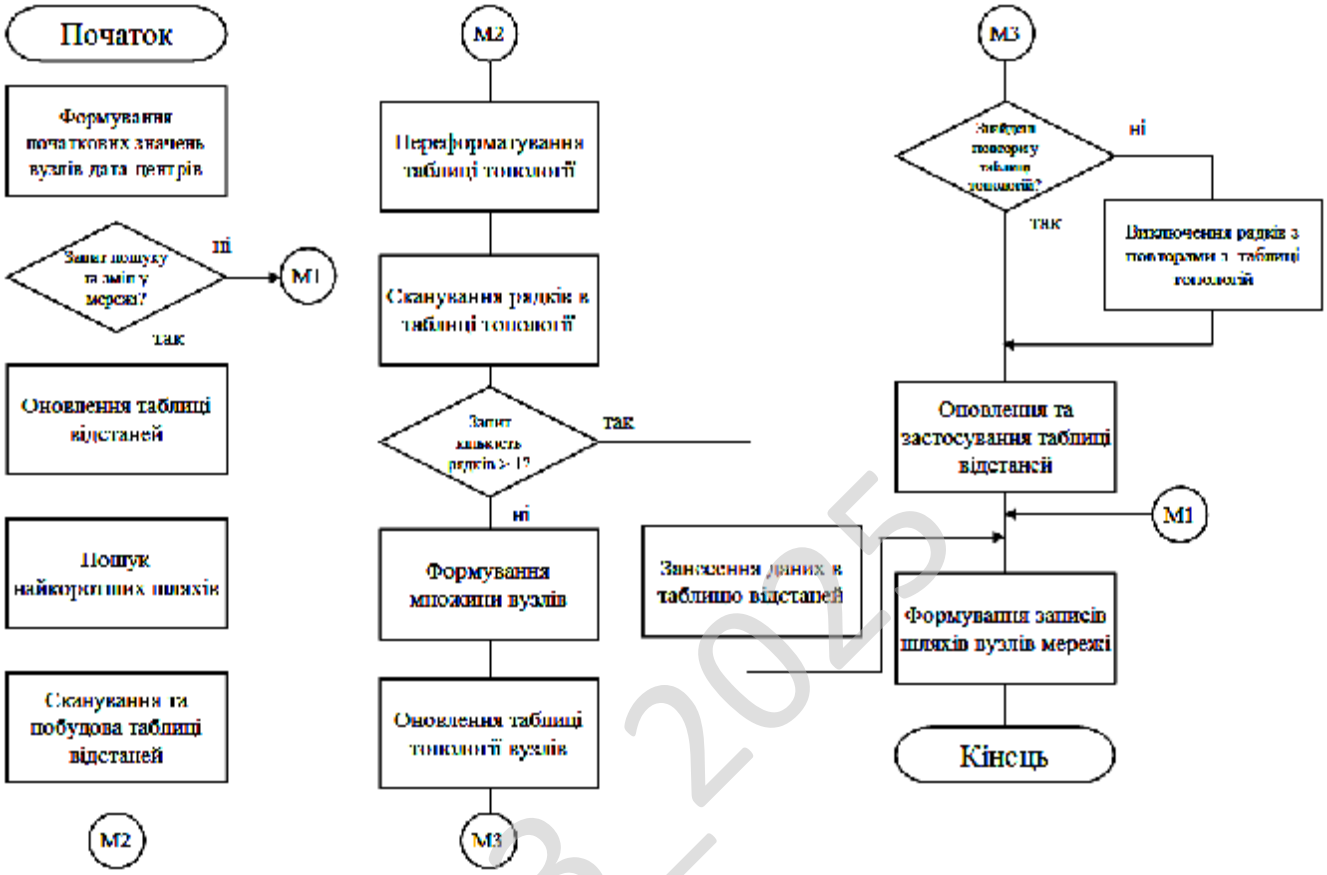

Рисунок 4.2 – Блок-схема роботи підпрограми

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

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

Функціональні можливості:

– Ведення декількох проектів.

– Гнучка система доступу з використанням ролей.

– Система відстеження помилок.

– Діаграми Ганта та календар.

– Ведення новин проекту, документів та управління файлами.

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

– Власна Wiki для кожного проекту.

– Форуми для кожного проекту.

– Облік часових витрат.

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

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

– Створення записів про помилки на основі отриманих листів

– Підтримка LDAP автентифікації.

– Можливість самореєстрації нових користувачів.

– Багатомовний інтерфейс (у тому числі українська мова).

– Підтримка СКБД: MySQL, PostgreSQL, SQLite.

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

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

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню, заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні».

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

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

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

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

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

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

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

– номер (ідентифікатор) дефекту;

– хто повідомив про дефект;

– дата і час виявлення дефекту;

– версія продукту, в якій виявлено дефект;

– серйозність (критичність) дефекту та пріоритет рішення;

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

– відповідальний за усунення дефекту;

– обговорення можливих рішень та їх наслідків;

– поточний стан виправлення дефекту;

– версії продукту, в якій дефект виправлений.

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

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

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

Типовий життєвий цикл дефекту:

1. Новий – дефект зареєстрований тестувальником.

2. Призначений – призначений відповідальний за виправлення дефекту.

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

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

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

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

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

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

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

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

У корпоративному середовищі, система відстеження помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає достатньо точних результатів через те, що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми прямо не стосується складності її усунення.

Опис системи передачі даних між дата-центрами

Система передачі даних між дата-центрами базується на технології Web Scale та використовує мову програмування Python. Основна мета системи – забезпечити високу масштабованість, надійність та ефективність обробки великих обсягів інформації у розподіленому середовищі.

Архітектура системи

Система складається з кількох ключових компонентів. Кожен елемент виконує визначені функції, що забезпечують узгоджену роботу всієї мережі дата-центрів.

1. Шлюзи зв’язку між дата-центрами. Вони забезпечують транспортування пакетів даних через заздалегідь встановлені маршрути. Шлюзи підтримують балансування навантаження та автоматичне перенаправлення трафіку у разі збою.

2. Кластер обробки даних. Складається з кількох серверів, які виконують розподілену обробку інформації. Кожен сервер використовує бібліотеки Python

для паралельних обчислень.

3. Система моніторингу. Відстежує поточний стан мережі, використовуючи спеціальні Python-скрипти. Моніторинг здійснюється за допомогою протоколу SNMP або API-запитів до обладнання.

4. Система кешування. Дозволяє зменшити затримку при передачі даних, зберігаючи найбільш затребувану інформацію у швидкодоступних сховищах. Використовуються бібліотеки Redis або Memcached.

5. Механізм захисту даних. Реалізується за допомогою Python-бібліотек для шифрування та автентифікації користувачів. Включає використання алгоритмів AES або RSA.

Функції та елементи вихідного коду

Основний функціонал реалізується за допомогою Python-коду, що забезпечує передачу, обробку та моніторинг даних. Нижче наведені основні компоненти вихідного коду.

1. Шлюзи зв’язку

```
import socket
def send_data(destination_ip, data):
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((destination_ip, 8080))
s.sendall(data.encode())
s.close()
```

Функція забезпечує встановлення з'єднання та передачу даних через TCP-з'єднання.

2. Обробка даних у кластері

```
from multiprocessing import Pool
def process_data(chunk):
return sum(chunk)
data = [i for i in range(1000000)]
chunks = [data[i:i+10000] for i in range(0, len(data), 10000)]
with Pool(4) as p:
results = p.map(process_data, chunks)
```

Використовується багатопроцесорна обробка для збільшення швидкості обчислень.

3. Моніторинг стану мережі. Ця функція дозволяє отримати статистику щодо трафіку.

```
import psutil
def get_network_status():
return psutil.net_io_counters()
```

4. Система кешування. Зберігає дані у Redis для зменшення навантаження на систему.

```
import redis
cache = redis.Redis(host='localhost', port=6379, db=0)
def cache_data(key, value):
cache.set(key, value)
def get_cached_data(key):
return cache.get(key)
```

5. Шифрування переданих даних. Забезпечення шифрування перед передачею даних.

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
cipher = Fernet(key)
def encrypt_data(data):
return cipher.encrypt(data.encode())
def decrypt_data(encrypted_data):
return cipher.decrypt(encrypted_data).decode()
```


Розглянемо розрахунки та підтвердження проектних рішень

Основні параметри системи визначаються на основі розрахунків продуктивності мережевого з’єднання, обсягу оброблюваних даних та часу відповіді серверів.

1. Продуктивність мережевого з’єднання:

– Середня швидкість передачі даних – 10 Гбіт/с.

– Час на встановлення TCP-з'єднання – 5 мс.

– Максимальна затримка – 50 мс.

2. Продуктивність обробки даних:

– Кількість серверів у кластері – 10.

– Середня швидкість обробки одного сервера – 100 000 операцій/с.

– Загальна пропускна здатність системи – 1 000 000 операцій/с.

3. Моніторинг стану мережі:

– Частота оновлення метрик – 5 с.

– Використання процесора на сервер моніторингу – 2%.

4. Ефективність кешування:

– Середній час доступу до даних у Redis – 1 мс.

– Зниження навантаження на основну базу – 60%.

Обрані проектні рішення дозволяють забезпечити стабільну роботу системи навіть при високих навантаженнях. Масштабованість досягається шляхом горизонтального розширення серверів, а оптимізація алгоритмів передачі даних дозволяє зменшити затримку до мінімальних значень.

Приклад вихідного коду

```
import socket
from multiprocessing import Pool
import psutil
import redis
from cryptography.fernet import Fernet

# Ініціалізація кешу
cache = redis.Redis(host='localhost', port=6379, db=0)

# Генерація ключа для шифрування
key = Fernet.generate_key()
cipher = Fernet(key)

def send_data(destination_ip, data):
    """Функція для відправлення даних по TCP-з'єднанню"""
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((destination_ip, 8080))
    encrypted_data = encrypt_data(data)
    s.sendall(encrypted_data)
```

```
    s.close()
def receive_data():
    """Функція для прийому даних по TCP-з'єднанню"""
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(('0.0.0.0', 8080))
    s.listen(5)
    conn, addr = s.accept()
    data = conn.recv(1024)
    decrypted_data = decrypt_data(data)
    conn.close()
    return decrypted_data

def process_data(chunk):
    """Функція для обробки даних"""
    return sum(chunk)

def parallel_processing(data):
    """Функція для паралельної обробки великих обсягів даних"""
    chunks = [data[i:i+10000] for i in range(0, len(data), 10000)]
    with Pool(4) as p:
        results = p.map(process_data, chunks)
    return sum(results)

def get_network_status():
    """Функція для моніторингу мережевого стану"""
    return psutil.net_io_counters()

def cache_data(key, value):
    """Функція для збереження даних у кеші"""
    cache.set(key, value)

def get_cached_data(key):
    """Функція для отримання даних з кешу"""
    return cache.get(key)

def encrypt_data(data):
    """Функція для шифрування даних"""
    return cipher.encrypt(data.encode())

def decrypt_data(encrypted_data):
    """Функція для розшифрування даних"""
    return cipher.decrypt(encrypted_data).decode()
```

```
if __name__ == "__main__":
    # Тестова передача даних
    test_data = "Hello, Data Center!"
    send_data('127.0.0.1', test_data)

    # Тестова обробка даних
    data_list = [i for i in range(1000000)]
    result = parallel_processing(data_list)
    print("Оброблений результат:", result)

    # Тестовий моніторинг мережі
    network_status = get_network_status()
    print("Статистика мережі:", network_status)

    # Тестове кешування
    cache_data("key1", "value1")
    cached_value = get_cached_data("key1")
    print("Збережене значення у кеші:", cached_value.decode())
```

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне

забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

– прикладний рівень, який реалізує основну логіку ПЗ і на якому здійснюється необхідна обробка інформації;

– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка ПЗ та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка ПЗ. Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення – звичайний браузер. Це позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає).

Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних програм, крім PHP, широко застосовуються Java, Perl, Python, Delphi.

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а

також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

Була використана водоспадна (каскадна) модель життєвого циклу ПЗ (waterfall model) – послідовний метод розробки програмного забезпечення, названий так через діаграму схожу на водоспад.

Ця модель розробки запозичена з системної інженерії у виробництві та будівництві – областях, в яких зміни на пізніх етапах дуже дорогі, або неможливі. Наприклад, для створення складних інженерних конструкцій (споруд, літаків, мостів і т.п.). Зміни в проекті фундаменту будинку після того, як покладений дах коштують дуже дорого, тому перфекціонізм на початкових етапах проектування просто необхідний. Інженери, які починали займатись розробкою програмного забезпечення перейшовши з інших галузей, просто адаптували звичну модель, тому що на ранніх етапах розвитку комп'ютерної техніки не було методологій створених саме для програмування. Проте, схожі методології застосовуються для програмного забезпечення й далі, у випадках коли вимоги фіксовані, і вимагається висока якість та надійність, наприклад в системах для військових чи медичних потреб.

Перший формальний опис водоспадної моделі, після якої вона стала популярною був здійснений В. В. Ройсом у 1970. Попри те, що стаття містить переважно критику методу, на неї часто посилаються.

Переваги методу:

– Ніяких переробок.

– Гарна специфікація перетікає в гарну документацію.

– Зрозуміла модель.

– Розробники можуть мати низьку кваліфікацію.

Недоліки:

– Необхідний перфекціонізм на кожному етапі.

– Важко вносити зміни (якщо взагалі можливо).

– Надлишкове проектування.

– Поділ розробників на "perfect" та "code monkeys".

Модифікації. Через те що цей метод погано підходить для розробки саме ПЗ, частіше використовують його модифікації.

Найвідоміша модифікація – Sashimi. Названа так через японську страву сашимі (суші нарізане і сервіроване так, що складені рядочком шматочки накладаються один на одного). В моделі розробки Сашимі фази життєвого циклу йдуть одна за одною, але при цьому перекриваються одна з одною в часі.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Sinople – симетричний блоковий криптоалгоритм, побудований на основі незбалансованої «мережі Фейстеля». Алгоритм розроблено у 2003 році.

Основні вимоги до алгоритму при його розробці:

– Можливість програмної і апаратної реалізації.

– Висока швидкість.

– Простота.

– Низькі вимоги до пам'яті.

– Високий рівень безпеки.

Алгоритм заснований на 32-розрядних операціях і має 64 раунду, серед яких два типи – C і D. D раунди спроектовані для досягнення максимальної дифузії, C раунди – для досягнення перемішування. F-функція D раунду використовує один з елементів блоку даних (D[3]) та поточного з'єднання (K[r]) для трансформації 3-х елементів блоку даних. F-функція C раунду, навпаки, використовує перші три елемента блоку даних і поточний з'єднання (K[r]) для трансформації останнього елемента блоку даних (D[3]). Раунди D-типу виконуються до раундів C-типу. Додавання ключів з даними проводиться тільки

через таблиці замін. Операції XOR (додавання по модулю 2) обов'язково поєднуються з операціями ADD (додавання по модулю 2^{32}).

Таблиці замін спочатку запозичені з алгоритму MARS і містять 512 32-розрядних елементів, проте були жорстко проаналізовано на предмет посилення.

Ключове розклад було спроектовано з урахуванням вимог:

– Простота.

– Використовується та ж процедура, що і при шифруванні та розшифрування.

– Установка ключа займає менше часу, ніж зашифрування.

– Виключення еквівалентних ключів.

– Виключення слабких ключів.

Алгоритм, згідно із заявою авторів, стійкий до лінійного і диференціального аналізу.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ передачі даних між дата-центрами на основі технології Web Scale яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Навігаційне меню: Мережа; Налаштування; Довідка.

– Розділу обрання вузлів мережі.

– Розділу виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

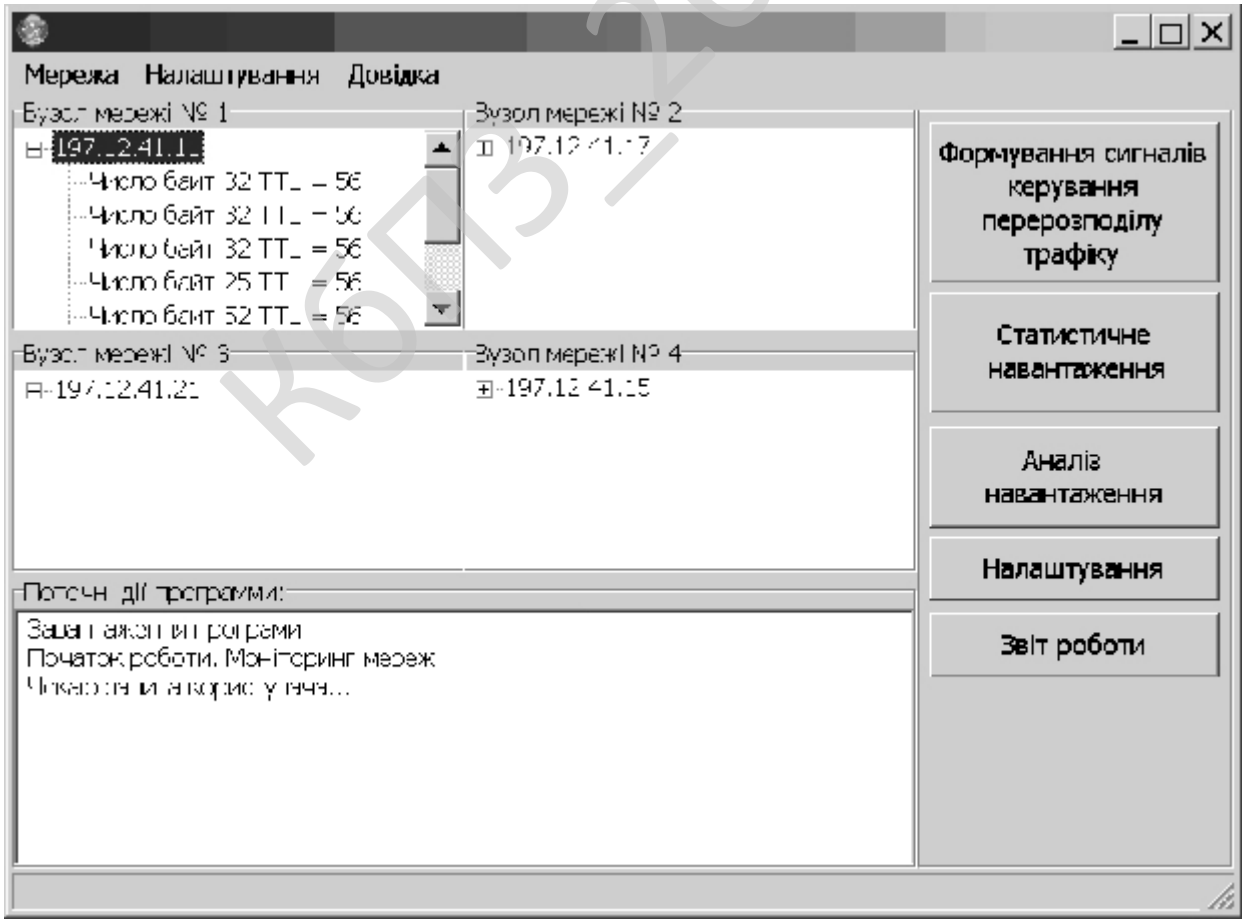

Рисунок 5.1 – Головне вікно ПЗ

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.

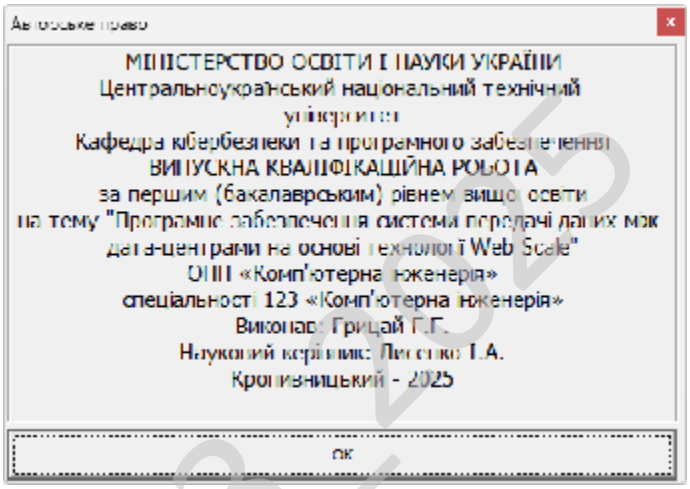

Рисунок 5.2 – Авторське право

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

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

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

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

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

– У програмі можуть бути пропущені деякі маршрути.

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

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

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

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

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

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

Обрано умови розповсюдження – Freeware. Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів. Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення. Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи передачі даних між дата-центрами на основі технології Web Scale.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем передачі даних між дата-центрами на основі технології Web Scale.

– Досліджена система передачі даних між дата-центрами на основі технології Web Scale.

– На основі отриманих результатів досліджень створена програмна реалізація системи передачі даних між дата-центрами на основі технології Web Scale.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання передачі даних між дата-центрами на основі технології Web Scale.

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 призначені для системи передачі даних між дата-центрами на основі технології Web Scale. Це

дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Sinople.

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

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. 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.

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

9. 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.

10. 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.

11. 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.

12. 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.

13. 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.

14. 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.

15. 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.

16. 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.

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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).

40. 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

41. 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.

42. 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.

43. 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.

44. 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.

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

46. 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.

47. 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.

48. 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.

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

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

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

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи передачі даних між дата-центрами на основі технології Web Scale.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 46-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи передачі даних між дата-центрами на основі технології Web Scale.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) 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;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи передачі даних між дата-центрами на основі технології Web Scale;

– 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 забезпеченням.

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 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 77 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 5.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Лисенко І.А.

Програмне забезпечення системи передачі даних між дата-центрами на основі технології Web Scale

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 21

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
# Importing essential modules for system operations, networking, threading, and
data handling
import threading
import time
import random
import requests
import logging
import queue
import json
import uuid
import socket
import sys
import traceback

# Setting up logging configuration for detailed debug information
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s -
%(message)s')

# Class to represent a data packet for transmission between data centers
class DataPacket:
    def __init__(self, source, destination, payload):
        self.id = str(uuid.uuid4())
        self.source = source
        self.destination = destination
        self.timestamp = time.time()
        self.payload = payload
    # Converting packet information to JSON format for standardized transfer
    def to_json(self):
        return json.dumps({
            "id": self.id,
            "source": self.source,
            "destination": self.destination,
            "timestamp": self.timestamp,
            "payload": self.payload
        })
    # Creating a DataPacket object from JSON data
    @staticmethod
    def from_json(json_str):
        try:
            data = json.loads(json_str)
            packet = DataPacket(data.get("source"), data.get("destination"),
data.get("payload"))
            packet.id = data.get("id")
            packet.timestamp = data.get("timestamp")
            return packet
        except Exception as e:
            logging.error("Error parsing JSON: " + str(e))
            return None

# Class representing a Data Center node in the distributed system
class DataCenter:
    def __init__(self, name, host, port):
        self.name = name
        self.host = host
        self.port = port
        self.incoming_queue = queue.Queue()
        self.outgoing_queue = queue.Queue()
        self.running = True
        self.received_packets = []
        self.sent_packets = []
```

```
    # Starting the data center operations including server and sender threads
    def start(self):
        self.server_thread = threading.Thread(target=self.run_server)
        self.server_thread.daemon = True
        self.server_thread.start()
        self.sender_thread = threading.Thread(target=self.process_outgoing)
        self.sender_thread.daemon = True
        self.sender_thread.start()
    # Stopping all active threads of the data center
    def stop(self):
        self.running = False
        self.server_thread.join()
        self.sender_thread.join()
    # Running the server simulation to process incoming packets
    def run_server(self):
        # Simulating server reception loop for incoming data packets
        while self.running:
            try:
                if not self.incoming_queue.empty():
                    packet = self.incoming_queue.get()
                    self.handle_incoming(packet)
                time.sleep(0.1)
            except Exception as ex:
                logging.error("Error in server thread: " + str(ex))
                traceback.print_exc()
    # Processing the outgoing queue to send data packets
    def process_outgoing(self):
        # Simulating outgoing data transfer loop for packets in queue
        while self.running:
            try:
                if not self.outgoing_queue.empty():
                    packet = self.outgoing_queue.get()
                    self.send_packet(packet)
                time.sleep(0.1)
            except Exception as ex:
                logging.error("Error in sender thread: " + str(ex))
                traceback.print_exc()
    # Handling an incoming packet after reception
    def handle_incoming(self, packet):
        logging.info("DataCenter " + self.name + " received packet: " +
packet.id)
        self.received_packets.append(packet)
        self.process_payload(packet.payload)
    # Processing the payload of the received packet
    def process_payload(self, payload):
        logging.info("DataCenter " + self.name + " is processing payload")
        time.sleep(random.uniform(0.05, 0.2))
        logging.info("DataCenter " + self.name + " finished processing payload")
    # Simulating the sending of a data packet over the network
    def send_packet(self, packet):
        logging.info("DataCenter " + self.name + " sending packet: " +
packet.id)
        time.sleep(random.uniform(0.1, 0.3))
        self.sent_packets.append(packet)
        logging.info("DataCenter " + self.name + " finished sending packet: " +
packet.id)
    # Queuing a packet to be sent in the outgoing queue
    def queue_packet_for_sending(self, packet):
        self.outgoing_queue.put(packet)
    # Simulating the receipt of network input by adding the packet to the
incoming queue
    def simulate_network_input(self, packet):
        self.incoming_queue.put(packet)
```

```
# Manager class to handle data transfer operations among multiple data centers
class DataTransferManager:
    def __init__(self):
        self.data_centers = {}
        self.lock = threading.Lock()
        self.running = True
    # Registering a new data center to the transfer manager
    def register_data_center(self, data_center):
        with self.lock:
            self.data_centers[data_center.name] = data_center
            logging.info("Registered DataCenter: " + data_center.name)
    # Deregistering an existing data center from the system
    def deregister_data_center(self, name):
        with self.lock:
            if name in self.data_centers:
                del self.data_centers[name]
                logging.info("Deregistered DataCenter: " + name)
    # Initiating data transfer between two specified data centers
    def send_data_between_centers(self, source_name, destination_name, payload):
        with self.lock:
            if source_name in self.data_centers and destination_name in
self.data_centers:
                packet = DataPacket(source_name, destination_name, payload)
                source_center = self.data_centers[source_name]
                dest_center = self.data_centers[destination_name]
                source_center.queue_packet_for_sending(packet)
                dest_center.simulate_network_input(packet)
                logging.info("Data transfer initiated from " + source_name + "
to " + destination_name)
            else:
                logging.error("Invalid data center names provided for transfer")
    # Broadcasting data from one data center to all others in the network
    def broadcast_data(self, source_name, payload):
        with self.lock:
            if source_name in self.data_centers:
                for dc_name, dc in self.data_centers.items():
                    if dc_name != source_name:
                        packet = DataPacket(source_name, dc_name, payload)

self.data_centers[source_name].queue_packet_for_sending(packet)
                        dc.simulate_network_input(packet)
                        logging.info("Broadcasted packet from " + source_name +
" to " + dc_name)
            else:
                logging.error("Source data center not found for broadcasting")

# Class to simulate a web scale network with multiple interconnected data
centers
class WebScaleNetwork:
    def __init__(self):
        self.transfer_manager = DataTransferManager()
        self.threads = []
        self.running = True
    # Adding a new data center node to the network simulation
    def add_data_center(self, name, host, port):
        dc = DataCenter(name, host, port)
        self.transfer_manager.register_data_center(dc)
        dc.start()
        return dc
    # Simulating network traffic by performing a series of data transfers
between centers
    def simulate_traffic(self, num_requests):
```

```
        for i in range(num_requests):
            try:
                with self.transfer_manager.lock:
                    if len(self.transfer_manager.data_centers) < 2:
                        logging.warning("Not enough data centers for
simulation")
                        break
                    source_name, destination_name =
random.sample(list(self.transfer_manager.data_centers.keys()), 2)
                payload = "Data payload " + str(i) + " - " + str(uuid.uuid4())
                self.transfer_manager.send_data_between_centers(source_name,
destination_name, payload)
                time.sleep(random.uniform(0.05, 0.15))
            except Exception as e:
                logging.error("Error during traffic simulation: " + str(e))
                traceback.print_exc()
    # Shutting down the entire network by stopping all data centers
    def shutdown_network(self):
        with self.transfer_manager.lock:
            for dc in self.transfer_manager.data_centers.values():
                dc.stop()
            self.running = False
            logging.info("WebScaleNetwork shutdown complete")

# Function to perform periodic health checks across all data centers in the
network
def perform_health_checks(transfer_manager):
    while True:
        try:
            logging.info("Performing health checks on all data centers")
            with transfer_manager.lock:
                for dc_name, dc in transfer_manager.data_centers.items():
                    logging.info("Health check for DataCenter: " + dc_name + " -
Outgoing Queue Size: " + str(dc.outgoing_queue.qsize()))
            time.sleep(5)
        except Exception as e:
            logging.error("Error during health check: " + str(e))
            traceback.print_exc()

# Function to simulate an external HTTP query to a specified URL with given data
def simulate_external_query(url, data):
    try:
        logging.info("Simulating external query to URL: " + url)
        response = requests.post(url, json=data)
        logging.info("Received response: " + response.text)
        return response.json()
    except Exception as e:
        logging.error("External query failed: " + str(e))
        traceback.print_exc()
        return None

# Function to simulate data transfers in the network by initializing multiple
operations
def run_simulation():
    logging.info("Starting simulation of Web Scale Data Transfer System")
    network = WebScaleNetwork()
    dc1 = network.add_data_center("DC1", "127.0.0.1", 8001)
    dc2 = network.add_data_center("DC2", "127.0.0.1", 8002)
    dc3 = network.add_data_center("DC3", "127.0.0.1", 8003)
    dc4 = network.add_data_center("DC4", "127.0.0.1", 8004)
    health_thread = threading.Thread(target=perform_health_checks,
args=(network.transfer_manager,))
    health_thread.daemon = True
```

```
    health_thread.start()
    network.simulate_traffic(100)
    simulate_external_query("http://example.com/api/data", {"request":
"status"})
    network.shutdown_network()
    logging.info("Simulation complete")

# Function to log detailed information for debugging and performance tracking
def detailed_logging():
    for i in range(50):
        try:
            logging.debug("Detailed log entry number: " + str(i))
            time.sleep(0.01)
        except Exception as e:
            logging.error("Error in detailed logging: " + str(e))
            traceback.print_exc()

# Dummy function to generate additional load and simulate extended code
complexity
def additional_dummy_function():
    dummy_list = []
    for i in range(100):
        try:
            dummy_value = "DummyValue_" + str(i)
            dummy_list.append(dummy_value)
            if i % 10 == 0:
                logging.debug("Appending " + dummy_value + " to dummy_list")
        except Exception as e:
            logging.error("Error in additional dummy function: " + str(e))
            traceback.print_exc()
    return dummy_list

# Function to provide periodic status updates during system operation
def periodic_status_updates():
    counter = 0
    while counter < 20:
        try:
            logging.info("Status Update: System operational, counter at " +
str(counter))
            time.sleep(0.2)
            counter += 1
        except Exception as e:
            logging.error("Error during periodic status update: " + str(e))
            traceback.print_exc()

# Function to simulate a retry mechanism for ensuring reliable data transfer
def retry_data_transfer(transfer_manager, source, destination, payload,
retries=3):
    attempt = 0
    success = False
    while attempt < retries and not success:
        try:
            logging.info("Retry attempt " + str(attempt) + " for data transfer
from " + source + " to " + destination)
            transfer_manager.send_data_between_centers(source, destination,
payload)
            success = True
        except Exception as e:
            logging.error("Retry attempt " + str(attempt) + " failed: " +
str(e))
            traceback.print_exc()
            attempt += 1
            time.sleep(0.5)
```

```
    if not success:
        logging.error("Data transfer from " + source + " to " + destination + "
failed after " + str(retries) + " attempts")
    return success

# Function to perform asynchronous data transfer using a separate thread
def async_data_transfer(transfer_manager, source, destination, payload):
    try:
        thread =
threading.Thread(target=transfer_manager.send_data_between_centers,
args=(source, destination, payload))
        thread.start()
        return thread
    except Exception as e:
        logging.error("Error starting async data transfer: " + str(e))
        traceback.print_exc()
        return None

# Placeholder function for future system enhancements and additional features
def future_enhancement_placeholder():
    for i in range(10):
        try:
            logging.info("Future enhancement step " + str(i) + " in progress")
            time.sleep(0.1)
        except Exception as e:
            logging.error("Error in future enhancement placeholder: " + str(e))
            traceback.print_exc()

# Function to simulate data replication across multiple data centers
def simulate_data_replication(transfer_manager, source, payload):
    try:
        with transfer_manager.lock:
            if source not in transfer_manager.data_centers:
                logging.error("Source data center not found for replication")
                return
            for dc_name, dc in transfer_manager.data_centers.items():
                if dc_name != source:
                    packet = DataPacket(source, dc_name, payload + " [replica]")

transfer_manager.data_centers[source].queue_packet_for_sending(packet)
                    dc.simulate_network_input(packet)
                    logging.info("Replicated packet from " + source + " to " +
dc_name)
    except Exception as e:
        logging.error("Error during data replication: " + str(e))
        traceback.print_exc()

# Function to simulate load balancing by randomly selecting a destination for
data transfer
def simulate_load_balancing(transfer_manager, payload):
    try:
        with transfer_manager.lock:
            if not transfer_manager.data_centers:
                logging.error("No data centers available for load balancing")
                return
            source = random.choice(list(transfer_manager.data_centers.keys()))
            destinations = list(transfer_manager.data_centers.keys())
            destinations.remove(source)
            if not destinations:
                logging.error("Only one data center present; load balancing not
applicable")
                return
            destination = random.choice(destinations)
```

```
            transfer_manager.send_data_between_centers(source, destination,
payload)
            logging.info("Load balanced transfer from " + source + " to " +
destination)
    except Exception as e:
        logging.error("Error during load balancing simulation: " + str(e))
        traceback.print_exc()

# Function containing extensive dummy routines to further extend the code base
def extensive_dummy_routine():
    try:
        dummy_counter = 0
        dummy_storage = {}
        for i in range(200):
            dummy_key = "key_" + str(i)
            dummy_value = "value_" + str(i) + "_" + str(random.randint(1000,
9999))
            dummy_storage[dummy_key] = dummy_value
            dummy_counter += 1
            if i % 20 == 0:
                logging.debug("Extensive dummy routine iteration " + str(i) + "
with dummy key: " + dummy_key)
        for key, value in dummy_storage.items():
            logging.debug("Processing dummy storage item: " + key + " => " +
value)
        logging.info("Extensive dummy routine completed with total count: " +
str(dummy_counter))
    except Exception as e:
        logging.error("Error in extensive dummy routine: " + str(e))
        traceback.print_exc()

# Function to perform a dummy computation for simulating processor load
def dummy_computation():
    try:
        result = 0
        for i in range(1, 500):
            result += (i * i) % (i + 1)
            if i % 50 == 0:
                logging.debug("Dummy computation reached iteration: " + str(i))
        logging.info("Dummy computation result: " + str(result))
        return result
    except Exception as e:
        logging.error("Error during dummy computation: " + str(e))
        traceback.print_exc()
        return None

# Function to run all auxiliary routines and simulate a full-fledged distributed
system
def run_full_system():
    try:
        logging.info("Running full system simulation")
        dummy_data = additional_dummy_function()
        detailed_logging()
        periodic_status_updates()
        future_enhancement_placeholder()
        extensive_dummy_routine()
        dummy_computation()
        transfer_manager = DataTransferManager()
        dcA = DataCenter("DC_A", "192.168.1.1", 9001)
        dcB = DataCenter("DC_B", "192.168.1.2", 9002)
        transfer_manager.register_data_center(dcA)
        transfer_manager.register_data_center(dcB)
        dcA.start()
```

```
        dcB.start()
        retry_data_transfer(transfer_manager, "DC_A", "DC_B", "Critical payload
data")
        async_thread = async_data_transfer(transfer_manager, "DC_A", "DC_B",
"Asynchronous payload")
        if async_thread:
            async_thread.join()
        simulate_data_replication(transfer_manager, "DC_A", "Replication payload
data")
        simulate_load_balancing(transfer_manager, "Load balanced payload")
        dcA.stop()
        dcB.stop()
        logging.info("Full system simulation completed")
    except Exception as e:
        logging.error("Error in run_full_system: " + str(e))
        traceback.print_exc()

# Main function to start the entire simulation of the web scale data transfer
system
def main():
    try:
        logging.info("Starting main execution of Web Scale Data Transfer
System")
        run_simulation()
        run_full_system()
        logging.info("Main execution completed")
    except Exception as e:
        logging.error("Error in main execution: " + str(e))
        traceback.print_exc()

# Entry point of the program
if __name__ == "__main__":
    main()
```

Файл wids.py

```
import base64
import hashlib
import io
import math
import os
import random
import re
import sqlite3
import sys
import uuid
import warnings
from functools import partial
from typing import BinaryIO, Optional, TypeVar, Union
from urllib.parse import quote, urlparse

import numpy as np
import torch.distributed as dist

from .wids_decode import default_decoder
from .wids_dl import download_and_open
from .wids_lru import LRUCache
from .wids_mmtar import MMIndexedTar
from .wids_specs import load_dsdesc_and_resolve, urldir
from .wids_tar import TarFileReader, find_index_file

try:
    from torch.utils.data import Dataset, Sampler
except ImportError:

    class Dataset:
        pass

    class Sampler:
        pass


T = TypeVar("T")

T_co = TypeVar("T_co", covariant=True)


def compute_file_md5sum(fname: Union[str, BinaryIO], chunksize: int = 1000000) -
> str:
    """Compute the md5sum of a file in chunks.

    Parameters
    ----------
    fname : Union[str, BinaryIO]
        Filename or file object
    chunksize : int, optional
        Chunk size in bytes, by default 1000000

    Returns
    -------
    str
        MD5 sum of the file

    Examples
    --------
    >>> compute_file_md5sum("test.txt")
    'd41d8cd98f00b204e9800998ecf8427e'
```

```
    """
    md5 = hashlib.md5()
    if isinstance(fname, str):
        with open(fname, "rb") as f:
            for chunk in iter(lambda: f.read(chunksize), b""):
                md5.update(chunk)
    else:
        fname.seek(0)
        for chunk in iter(lambda: fname.read(chunksize), b""):
            md5.update(chunk)
    return md5.hexdigest()


def compute_num_samples(fname):
    ds = IndexedTarSamples(path=fname)
    return len(ds)


def splitname(fname):
    """Returns the basename and extension of a filename"""
    assert "." in fname, "Filename must have an extension"
    basename, extension = re.match(r"^((?:.*/)?.*?)(\..*)$", fname).groups()
    return basename, extension


def group_by_key(names):
    """Group the file names by key.

    Args:
        names: A list of file names.

    Returns:
        A list of lists of indices, where each sublist contains indices of files
        with the same key.
    """
    groups = []
    last_key = None
    current = []
    for i, fname in enumerate(names):
        # Ignore files that are not in a subdirectory.
        if "." not in fname:
            print(f"Warning: Ignoring file {fname} (no '.')")
            continue
        key, ext = splitname(fname)
        if key != last_key:
            if current:
                groups.append(current)
            current = []
            last_key = key
        current.append(i)
    if current:
        groups.append(current)
    return groups


open_itfs = {}


class IndexedTarSamples:
    """A class that accesses samples in a tar file. The tar file must follow
    WebDataset conventions. The tar file is indexed when the IndexedTarSamples
    object is created. The samples are accessed by index using the __getitem__
    method. The __getitem__ method returns a dictionary containing the files
```

```
    for the sample. The key for each file is the extension of the file name.
    The key "__key__" is reserved for the key of the sample (the basename of
    each file without the extension). For example, if the tar file contains
    the files "sample1.jpg" and "sample1.txt", then the sample with key
    "sample1" will be returned as the dictionary {"jpg": ..., "txt": ...}.
    """

    def __init__(
        self,
        *,
        path=None,
        stream=None,
        md5sum=None,
        expected_size=None,
        use_mmap=True,
        index_file=find_index_file,
    ):
        assert path is not None or stream is not None

        # Create TarFileReader object to read from tar_file
        self.path = path
        stream = self.stream = stream or open(path, "rb")

        # verify the MD5 sum
        if md5sum is not None:
            stream.seek(0)
            got = compute_file_md5sum(stream)
            assert got == md5sum, f"MD5 sum mismatch: expected {md5sum}, got
{got}"
            stream.seek(0)

        # use either the mmap or the stream based implementation
        if use_mmap:
            self.reader = MMIndexedTar(stream)
        else:
            self.reader = TarFileReader(stream, index_file=index_file)

        # Get list of all files in stream
        all_files = self.reader.names()

        # Group files by key into samples
        self.samples = group_by_key(all_files)

        # check that the number of samples is correct
        if expected_size is not None:
            assert (
                len(self) == expected_size
            ), f"Expected {expected_size} samples, got {len(self)}"

        self.uuid = str(uuid.uuid4())

    def close(self):
        self.reader.close()
        if not self.stream.closed:
            self.stream.close()

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        # Get indexes of files for the sample at index idx
        indexes = self.samples[idx]
        sample = {}
```

```
        key = None
        for i in indexes:
            # Get filename and data for the file at index i
            fname, data = self.reader.get_file(i)
            # Split filename into key and extension
            k, ext = splitname(fname)
            # Make sure all files in sample have same key
            key = key or k
            assert key == k
            sample[ext] = data
        # Add key to sample
        sample["__key__"] = key
        return sample

    def __str__(self):
        return f"<IndexedTarSamples-{id(self)} {self.path}>"

    def __repr__(self):
        return str(self)


def hash_localname(dldir="/tmp/_wids_cache"):
    os.makedirs(dldir, exist_ok=True)

    connection = sqlite3.connect(os.path.join(dldir, "cache.db"))
    cursor = connection.cursor()
    cursor.execute(
        "CREATE TABLE IF NOT EXISTS cache (url TEXT PRIMARY KEY, path TEXT,
checksum TEXT)"
    )
    connection.commit()

    def f(shard):
        """Given a URL, return a local name for the shard."""
        if shard.startswith("pipe:"):
            # uuencode the entire URL string
            hex32 =
base64.urlsafe_b64encode(hashlib.sha256(shard.encode()).digest())[
                :32
            ].decode()
            return os.path.join(dldir, "pipe__" + hex32)
        else:
            # we hash the host and directory components into a 16 character
string
            dirname = urldir(shard)
            hex16 =
base64.urlsafe_b64encode(hashlib.sha256(dirname.encode()).digest())[
                :16
            ].decode()
            # the cache name is the concatenation of the hex16 string and the
file name component of the URL
            cachename = "data__" + hex16 + "__" +
os.path.basename(urlparse(shard).path)
            checksum = None
            cursor.execute(
                "INSERT OR REPLACE INTO cache VALUES (?, ?, ?)",
                (shard, cachename, checksum),
            )
            connection.commit()
            return os.path.join(dldir, cachename)

    return f
```

```
class CacheLocalname:
    def __init__(self, cachedir: str) -> None:
        self._cachedir = cachedir
        os.makedirs(cachedir, exist_ok=True)

    def __call__(self, shard: str) -> str:
        """Given a URL, return a local name for the shard."""
        path = urlparse(shard).path
        fname = os.path.basename(path)
        return os.path.join(self._cachedir, fname)


class DefaultLocalname:
    def __init__(self, dldir: str = "/tmp/_wids_cache") -> None:
        self._dldir = dldir
        os.makedirs(dldir, exist_ok=True)

    def __call__(self, shard: str) -> str:
        """Given a URL, return a local name for the shard."""
        cachename = quote(shard, safe="+-")
        return os.path.join(self._dldir, cachename)


class LRUShards:
    """A class that manages a cache of shards. The cache is a LRU cache that
    stores the local names of the shards as keys and the downloaded paths as
    values. The shards are downloaded to a directory specified by dldir.
    The local name of a shard is computed by the localname function, which
    takes the shard URL as an argument. If keep is True, the downloaded files
    are not deleted when they are no longer needed.
    """

    def __init__(self, lru_size, keep=False, localname=DefaultLocalname()):
        self.localname = localname
        # the cache contains the local name as the key and the downloaded path
as the value
        self.lru = LRUCache(lru_size, release_handler=self.release_handler)
        # keep statistics
        self.reset_stats()

    def reset_stats(self):
        self.accesses = 0
        self.misses = 0

    def __len__(self):
        return len(self.lru)

    def release_handler(self, key, value):
        value.close()

    def clear(self):
        self.lru.clear()

    def get_shard(self, url):
        assert isinstance(url, str)
        self.accesses += 1
        if url not in self.lru:
            local = self.localname(url)
            with download_and_open(url, local) as stream:
                itf = IndexedTarSamples(path=local, stream=stream)
            self.lru[url] = itf
            self.misses += 1
```

```
            self.last_missed = True
        else:
            self.last_missed = False
        return self.lru[url]


def interpret_transformations(transformations):
    """Interpret the transformations argument.

    This takes care of transformations specified as string shortcuts
    and returns a list of callables.
    """
    if not isinstance(transformations, list):
        transformations = [transformations]

    result = []

    for transformation in transformations:
        if transformation == "PIL":
            warnings.warn(
                "String specifications for transformations are deprecated. Use
functions instead."
            )
            transformation = partial(default_decoder, format="PIL")
        elif transformation == "numpy":
            warnings.warn(
                "String specifications for transformations are deprecated. Use
functions instead."
            )
            transformation = partial(default_decoder, format="numpy")
        else:
            assert callable(transformation)
        result.append(transformation)

    return result


def hash_dataset_name(input_string):
    """Compute a hash of the input string and return the first 16 characters of
the hash."""
    # Compute SHA256 hash of the input string
    hash_object = hashlib.sha256(input_string.encode())
    hash_digest = hash_object.digest()

    # Encode the hash in base64
    base64_encoded_hash = base64.urlsafe_b64encode(hash_digest)

    # Return the first 16 characters of the base64-encoded hash
    return base64_encoded_hash[:16].decode("ascii")


class ShardListDataset(Dataset[T]):
    """An indexable dataset based on a list of shards.

    The dataset is either given as a list of shards with optional options and
name,
    or as a URL pointing to a JSON descriptor file.

    Datasets can reference other datasets via `source_url`.

    Shard references within a dataset are resolve relative to an explicitly
    given `base` property, or relative to the URL from which the dataset
    descriptor was loaded.
```

```
    """

    def __init__(
        self,
        shards,
        *,
        cache_size=int(1e12),
        cache_dir=None,
        lru_size=10,
        dataset_name=None,
        localname=None,
        transformations="PIL",
        keep=False,  # keep downloaded files
        base=None,  # base URL for resolving relative URLs
        options=None,  # override JSON options
    ):
        """Create a ShardListDataset.

        Args:
            shards: a list of (filename, length) pairs or a URL pointing to a
JSON descriptor file
            cache_size: the number of shards to keep in the cache
            lru_size: the number of shards to keep in the LRU cache
            localname: a function that maps URLs to local filenames

        Note that there are two caches: an on-disk directory, and an in-memory
LRU cache.
        """
        if options is None:
            options = {}
        super(ShardListDataset, self).__init__()
        # shards is a list of (filename, length) pairs. We'll need to
        # keep track of the lengths and cumulative lengths to know how
        # to map indices to shards and indices within shards.
        if isinstance(shards, (str, io.IOBase)):
            if base is None and isinstance(shards, str):
                base = urldir(shards)
            self.base = base
            self.spec = load_dsdesc_and_resolve(shards, options=options,
base=base)
            self.shards = self.spec.get("shardlist", [])
            self.dataset_name = self.spec.get("name") or
hash_dataset_name(str(shards))
        else:
            self.base = None
            self.spec = options
            self.shards = shards
            self.dataset_name = dataset_name or hash_dataset_name(str(shards))

        self.lengths = [shard["nsamples"] for shard in self.shards]
        self.cum_lengths = np.cumsum(self.lengths)
        self.total_length = self.cum_lengths[-1]

        if cache_dir is not None:
            # when a cache dir is explicitly given, we download files into
            # that directory without any changes
            self.cache_dir = cache_dir
            self.localname = CacheLocalname(cache_dir)
        elif localname is not None:
            # when a localname function is given, we use that
            self.cache_dir = None
            self.localname = localname
        else:
```

```
            # when no cache dir or localname are given, use the cache from the
environment
            self.cache_dir = os.environ.get("WIDS_CACHE", "/tmp/_wids_cache")
            self.localname = DefaultLocalname(self.cache_dir)

        if True or int(os.environ.get("WIDS_VERBOSE", 0)):
            nbytes = sum(shard.get("filesize", 0) for shard in self.shards)
            nsamples = sum(shard["nsamples"] for shard in self.shards)
            print(
                str(shards)[:50],
                "base:",
                self.base,
                "name:",
                self.spec.get("name"),
                "nfiles:",
                len(self.shards),
                "nbytes:",
                nbytes,
                "samples:",
                nsamples,
                "cache:",
                self.cache_dir,
                file=sys.stderr,
            )
        self.transformations = interpret_transformations(transformations)

        if lru_size > 200:
            warnings.warn(
                "LRU size is very large; consider reducing it to avoid running
out of file descriptors"
            )
        self.cache = LRUShards(lru_size, localname=self.localname, keep=keep)

    def add_transform(self, transform):
        """Add a transformation to the dataset."""
        self.transformations.append(transform)
        return self

    def __len__(self):
        """Return the total number of samples in the dataset."""
        return self.total_length

    def get_stats(self):
        """Return the number of cache accesses and misses."""
        return self.cache.accesses, self.cache.misses

    def check_cache_misses(self):
        """Check if the cache miss rate is too high."""
        accesses, misses = self.get_stats()
        if accesses > 100 and misses / accesses > 0.3:
            # output a warning only once
            self.check_cache_misses = lambda: None
            print(
                "Warning: ShardListDataset has a cache miss rate of
{:.1%}%".format(
                    misses * 100.0 / accesses
                )
            )

    def get_shard(self, index):
        """Get the shard and index within the shard corresponding to the given
index."""
        # Find the shard corresponding to the given index.
```

```
            shard_idx = np.searchsorted(self.cum_lengths, index, side="right")

            # Figure out which index within the shard corresponds to the
            # given index.
            if shard_idx == 0:
                inner_idx = index
            else:
                inner_idx = index - self.cum_lengths[shard_idx - 1]

            # Get the shard and return the corresponding element.
            desc = self.shards[shard_idx]
            url = desc["url"]
            shard = self.cache.get_shard(url)
            return shard, inner_idx, desc

        def __getitem__(self, index):
            """Return the sample corresponding to the given index."""
            shard, inner_idx, desc = self.get_shard(index)
            sample = shard[inner_idx]

            # Check if we're missing the cache too often.
            self.check_cache_misses()

            sample["__dataset__"] = desc.get("dataset")
            sample["__index__"] = index
            sample["__shard__"] = desc["url"]
            sample["__shardindex__"] = inner_idx

            # Apply transformations
            for transform in self.transformations:
                sample = transform(sample)

            return sample

        def close(self):
            """Close the dataset."""
            self.cache.clear()


    def lengths_to_ranges(lengths):
        """Convert a list of lengths to a list of ranges."""
        ranges = []
        start = 0
        for length in lengths:
            ranges.append((start, start + length))
            start += length
        return ranges


    def intersect_range(a, b):
        """Return the intersection of the two half-open integer intervals."""
        result = max(a[0], b[0]), min(a[1], b[1])
        if result[0] >= result[1]:
            return None
        return result


    def intersect_ranges(rangelist, r):
        """Return the intersection of the half-open integer interval r with the list
    of half-open integer intervals."""
        result = []
        for a in rangelist:
            x = intersect_range(a, r)
```

```
        if x is not None:
            result.append(x)
    return result


def iterate_ranges(ranges, rng, indexshuffle=True, shardshuffle=True):
    """Iterate over the ranges in a random order."""
    shard_indexes = list(range(len(ranges)))
    if shardshuffle:
        rng.shuffle(shard_indexes)
    for i in shard_indexes:
        lo, hi = ranges[i]
        sample_indexes = list(range(lo, hi))
        if indexshuffle:
            rng.shuffle(sample_indexes)
        yield from sample_indexes


class ShardListSampler(Sampler):
    """A sampler that samples consistent with a ShardListDataset.

    This sampler is used to sample from a ShardListDataset in a way that
    preserves locality.

    This returns a permutation of the indexes by shard, then a permutation of
    indexes within each shard. This ensures that the data is accessed in a
    way that preserves locality.

    Note that how this ends up splitting data between multiple workers ends up
    on the details of the DataLoader. Generally, it will likely load samples
from the
    same shard in each worker.

    Other more sophisticated shard-aware samplers are possible and will likely
    be added.
    """

    def __init__(self, dataset, *, lengths=None, seed=0, shufflefirst=False):
        if lengths is None:
            lengths = list(dataset.lengths)
        self.ranges = lengths_to_ranges(lengths)
        self.seed = seed
        self.shufflefirst = shufflefirst
        self.epoch = 0

    def __iter__(self):
        self.rng = random.Random(self.seed + 1289738273 * self.epoch)
        shardshuffle = self.shufflefirst or self.epoch > 0
        yield from iterate_ranges(self.ranges, self.rng,
shardshuffle=shardshuffle)
        self.epoch += 1


ShardedSampler = ShardListSampler


class ChunkedSampler(Sampler):
    """A sampler that samples in chunks and then shuffles the samples within
each chunk.

    This preserves locality of reference while still shuffling the data.
    """
```

```
    def __init__(
        self,
        dataset,
        *,
        dslength_per_replica=-1,
        num_samples=None,
        chunksize=2000,
        seed=0,
        shuffle=True,
        shufflefirst=False,
    ):
        if isinstance(num_samples, int):
            lo, hi = 0, num_samples
        elif num_samples is None:
            lo, hi = 0, len(dataset)
        else:
            lo, hi = num_samples

        self.dslength_per_replica = (
            dslength_per_replica if dslength_per_replica > 0 else (hi - lo)
        )

        self.ranges = [(i, min(i + chunksize, hi)) for i in range(lo, hi,
chunksize)]
        self.seed = seed
        self.shuffle = shuffle
        self.shufflefirst = shufflefirst
        self.epoch = 0

    def set_epoch(self, epoch):
        self.epoch = epoch

    def __iter__(self):
        self.rng = random.Random(self.seed + 1289738273 * self.epoch)
        shardshuffle = self.shufflefirst or self.epoch > 0
        yield from iterate_ranges(
            self.ranges,
            self.rng,
            indexshuffle=self.shuffle,
            shardshuffle=(self.shuffle and shardshuffle),
        )
        self.epoch += 1

    def __len__(self) -> int:
        return self.dslength_per_replica


def DistributedChunkedSampler(
    dataset: Dataset,
    *,
    num_replicas: Optional[int] = None,
    num_samples: Optional[int] = None,
    rank: Optional[int] = None,
    shuffle: bool = True,
    shufflefirst: bool = False,
    seed: int = 0,
    drop_last: bool = None,
    chunksize: int = 1000000,
) -> ChunkedSampler:
    """Return a ChunkedSampler for the current worker in distributed training.

    Reverts to a simple ChunkedSampler if not running in distributed mode.
```

```
    Since the split among workers takes place before the chunk shuffle,
    workers end up with a fixed set of shards they need to download. The
    more workers, the fewer shards are used by each worker.
    """
    if drop_last is not None:
        warnings.warn(
            "DistributedChunkedSampler does not support drop_last, thus it will
be ignored"
        )
    if not dist.is_initialized():
        warnings.warn(
            "DistributedChunkedSampler is called without distributed
initialized; assuming single process"
        )
        num_replicas = 1
        rank = 0
    else:
        num_replicas = num_replicas or dist.get_world_size()
        rank = rank or dist.get_rank()
    assert rank >= 0 and rank < num_replicas

    # From
https://github.com/pytorch/pytorch/blob/13fa59580e4dd695817ccf2f24922fd211667fc8
/torch/utils/data/distributed.py#L93
    dslength_per_replica = (
        math.ceil(len(dataset) / num_replicas) if num_replicas > 1 else
len(dataset)
    )

    num_samples = num_samples or len(dataset)
    worker_chunk = (num_samples + num_replicas - 1) // num_replicas
    worker_start = rank * worker_chunk
    worker_end = min(worker_start + worker_chunk, num_samples)
    return ChunkedSampler(
        dataset,
        dslength_per_replica=dslength_per_replica,
        num_samples=(worker_start, worker_end),
        chunksize=chunksize,
        seed=seed,
        shuffle=shuffle,
        shufflefirst=shufflefirst,
    )
```