НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

На правах рукопису

СТРУБИЦЬКИЙ РОСТИСЛАВ ПАВЛОВИЧ

УДК 621.391

МЕТОДИ ТА АЛГОРИТМИ ПОБУДОВИ ХМАРКОВИХ СХОВИЩ ДАНИХ НА ОСНОВІ РОЗПОДІЛЕНИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ

05.12.02 – телекомунікаційні системи та мережі

Дисертація на здобуття наукового ступеня
кандидата технічних наук

Науковий керівник:
д.т.н., професор Шаховська Н.Б.

Ідентичність всіх примірників дисертації
ЗАСВІДЧУЮ:
Вчений секретар спеціалізованої
вченої ради
/І.В. Демидов/

Львів - 2017

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AIMD – Additive Increase/Multiplicative Decrease

API – Application Programming Interface

ARM – Advanced RISC Machine

ATA – Advanced Technology Attachment

BGP – Border Gateway Protocol

CDN – Content Distribution/Delivery Network

CIFS – Common Internet File System

DAS – Direct-attached storage

DCCP – Datagram Congestion Control Protocol)

DNS – Domain Name System

EDA – Equipment Distribution Area

eSATA – External Serial Advanced Technology Attachment

EULA – End-User License Agreement

FASP – Fast and Secure Protocol

FTP – File Transfer Protocol

HDA – Horizontal Distribution Area

HDS – Hitachi Data Systems

HTML – HyperText Markup Language

HTTP – Hyper Text Transfer Protocol

IDA – Fast and Secure Protocol

IETF – Internet Engineering Task Force

IP – Internet Protocol

MAN – Metropolitan Area Network

MDA – Main Distribution Area

MPLS – Multiprotocol Label Switching

MTU – Maximum Transmission Unit

NAS – Network Attached Storage

NFS – Network File System

OSI – Open Systems Interconnection

QoS – Quality of Service

RAID – Redundant Array of Independent/Inexpensive Disks

RBUDP – Reliable Blast User Datagram Protocol

REST – Representational State Transfer

RISC – Reduced Instruction Set Computing

RPC – Remote procedure call

RTT – Round-Trip Time

SAN – Storage Area Network

SATA – Serial Advanced Technology Attachment

SCSI – Small Computer Systems Interface

SCTP – Stream Control Transmission Protocol

SDK – Software Development Kit

SLA – Service-level agreement

SMB – Server Message Block

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

TCP – Transmission Control Protocol

TFRC – Transferrin Receptor Protein

UDP – User Datagram Protocol

VPN – Virtual Private Network

XCP – Universal Measurement and Calibration Protocol

ІТ – Інформаційні технології

ОС – Операційна система

СУБД – Система управління базами даних

XMPP – Extensible Messaging and Presence Protocol

ХСД – Хмаркове сховище даних

ВСТУП

Актуальність теми. На сьогодні, в умовах високої конвергенції бізнес процесів у глобальному інформаційному просторі, високої актуальності набувають питання інформаційної взаємодії корпоративних клієнтів, наприклад транснаціональних корпорацій, що розміщуються на територіях, які мають ускладнення у вільному доступі до глобальної інформаційної інфраструктури (КНР, країни Перської затоки, Африки, деякі країни Азії тощо). Хмарні сховища даних дозволяють кардинально підвищити доступність клієнтських даних та необхідних мережних елементів для їх надійного передавання і зберігання. На сьогодні методи локального опрацювання та зберігання мають надзвичайно низький рівень консолідації обчислювальних ресурсів та пам'яті (менше 18%). Географічна розподіленість клієнтських застосувань, їх мобільність з одночасною потребою у збереженні цілісності даних породжують протиріччя, яке полягає у необхідності підвищення пропускної спроможності існуючої телекомунікаційної складової розподілених сховищ даних в умовах підвищення вимог щодо їх доступності, а також відносно несанкціонованого доступу та захисту від пошкоджень.

Отже, наукове завдання розроблення моделей та методів підвищення пропускної спроможності розподілених телекомунікаційних систем високодоступних хмарних сховищ даних на основі нових протоколів доступу є актуальним і своєчасним.

Поряд із цим, існує ряд недостатньо опрацьованих наукових завдань, що стають на заваді ефективній організації хмаркових сховищ даних, та, відповідно, розподілених обчислювальних систем на їх основі, а саме:

- недостатньо розвинена теоретична база, яка прийшла би на зміну класичній теорії масового обслуговування при проектуванні сучасних телекомунікаційних систем розподілу інформації з самоподібним трафіком;

- недостатньо опрацьовані питання визначення показників якості функціонування систем передавання та розподілу інформації у розподіленому гетерогенному мережному середовищі;
- недостатньо розвинуті методи та алгоритми, які забезпечують якість обслуговування, зокрема пропускну спроможність в умовах гетерогенності мережних платформ.

Отже, наукове завдання розроблення моделей та методів підвищення пропускної спроможності розподілених телекомунікаційних систем високодоступних хмарних сховищ даних на основі нових протоколів доступу є актуальним і своєчасним.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалася в межах наукової тематики кафедри інформаційних систем та мереж Національного університету "Львівська політехніка", тема «Комплекс інтелектуальний інформаційних технологій інтеграції даних для обліку та аналізу підвищення кваліфікації вчителів», № держреєстрації 0113U005273 (автор розробив архітектуру хмаркового сховища даних для розподіленого зберігання мультимедійної навчальної інформації).

Мета і задачі дослідження. Метою дисертаційної роботи є розроблення елементів телекомунікаційної мережної архітектури для передавання даних через хмаркові сховища.

Мета дисертаційної роботи визначає необхідність розв'язання таких завдань:

1. Аналіз проблеми запровадження та функціонування хмаркових сховищ даних;

2. Розроблення моделі хмаркового сховища даних та проведення моделювання його завантаженості;

3. Розроблення телекомунікаційного протоколу сеансового рівня на базі UDP для магістральних розподілених MAN- мереж;

4. Розроблення методу мультипротокольного доступу до потокових даних при їх наскрізному передаванні у розподіленій телекомунікаційній системі;

5. Розроблення методу агрегації навантаження декількох джерел даних для їх одночасного передавання через розподілені телекомунікаційні системи хмарних сховищ;

6. Розроблення методу вибору мережного шлюзу за складністю запитів;

7. Апробація розробленого програмного забезпечення для передавання даних у телекомунікаційній системі між вузлами хмарного сховища даних.

Об'єкт дослідження – процес передавання даних через хмаркові сховища.

Предмет дослідження – методи та засоби організації хмаркових сховищ даних на основі розподілених телекомунікаційних систем.

Методи дослідження. Дослідження, виконані під час роботи над дисертацією, ґрунтуються на методах системного аналізу – для розв'язання завдання аналізу процесів функціонування хмаркових сховищ даних та вдосконалення чинних підходів до організації таких сховищ; комп'ютерного (імітаційного чисельного) моделювання – для проведення стендових експериментів з розробленими телекомунікаційними протоколами, спостереження, синтезу; статистичних – для оброблення та аналізу результатів експериментів; об'єктно-орієнтованого аналізу та проектування – для розроблення елементів телекомунікаційної мережної архітектури системи.

Наукова новизна одержаних результатів. Отримано такі нові наукові результати:

1. Вперше розроблено метод мультипротокольного наскрізного передавання даних у розподіленій телекомунікаційній системі хмарних сховищ, який, на відміну від методу вибору протоколу в сегментованій мережі, характеризується вищою продуктивністю, адаптуючись під кожний окремий мережний сегмент.

2. Удосконалено модель хмаркового сховища даних, подану як алгебраїчну систему, що відрізняється від існуючих введенням у архітектуру пов’язаної телекомунікаційної мережі системи методів опрацювання даних на основі протокольних засобів сеансового рівня, що дало змогу більш точно і повно визначити і використовувати її пропускну спроможність відповідно.

3. Набув подальшого розвитку метод агрегації навантаження декількох джерел даних, який, на відміну від методу балансування навантаження, в режимі реального часу визначає завантаженість сервісів хмаркового сховища даних та каналів телекомунікаційної системи, що дає змогу оптимізувати їх продуктивність.

Практичне значення одержаних результатів полягає у досягненні таких результатів:

- розроблено уніфікований протокол передавання даних сеансового рівня. Це дало змогу збільшити пропускну здатність телекомунікаційних каналів розподіленої системи сховищ даних від 1,5 до 2 разів;
- розроблено метод мультипротокольного передавання даних, що дало змогу адаптуватися до наскрізного каналу передавання даних та обмежень, які на нього накладені;
- розроблено метод агрегації навантаження декількох джерел даних, що уможливило їх паралельне передавання;
- удосконалено елементи мережної архітектури хмарних сховищ даних, що дало змогу збільшити їх продуктивність та відмовостійкість шляхом оптимального вибору шлюзу для передавання даних;
- на основі розробленої архітектури побудовано та впроваджено хмаркове сховище даних для промислового використання.

Одержані в дисертаційній роботі результати використано під час розроблення хмаркового сховища даних та організації обчислень на його основі компаніями ТОВ «Глобальна платіжна мережа» (WIDEUP), Ypsilon.Net AG (ФРН).

Апробація результатів дисертації. Основні результати дисертаційної роботи доповідалися на семінарах та конференціях:

- XII Міжнародний науковий семінар «Сучасні проблеми інформатики в управлінні, економіці та освіті», Київ-оз.Світязь, 1-5 липня 2013 р.;
- V Всеукраїнська науково-практична конференція «Стан та удосконалення безпеки інформаційно-телекомунікаційних систем (SITS'2013)», Миколаїв-Коблево, 10-13 вересня 2013 р.;
- Міжнародна науково-практична конференція молодих вчених та студентів «Інформаційні технології, економіка та право: стан та перспективи розвитку», Чернівці, 1-4 квітня 2014 р.;
- VI Всеукраїнська науково-практична конференція «Стан та удосконалення безпеки інформаційно-телекомунікаційних систем (SITS'2014)», Миколаїв-Коблево, 9-12 вересня 2014 р.;
- Міжнародна науково-практична конференція молодих вчених та студентів «Інформаційні технології, економіка та право: стан та перспективи розвитку», Чернівці, 2014 р.;
- VII Всеукраїнська науково-практична конференція «Стан та удосконалення безпеки інформаційно-телекомунікаційних систем (SITS'2015)», Миколаїв-Коблево, 9-12 червня 2015 р.;
- V Всеукраїнська школа-семінар молодих вчених і студентів «Сучасні комп'ютерні інформаційні технології (ACIT'2015)», Тернопіль, 22-23 травня 2015 р.
- XIII Міжнародна конференція «сучасні проблеми радіоелектроніки, телекомунікацій, комп’ютерної інженерії (TCSET’2016)», Львів–Славське, 23–26 лютого 2016.

Публікації. Основні результати дисертаційної роботи висвітлено в 15 друкованих працях, у тому числі дві статті в іноземних періодичних наукових виданнях [95, 99], 6 – у фахових наукових виданнях України [19,

17, 20, 21, 11, 12], 6 – у матеріалах конференцій [14, 16, 15, 18, 10, 100] та одна у науковому економічному журналі [13].

Особистий внесок здобувача. Усі наукові результати, подані у дисертації, одержані здобувачем особисто. У друкованих працях, опублікованих у співавторстві, внесок здобувача такий: [95] – розроблено схему бази даних начального департаменту та ієрархічну організацію процесу опрацювання наукових матеріалів; [10, 11] – виділено особливості організації розподіленого сховища даних для забезпечення захисту від потенційних загроз; [12, 100] – удосконалено модель організації хмаркового сховища даних; [19] – розроблено метод вибору оптимального шлюзу доступу до хмаркового сховища даних; [13] – розроблено схему ієрархії хмаркового сховища даних та діаграми потоків даних процесу визначення оптимального сателіта; [18] – розроблено та оптимізовано модель захищеного сховища засобами відкритого програмного забезпечення.

Структура та обсяг роботи. Дисертація складається зі вступу, чотирьох розділів, висновків, що містять основні результати роботи, списку використаної літератури зі 116 найменувань, 2 додатків, 70 рисунків та 6 таблиць. Повний обсяг дисертації складає 176 сторінок, із них 138 сторінок основного тексту.

РОЗДІЛ 1.
АНАЛІЗ РОЗВИТКУ ТА СУЧАСНИЙ СТАН СИСТЕМ ЗБЕРІГАННЯ ДАНИХ

У розділі – систематизовано джерела та з’ясований стан наукових досліджень за темою дисертації. Зокрема, проведено аналіз архітектур систем зберігання даних (див. Додаток А), досліджено основні підходи до хмаркових технологій, виділено особливості формування архітектури хмаркових сховищ даних і їх проблематику та проведено аналіз транспортування даних в розподілених системах.

1.1. Основні підходи до хмаркових технологій.

Хмарові обчислення (cloud computing, «Хмаркова (розсіяна) обробка даних») – технологія, яка забезпечує користувачів (обладнання) мережевим доступом до загального пулу ресурсів, як Інтернет-сервіс. Користувач має доступ до особистих даних, але не може управляти і не повинен піклуватися про інфраструктуру, операційну систему і програмне забезпечення, з яким він працює. Це можуть бути як безпосередньо обчислювальні потужності, так і сховища даних, різні сервіси, програмне забезпечення, і навіть мережі передачі даних. При цьому дані ресурси стають доступними оперативно, і з різко зниженими експлуатаційними затратами.

Термін «Хмарка» використовується як метафора зображення Інтернету на діаграмі комп’ютерної мережі, або як образ складної інфраструктури, за якою приховуються всі технічні деталі реалізації. Сам термін походить з телефонії, тому що телекомунікаційні компанії, які до 1990-х років пропонували в основному виділені схеми передачі «точка - точка», почали пропонувати віртуальні приватні мережі (VPN), з порівняною якістю обслуговування, але при набагато менших витратах.

Перемикаючи трафік для оптимального використання каналів вони мали змогу ефективніше використовувати мережу. Символ хмарки був використаний для позначення розмежування між користувачем і

постачальником.

Згідно документу IEEE опублікованому в 2008 році, «Хмаркова обробка даних – це парадигма, в рамках якої інформація постійно зберігається на серверах в мережі Інтернет і тимчасово кешується на клієнтській стороні, наприклад, на персональних комп'ютерах, ігрових приставках, ноутбуках, смартфонах і т.д.» [34, 62, 82].

За оцінками IDC ринок публічних хмаркових обчислень у 2009 році склав $ 17 млрд - близько 5 % від усього ринку інформаційних технологій. Згідно з прогнозами до 2016 року ринок хмаркових послуг досягне позначки в $ 83 млрд.

Крім того, за даними консалтингових компаній понад 30 % підприємств у всьому світі вже розгортають, принаймні, одне хмаркове рішення.

Розглянемо основні причини, через які хмаркові системи стали доступними.

1. Розвиток багатоядерних процесорів привів до:

- збільшення продуктивності, при тих же розмірах обладнання;
- зниження вартості обладнання, як наслідок експлуатаційних витрат;
- зниження енергоспоживання хмарних систем.

2. Збільшення ємкостей носіїв інформації, зниження вартості зберігання 1 Мб інформації дозволило:

- збільшити об'єми збереженої інформації;
- знизити вартість обслуговування сховищ інформації, значно збільшивши об'єми збережених даних.

3. Розвиток технології багато-поточного програмування привело до:

- ефективного використанню обчислювальних ресурсів багатопроцесорних систем;
- гнучкого розподілу обчислювальних потужностей хмарок.

4. Розвиток технологій віртуалізації привело до:

- створення програмного забезпечення, яке дозволяє створювати віртуальну інфраструктуру незалежно від кількості наданих апаратних ресурсів;
- легкість масштабування і нарощування систем;
- зменшення витрат на адміністрування хмаркових систем;
- доступність віртуальної інфраструктури через мережу Інтернет.

5. Збільшення пропускної здатності мереж привело до:

- збільшення швидкості роботи з хмарковими системами, зокрема віртуальний графічний інтерфейс і робота з віртуальними носіями інформації;
- зниження вартості Інтернет трафіку для роботи з великими обсягами інформації;
- проникнення хмаркових обчислень в маси.

Усі вище перераховані фактори привели до підвищення конкурентоспроможності хмаркових обчислень в IT сфері.

У основі Cloud Computing лежить декілька підходів:

1. Доступність через Інтернет. Бувають, також, закриті системи, але, як правило, до всього можна проникнути через мережі.

2. Віртуалізація. Завдяки віртуалізації, користувачі отримують стільки ресурсів, скільки їм потрібно.

3. Cloud Computing – це послуга. Хмарка для користувача – це деякий набір послуг, які споживаються і оплачуються, деколи без щонайменшої уяви про те, що ж використовується усередині.

4. Простота і стандартність. Хоча хмарки і стоять на передньому краю комп'ютерних технологій, у них немає жодних нових мов програмування, складних конфігураційних файлів і багатогодинних сесій в терміналі для налаштування всіх демонів.

Національним інститутом стандартів і технологій США при описі хмаркових технологій використовують принцип «4-3-2». Перша принципу

визначає обов’язкові характеристики хмаркових сервісів [35]:

- Самообслуговування на вимогу (self service on demand), споживач самостійно визначає і змінює обчислювальні потреби без взаємодії з представником постачальника послуг;
- Об’єднання ресурсів (resource pooling), постачальник послуг об’єднує ресурси для обслуговування великої кількості споживачів в єдиний пул для динамічного перерозподілу потужностей між споживачами в умовах постійної зміни попиту на потужності;
- Еластичність (elastic), послуги можуть бути надані, розширені, звужені в будь-який момент часу, без додаткових витрат на взаємодію з постачальни-ком, як правило, в автоматичному режимі;
- Облік споживання (usage based), постачальник послуг автоматично обчислює спожиті ресурси на певному рівні абстракції, і на основі цих даних оцінює обсяг наданих споживачам послуг.

З точки зору постачальника, завдяки об’єднанню ресурсів та непостійному характеру споживання з боку споживачів, хмаркові технології дозволяють економити на масштабах, використовуючи менші апаратні ресурси, ніж при виділенні апаратних потужностей для кожного споживача, а за рахунок автоматизації процедур модифікації виділення ресурсів істотно знижуються витрати на абонентське обслуговування.

З точки зору споживача, ці характеристики дозволяють отримати послуги з високим рівнем доступності (high availability) і низькими ризиками непрацездатності, забезпечити швидке масштабування обчислювальної системи завдяки еластичності без необхідності створення, обслуговування і модернізації власної апаратної інфраструктури [11].

Згідно згаданого принципу «4-3-2», друга цифра характеризує три основних методи постачання хмаркових сервісів: Infrastructure-As-A-Service, Platform-As-A-Service і Software-As-A-Service (рис. 1.1):

- IaaS – набір зв’язаних з інфраструктурою можливостей (операційна система, мережеві підключення і т.д.), які надаються клієнту на основі моделі «оплати-за-використання» і можуть використовуватися для розміщення

додатків. Найбільшими гравцями на ринку інфраструктури як послуги є Amazon, Microsoft, VMWare, Rackspace та Red Hat. Хоча деякі з них пропонують більше, ніж просто інфраструктуру, їх об’єднує мета продавати базові обчислювальні ресурси.

- PaaS – функціональність більше високого рівня, яка зв’язана з платфор-мою і надається як сервіс для розробників додатків. З PaaS розробники абстрагуються від нище лежачої інфраструктури. Наприклад, Google Apps на-дає застосунки для бізнесу в режимі онлайн, доступ до яких відбувається за допомогою Інтернет-браузера тоді як програмне забезпечення і дані зберігаються на серверах Google [61].
- SaaS – програмне забезпечення, яке пропонується в якості сервісів, коли організації просто споживають і використовують додатки. Прикладами програмного забезпечення як послуги, що працює на основі обчислювальної хмарки, є сервіси Gmail [29] та Google docs [37].

З зростанням рівня методу постачання хмаркових послуг зростає й оглядова цінність для кінцевого користувача, і, відповідно, кількості абонентів даного виду послуг (рис. 1.2).

Хоча на сьогодні й існує широка таксономія термінів, які звужують контекст, але в цілому все зводиться до цих трьох методів (сервісів). Ці три типи сервісів можуть працювати окремо або в комбінації один з одним.

Остання цифра в принципі «4-3-2» характеризує тип хмарки. Хоча тип хмарки впливає на розміщені в ньому сервіси достатньо опосередковано – для кінцевого користувача використання сервісу, який розміщений в приватній хмарці чи розміщений в публічній, може не нести жодної різниці – використання практично завжди повністю прозоре. Аналогічно до методів постачання, існують додаткові терміни, що характеризують тип хмарки, наприклад, Community Cloud, але дані типи так чи інакше є або розвитком, або симбіозом приватного чи публічного типів.

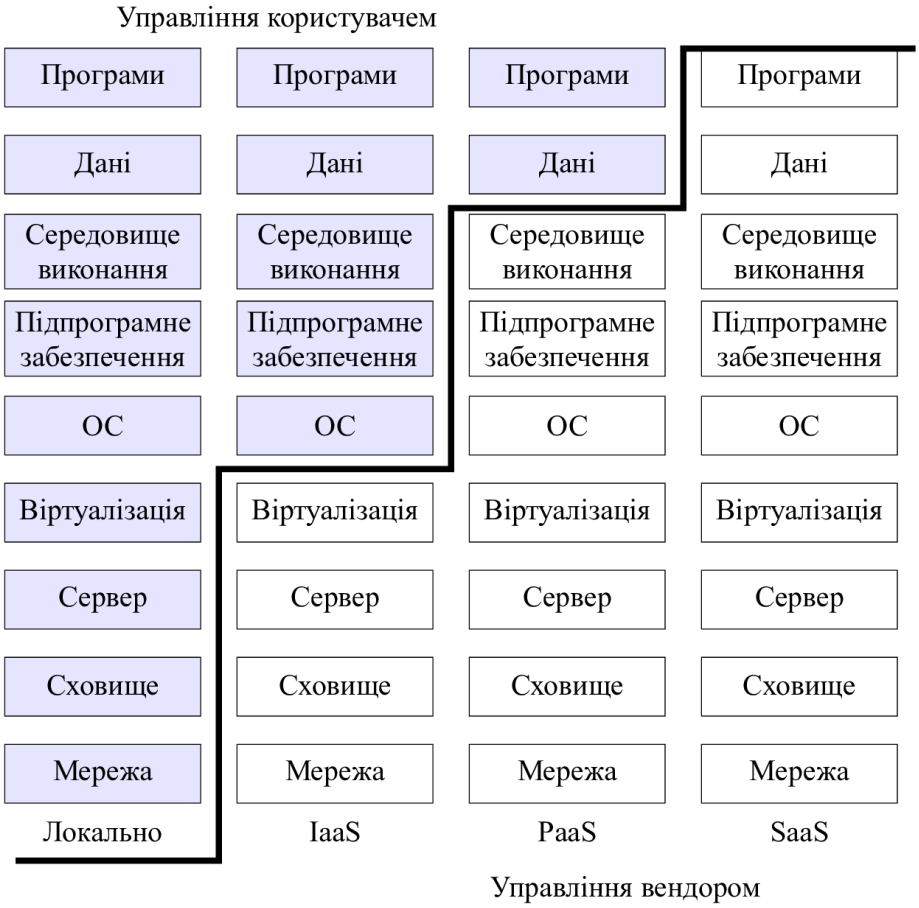


Рис. 1.1. Рівневе представлення хмаркових сервісів, побудовано за даними [27].

Хоча на сьогодні й існує широка таксономія термінів, які звужують контекст, але в цілому все зводиться до цих трьох методів (сервісів). Ці три типи сервісів можуть працювати окремо або в комбінації один з одним.

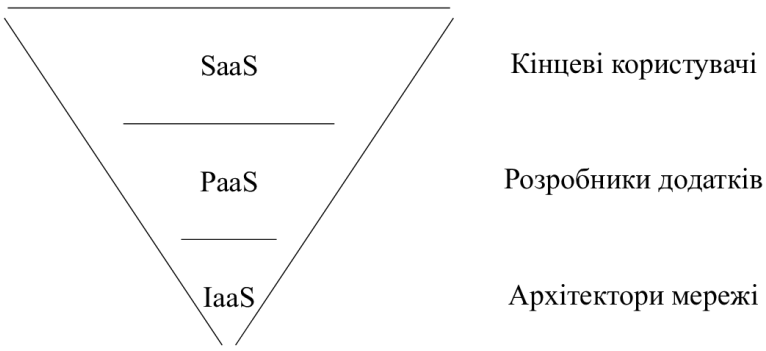


Рис. 1.2. Моделі роботи з хмаркою для різних груп користувачів, побудовано за даними [27].

Остання цифра в принципі «4-3-2» характеризує тип хмарки. Хоча тип хмарки впливає на розміщені в ньому сервіси достатньо опосередковано –

для кінцевого користувача використання сервісу, який розміщений в приватній хмарці чи розміщений в публічній, може не нести жодної різниці – використання практично завжди повністю прозоре. Аналогічно до методів постачання, існують додаткові терміни, що характеризують тип хмарки, наприклад, Community Cloud, але дані типи так чи інакше є або розвитком, або симбіозом приватного чи публічного типів.

Таким чином, хмаркова технологія охоплює окрім самих обчислень, такі області, як зберігання даних, інформаційні послуги, інтеграцію, безпеку, організація певних процесів і управління. Причому один і той же провайдер може пропонувати будь-який перелік послуг. Користувач же може отримувати весь спектр послуг у одного провайдера, а може скористатися і різними провайдерами для певних рі зновидів сервісів. Наявне різноманіття провайдерів і сервісів істотно ускладнює процес вибору для користувача.

До переваг концепції хмаркових сервісів для провайдерів можна віднести:

- Розвиток додатків і програмних засобів проміжного рівня (middleware) привів до можливості використання віртуальних ресурсів.
- Зниження вартості і підвищення масштабованості обчислювальних ресурсів дозволяє створювати потужні віртуальні машини. Це звільняє замовників від необхідності купувати і підтримувати апаратні засоби, а можуть користуватися віртуальними машинами, що працюють в «мережевій хмарці».
- Послуги, доступ до яких надається через «мережеву хмарку», мають вели-кий набір конкурентних переваг в порівнянні з іншими типами керованих послуг.
- Cloud Computing створює платформу для стандартних керованих послуг, які можуть пропонуватися на нішевих вертикальних ринках, що включають ринок малих підприємств.

Серед переваг Cloud Computing для замовників можна виділити:

- Можливість зниження накладних витрат, пов’язаних з підтримкою

апаратних і програмних компонентів (залежить від типа пропонованих послуг).

- Зниження сукупної вартості володіння у випадку оплати за фактом використання дає можливість замовникові починати роботу з системою з невеликих об’ємів і збільшувати об’єм використовуваних ресурсів в міру необхідності. Такий підхід дозволяє запобігти великим капітальним витратам на початковому етапі проекту.
- Швидке і просте придбання нових послуг і прискорення процесів виходу на ринок.

Приведений огляд показав усі різноманіття використання хмаркових технологій з точки зору користувача. Але при підході до проектування таких сервісів необхідно враховувати й іншу точку зору – точку зору розробника. Тому потрібно детально проаналізувати архітектурні вирішення хмаркових технологій.

1.2. Аналіз архітектур хмаркових сховищ даних.

Хмаркова система зберігання даних, або зберігання даних як послуга – це абстрактне поняття, яке відповідає системі зберігання даних, яку можна адмініструвати за вимогою через спеціальний інтерфейс. Цей інтерфейс абстрагує місцезнаходження системи, так що локальна вона чи віддалена, або гібридна – не має значення. Хмаркові інфраструктури зберігання даних утворюють нові архітектури, які підтримують різні рівні обслуговування поверх потенційно великої групи користувачів і географічно розподілених накопичувачів [13].

Важливе значення має здатність клієнта контролювати і управляти тим, як зберігаються його дані, і пов’язаними з цим витратами. Численні постачальники хмаркових послуг пропонують засоби управління, які забезпечують користувачам підвищений контроль над витратами.

Ефективність зберігання даних – важлива характеристика хмаркової інфраструктури зберігання, особливо враховуючи її акцент на загальну

економію. Щоб зробити систему зберігання ефективнішою, потрібно зберігати більше даних. Загальним рішенням є скорочення обсягу вихідних даних, щоб вони займали менше фізичного простору. Два способи досягнення цієї мети: стиснення – упаковка даних шляхом їх кодування з використанням різних представлень – і дедуплікація – виключення всіх дублікатів даних. Хоча обидва методи корисні, стиснення передбачає обробку (перекодування даних в інфраструктуру і з неї), а дедуплікація – обчислення сигнатур для пошуку дублікатів.

Одна з найбільш особливостей хмаркового зберігання даних - здатність забезпечити економію. Це економія на придбанні накопичувачів, їх енергопостачанні, ремонті, а також на управлінні зберіганням. Якщо розглядати хмаркове зберігання з цієї точки зору (включаючи SLA і підвищену ефективності зберігання), воно може виявитися вигідним при певних моделях використання (рис. 1.3). API доступу до сховища є найважливішим компонентом послуг. Багато додатків вимагають доступу до сховища послуг з використанням API, який оптимізований для цієї конкретної системи зберігання даних, або на власному обладнанні або хмарним. Так хмаркова система зберігання Amazon S3 API надає розробникам SDK для .Net і Java, а також бібліотеки для додаткових платформ і мов. Ці інтерфейси зазвичай використовують протоколи веб–сервісів передачі репрезентативного стану (REST) та/або простий протокол доступу до об’єктів (SOAP) [39].

При розгляді архітектури потрібно враховувати її робочі параметри. Під ними розуміють різні характеристики архітектури, враховуючи вартість, продуктивність, можливість віддаленого доступу і т.п.

Архітектура хмаркового зберігання даних – це насамперед доставка ресурсів зберігання даних на вимогу в високо-масштабованому і мультитенантному середовищі. Узагальнено архітектура хмаркового зберігання даних представляє собою зовнішній інтерфейс, який надає API для доступу до накопичувачів (рис. 1.4). У традиційних системах зберігання

даних це протокол SCSI, але в хмарці появляються нові протоколи. Серед них можна знайти зовнішні протоколи Web-сервісів, файлові протоколи і, навіть, більш традиційні зовнішні інтерфейси (Internet SCSI, iSCSI та ін.). За зовнішнім інтерфейсом розташовується рівень проміжного програмного забезпечення – логіка зберігання даних. Цей рівень реалізує ряд функцій, таких як реплікація даних і скорочення обсягу даних, за традиційними алгоритмами розміщення даних з урахуванням географічного розташування. Нарешті, внутрішній інтерфейс організовує фізичне зберігання даних. Це може бути внутрішній протокол, який реалізує специфічні функції, або традиційний сервер з фізичними дисками.

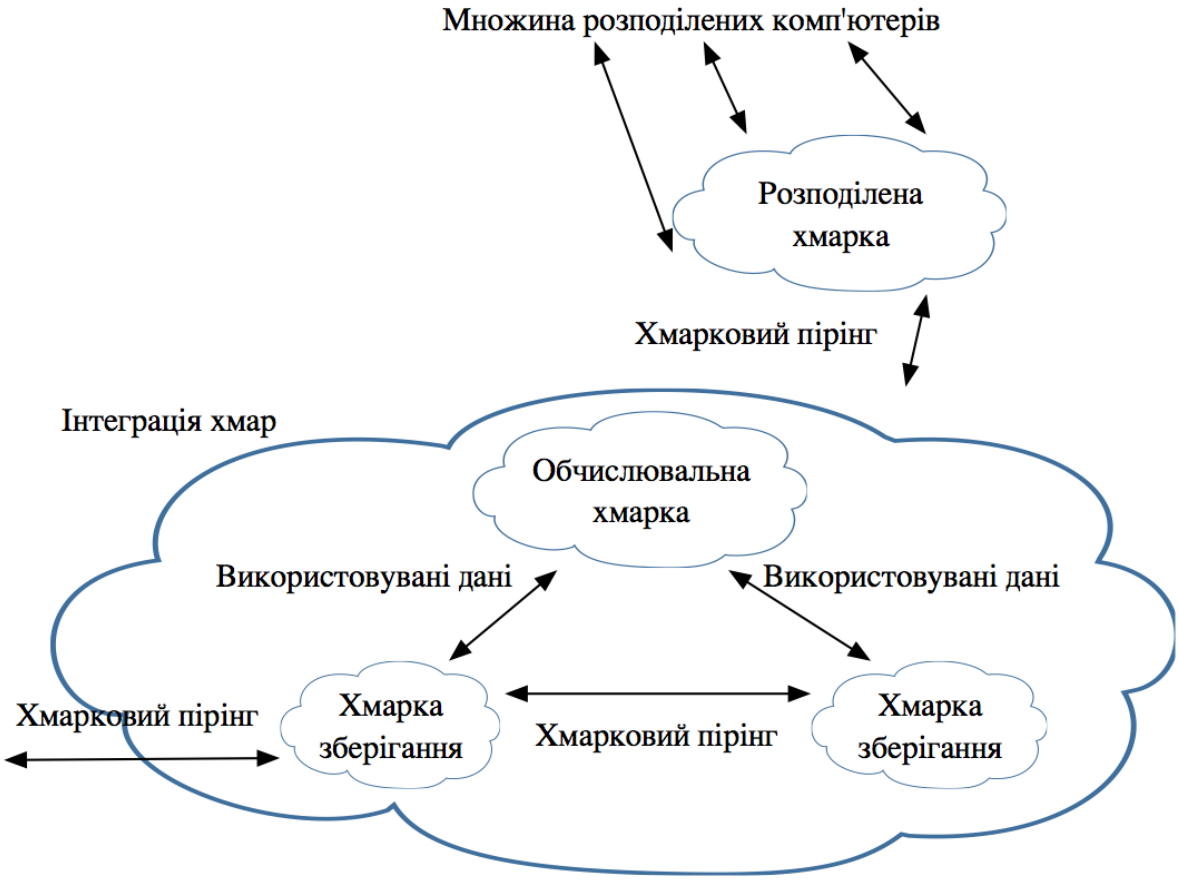


Рис. 1.3. Моделі хмаркової екології, побудовано за даними [69].

Представлена загальна архітектура (див. рис. 1.4) дозволяє виділити деякі характеристики сучасної архітектури хмаркового зберігання даних. Ці характеристики не належать винятково до певного рівня, а можуть відповідати декільком (табл. 1.1).

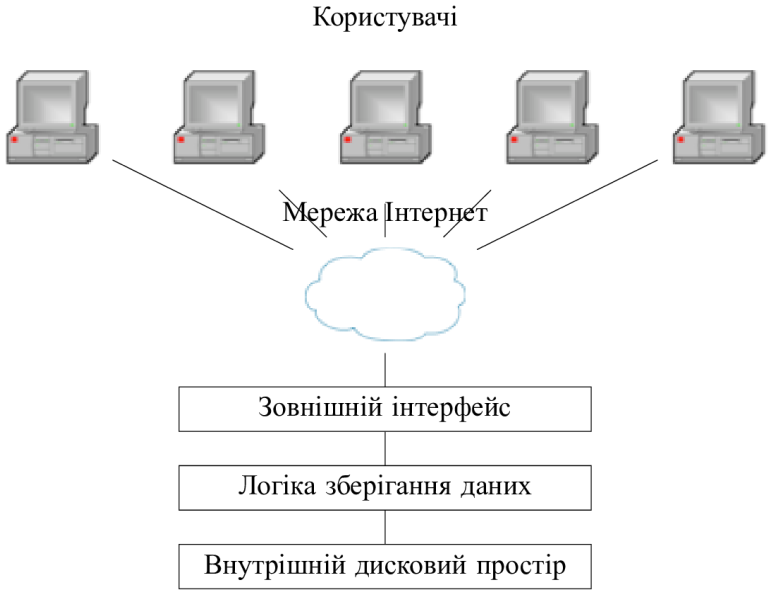


Рис. 1.4. Архітектура хмаркового зберігання даних, побудовано за даними [69].

Однією з найяскравіших відмінностей між хмарковою та традиційною системами зберігання є засоби доступу до них (рис. 1.5). Більшість постачальників пропонує різні методи доступу, однак загальноприйнятими є API Web-сервіс. Більшість з них реалізовані на принципах REST, основою якого є об'єктно-орієнтована схема, що розроблена поверх протоколу HTTP (з використанням TCP в якості транспорту).

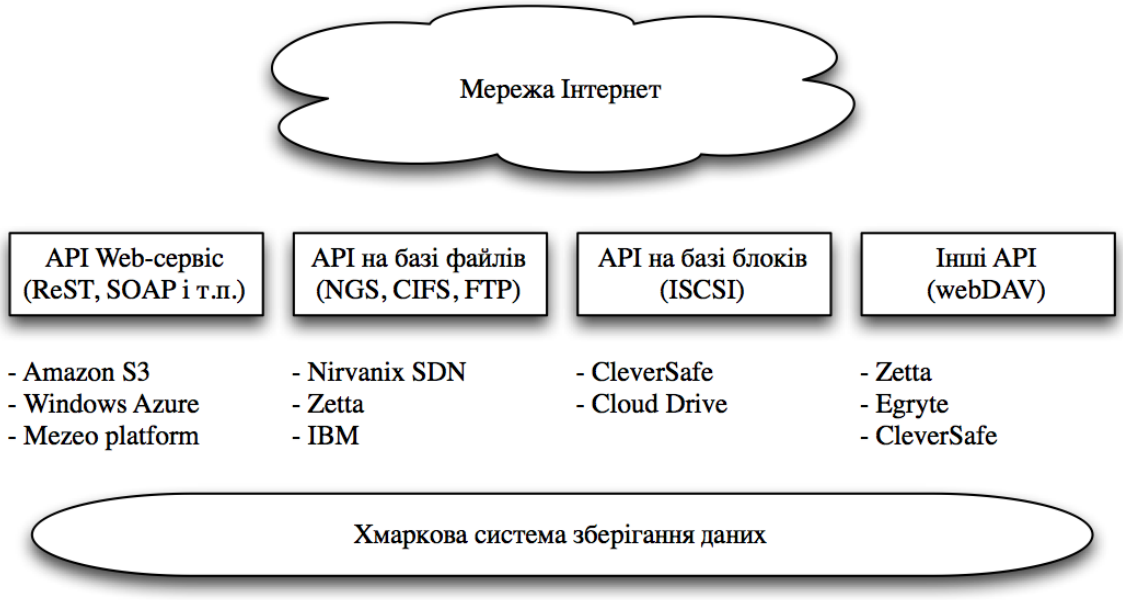


Рис. 1.5. Методи доступу до хмаркових систем зберігання даних, побудовано за даними [69].

Таблиця 1.1. Характеристики архітектури хмаркового зберігання даних, побудовано за даними [69].

Характеристика	Опис
Керованість	Здатність керувати системою при наявності мінімальних ресурсів
Метод доступу	Протокол, через який надаються послуги хмаркового зберігання даних
Продуктивність	Вимірюється пропускною здатністю і часом затримки
Багатокористувацькість	Підтримка багатьох користувачів (арендаторів)
Масштабованість	Можливість поступового нарощування для задоволення нових вимог, або обробки підвищеного навантаження
Готовність даних	Вимірюється часом безвідмовної роботи системи
Управління	Можливість управляти системою – зокрема, вибираючи вартість, продуктивність, або інші характеристики
Ефективності зберігання	Міра ефективності використання накопичувачів
Вартість	Міра вартості зберігання даних (часто в доларах за гігабайт)

Принципи побудови REST-API без запам’ятовування станів – прості та ефективні. Їх реалізують багато постачальників хмаркових послуг зберігання, зокрема Amazon Simple Storage Service (Amazon S3), Windows Azure і Mezeo Cloud Storage Platform.

Одна проблема API Web-сервісів полягає у тому, що для того щоб скористатися перевагами хмаркової системи зберігання, вони вимагають інтеграції з додатком. Тому з хмарковими системами зберігання для забезпечення безпосередньої інтеграції використовуються також загальні методи доступу. Наприклад, протоколи на основі файлів, такі як NFS/Common Internet File System (CIFS) або FTP, або протоколи на основі блоків, такі як iSCSI. Такі методи доступу надають Nirvanix, Zetta, Cleversafe та інші постачальники послуг хмарного зберігання.

Вищезазначені протоколи найбільш поширені, але для хмаркового зберігання підходять й інші. Один з найцікавіших – Web-based Distributed Authoring and Versioning (WebDAV). WebDAV також базується на HTTP і дозволяє використовувати Web в якості ресурсу для читання і запису. Серед популярних постачальників, що використовують WebDAV, є Zetta, Cleversafe та інші.

Можна знайти й такі рішення, які підтримують кілька протоколів доступу. Наприклад, IBM Smart Business Storage Cloud дозволяє використовувати протоколи на основі файлів (NFS і CIFS) і протоколи на основі SAN в одній і тій же інфраструктурі віртуалізації систем зберігання даних.

Проте, при використанні різних протоколів обміну даними між користувачами та хмарковими сховищами виникають багато перешкод для забезпечення оптимального надання якісних послуг. В основному, усі ці проблеми пов’язані з непристосованістю існуючої структури мережі Інтернет для надання таких послуг.

Можна розглядати проблеми надання якісних послуг хмаркового зберігання даних з різних поглядів.

Для поняття продуктивності існує багато аспектів, але головне завдання хмаркової системи зберігання даних – це переміщення даних між користувачем і віддаленим постачальником хмаркових послуг. Проблема криється в транспортному протоколі TCP, який є головним робочим протоколом Інтернету. TCP управляє потоком даних на основі підтвердження прийому пакетів з віддаленого вузла. Втрата або затримка пакетів дозволяють управляти перевантаженням, що ще більше обмежує продуктивність для уникнення глобальних мережевих проблем. TCP ідеально підходить для переміщення невеликих обсягів даних через глобальну мережу Інтернет, але не для доставки великих обсягів даних – у цьому випадку час обміну даними (RTT) збільшується.

Amazon за допомогою програмного забезпечення Aspera Software вирішила цю проблему шляхом виключення рівня TCP. Новий протокол Fast and Secure Protocol (FASP) був розроблений для прискорення передавання різних даних в умовах великого часу відгуку і втрати пакетів. Базою служить UDP, що є допоміжним транспортним протоколом TCP. UDP дозволяє керувати заторами, передаючи цей аспект до протоколу прикладного рівня FASP (рис. 1.6).

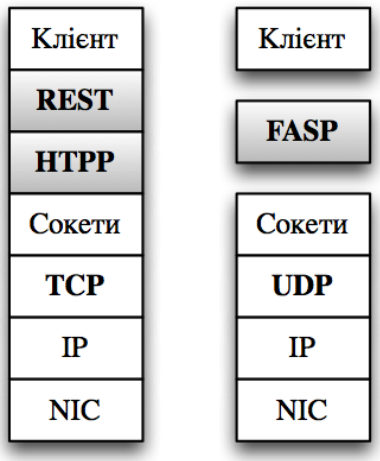


Рис. 1.6. Архітектура протоколу Fast and Secure Protocol, побудовано за даними [69].

Працюючи зі стандартними мережевими адаптерами (без прискорення), FASP ефективно використовує доступну для додатків смугу пропускання і виключає головні вузькі місця традиційних схем масової передачі даних.

Одна з ключових особливостей архітектури хмаркового сховища є її багатокористувацькість (сховище використовується багатьма користувачами). Ця особливість відноситься до різних рівнів хмаркової системи зберігання – від рівня додатку, де користувачам виділяється ізольований простір імен, до рівня зберігання, де окремим користувачам, або категоріям користувачів, можуть виділятися окремі фізичні накопичувачі. Багатокористувацькість поширюється навіть на мережеву інфраструктуру, яка з’єднує користувачів з накопичувачами, забезпечуючи гарантовану якість обслуговування і виділену смугу пропускання для конкретного користувача.

Масштабованість можна розглядати з кількох точок зору, але стосовно сховищ даних розглядається як виділення хмаркових ресурсів зберігання на вимогу.

Можливість динамічної зміни ресурсів зберігання (як в сторону збільшення, так і зменшення) означає поліпшену економічну ефективність для користувача і підвищену складність для постачальника хмаркових послуг.

Масштабованість повинна забезпечуватися не тільки для самої системи зберігання (функціональне масштабування), але й для її пропускної здатності (масштабування навантаження). Ще одна ключова особливість хмаркового зберігання – географічний розподіл даних (географічна масштабованість), яка дозволяє розташовувати дані в максимальній близькості до користувача завдяки групі центрів хмаркового зберігання даних (шляхом міграції). У випадку даних «тільки для читання» можливі також реплікація і розповсюдження (рис. 1.7).

Коли постачальник хмаркових послуг зберігає дані користувача, він повинен мати можливість повернути ці дані користувачеві на вимогу. З урахуванням простоїв мережі, помилок користувачів та інших обставин виконання цієї умови надійним і детермінованим способом може виявитися складним.

Останнім часом появилися нові схеми забезпечення високої готовності – розосередження інформації. Цей алгоритм (Information Dispersal Algorithm – IDA) підвищує доступність даних при фізичних відмовах і простоях мережі за рахунок «нарізки» даних за допомогою кодів Ріда-Соломона для їх відновлення у випадку втрати частини даних. Як і RAID, IDA дозволяє відновлювати дані з підмножини вихідних даних при деяких накладних витратах на коди помилок (рис. 1.8).

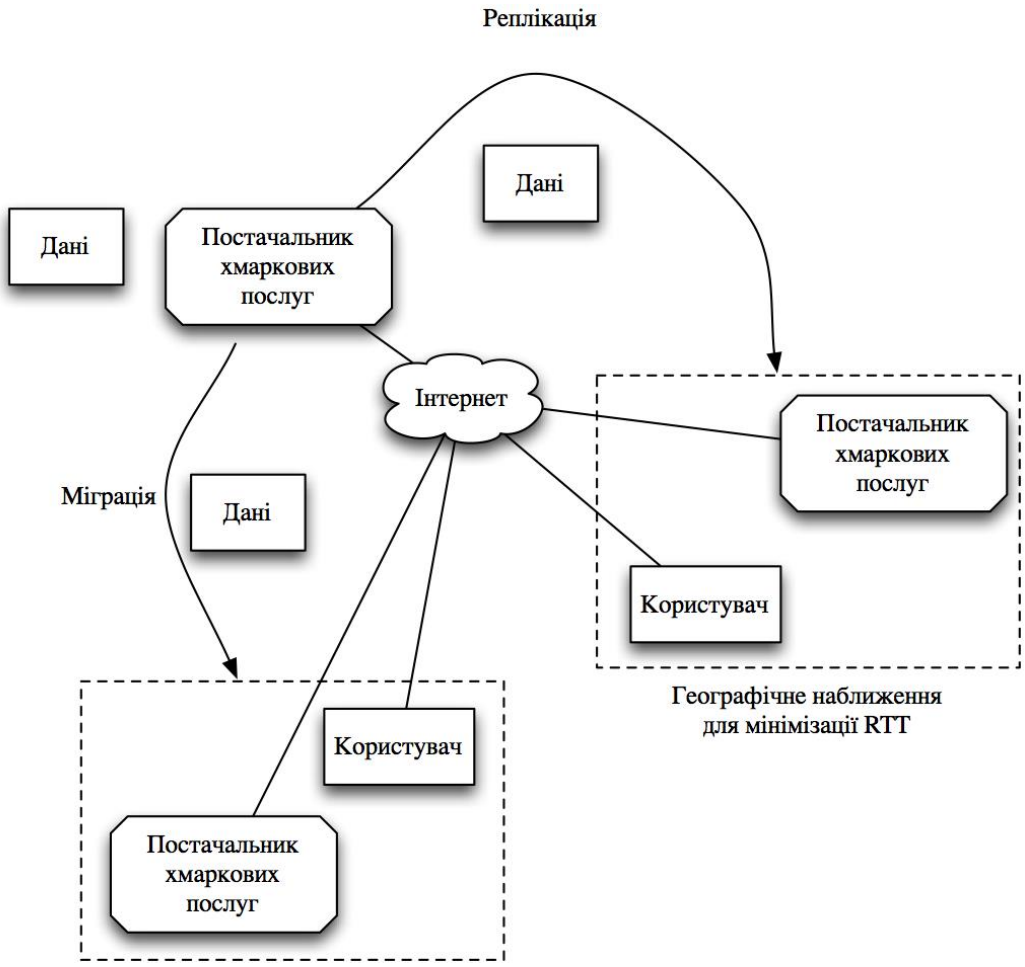


Рис. 1.7. Масштабованість хмаркової системи зберігання даних, побудовано за даними [69].

Можливість нарізати дані із застосуванням кодів корекції Ріда-Соломона дозволяє географічно розподіляти накопичувачі. При кількості часток p і допустимій кількості збоїв m результуючі накладні витрати становлять $p/(p-m)$. Так, у випадку, показаному на рис. 1.8, накладні витрати для системи зберігання при $p=4$ і $m=1$ складають 33 %.

Зворотний бік IDA – інтенсивна обробка без апаратного прискорення. Реплікація – ще один корисний метод, який використовують багато постачальників хмаркових послуг. Він простий і ефективний, хоча і накладні витрати достатньо великі (100 %).

Зворотний бік IDA – інтенсивна обробка без апаратного прискорення. Реплікація – ще один корисний метод, який використовують багато постачальників хмаркових послуг. Він простий і ефективний, хоча і накладні

витрати достатньо великі (100 %).

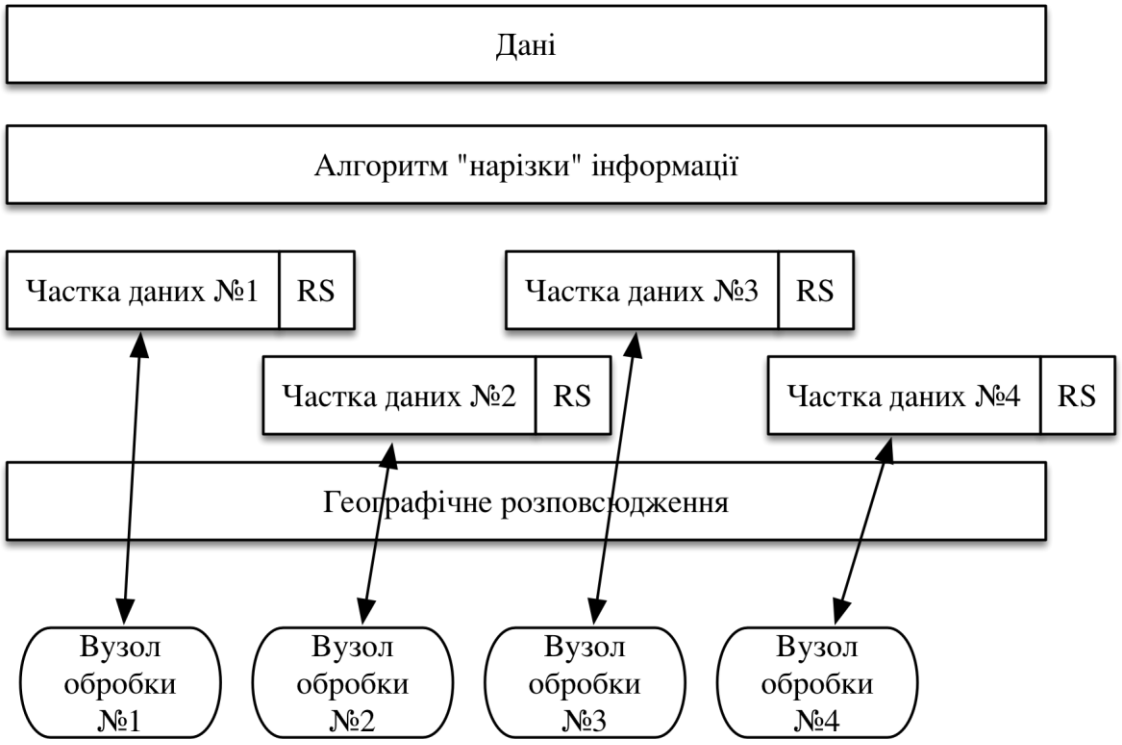


Рис. 1.8. Підхід Cleversafe до забезпечення високої готовності даних, побудовано за даними [69].

До тепер було розглянуто головним чином постачальників хмаркових послуг зберігання даних, але існують хмаркові моделі, які дозволяють користувачам зберігати контроль над своїми даними. Хмаркове зберігання розвивається в трьох напрямках, одне з яких допускає злиття двох інших для досягнення економічної ефективності та безпеки [24].

Постачальники загальнодоступних хмаркових систем зберігання даних надають інфраструктуру на умовах оренди (ресурси для довгострокового або короткострокового зберігання даних і смугу пропускання мережі). Приватні хмарки використовують ті ж концепції, що й загальнодоступні, але в такій формі, що інфраструктура може бути надійно вбудована в приватну мережу користувача. Нарешті гібридні хмарні системи зберігання дозволяють з’єднати обидві моделі, визначаючи правила, що регулюють те, які дані необхідно зберегти в приватному володінні, а які можна захистити в рамках публічних хмар (рис. 1.9).

Хмаркове зберігання даних – це цікавий напрямок розвитку моделей зберігання, який відкриває нові можливості для побудови, доступу та адміністрування систем зберігання даних на підприємствах та установах. Хоча сьогодні хмаркова система зберігання – переважно споживча технологія, вона швидко розвивається в напрямку підприємств. Гібридні моделі хмар дозволять організаціям зберігати конфіденційність своїх даних у межах локальних центрів обробки даних, передаючи менш конфіденційні дані в хмарку для економії витрат і географічного захисту.

Рис. 1.9. Хмаркові моделі зберігання даних, побудовано за даними [69].

Враховуючи усі різноманітність архітектурних рішень для хмаркових сховищ та широкий горизонт для використання виникають проблеми впровадження таких систем в бізнес.

Не дивлячись на інтерес до концепції «Cloud Computing» і пов’язаним з нею послугами, в галузі існують побоювання з приводу відсутності в цій області загальноприйнятих стандартів і методів забезпечення гарантованої якості обслуговування.

Опитування IT-керівників показало, що переважна їх більшість вважають безпеку важливою проблемою для Cloud Computing. Друге і третє місце у ряді проблем зайняли продуктивність і доступність.

Перед тим, як замовник зможе без всяких побоювань перейти до використання технології Cloud Computing, необхідно визначити стандарти безпеки, перенесення додатків між «хмарковими» платформами, угод про рівень обслуговування (SLA – Service-level agreement) і вирішити деякі інші питання [94]. Перераховані стандарти дозволять замовникам використовувати будь-які необхідні йому поєднання додатків, платформ і ресурсів.

До основних областей концепції Cloud Computing, які потребують стандартизації можна віднести:

- Програмне забезпечення проміжного рівня (Cloud middleware або Cloud OS) – базова система для управління послугами. Прикладами такого про-грамного забезпечення можна назвати Google App Engine і Amazon EC2/S3. Це програмне забезпечення повинно дозволяти користувачам розміщува-ти запити на створення екземплярів «хмаркових систем» (cloud instances), отримувати доступ до ресурсів і управляти їх життєвим циклом.
- Інтерфейси API для додатків, що надають доступ до таких ресурсів, як обчислювальні потужності, засоби зберігання і системи управління образами віртуальних серверів. Ці інтерфейси необхідні для роботи додатків в «мережевих хмарках». В даний час більшість постачальників пропонують власні несумісні інтерфейси API, що не дозволяють спільно використовувати ре-сурси різних мереж. До цих пір багато мережевих розробників обмежують свою роль в цій області функціями базового рівня, ринок ж вимагає тіснішої інтеграції мережевих інфраструктур за допомогою інтерфейсів API.
- Управління ресурсами – найважливіша область, що вимагає розвитку архітектури Cloud Computing. Зазвичай мережеві ресурси надаються користу-вачеві статичним чином, тоді як в середовищі Cloud Computing обчислювальні потужності, засоби зберігання і додатка повинні

надаватися динамічно, за запитом. Мережеві ресурси повинні надаватися окре-мо і незалежно від прикладних ресурсів. У типовому корпоративному IT-середовищі існує багато адміністративних доменів, які ізольовані один від одного і працюють абсолютно незалежно. Але в середовищі Cloud Computing, де віртуальні системи створюються за запитом, динамічно, в мінімальні терміни і так же швидко розформовуються, адміністративні бар'єри стають великою проблемою, яка різко уповільнює виділення ресурсів.

- Технологія віртуалізації використовується на ринку вже не перший рік і швидко поширюється в центрах обробки даних і в операторів зв'язку. При цьому існує декілька несумісних між собою технологій віртуалізації. Відсутність загальних стандартів затруднює створення загальнодоступного середовища Cloud Computing, сумісною з іншими «мережевими хмарами» і широким спектром обчислювальних і інформаційних ресурсів.

- Взаємодія між «мережевими хмарками» вимагає стандартного рівня управління, що забезпечує взаємодію мереж і спільне використання обчислювальних і комунікаційних ресурсів, що належать різним власникам. Прикладом такого підходу може слугувати рівень управління, вбудований в програмне забезпечення проміжного рівня (cloud middleware) або в розширення протоколу BGP (Border Gateway Protocol). Використання таких протоколів, як XMPP (Extensible Messaging and Presence Protocol), як стандартний інтерфейс між мережами різних операторів дозволить різним «мережевим хмаркам» використовувати єдині функції присвоєння імен і контролю доступності, а також єдині правила призначеного для користувача доступу, що, у свою чергу, сприятиме створенню сумісних послуг.

- Динамічне управління політиками на різних рівнях необхідне додаткам, що працюють в «мережевій хмарці», так само як додаткам, що працюють в корпоративному середовищі. Коли користувач за допомогою послуги Cloud Computing створює екземпляр «хмарки», він повинен визначити правила високого рівня для всіх ресурсів, що входять до складу

цього екземпляра. Правила управління мережевими ресурсами мають бути погоджені з правилами, розробленими для додатків.

Усі проблеми хмаркових технологій безпосередньо пов’язані з їх особливостями: віддаленістю, розподільністю, паралелізмом, абстрагованістю. Насправді, нічого поганого в цьому немає. Будь-яке явище, або предмет, має як позитивні, так і негативні сторони. Доводиться або миритися з недоліками, або взагалі відмовлятися від використання конкретного продукту. І хмарні обчислення не є винятком.

Отже, одна з основних проблем – це перенесення існуючих додатків в «хмарку». По-перше, це не завжди можливо зробити через особливості архітектури конкретного додатка, його прив’язок до інших систем або сервісів, які ще не перенесені в «хмарку». Найчастіше перехід до хмарних обчислень неможливий через використання специфічних API ОС, або виклик низькорівневих функцій для оптимізації роботи. Таких додатків, можливо, не дуже багато, але вони є. Бувають і такі випадки, коли перенесення теоретично можливе, але це вимагає або значної переробки коду, або переписування всього набору програмного забезпечення з нуля.

Дуже часто це економічно невигідно. На справді абсолютно аналогічна ситуація спостерігалася зовсім недавно, коли відбувався перехід на 64-розрядність і багатоядерність. Особливо складно йдуть справи з багатоядерністю. Не так багато є програм, які оптимізовані для роботи з декількома ядрами або потоками. Лічені відсотки від усього розмаїття світового програмного забезпечення при тому, що «найпростіші» 2-ядерні процесори існують на ринку не перший рік. Що вже тоді говорити про «хмарки», в яких можна запускати тисячі потоків одночасно.

Нарешті, є цілий клас важких додатків, які взагалі важко кудись перенести – відеоредактори і конвертери, 3D-ігри, CAD-системи і тривимірні редактори, програми реального часу. Втім, всі ці проблеми частково можна вирішити. Для розробників створюється багатий набір API, пропонуються нові моделі і парадигми програмування, розробляються інструменти для

спрощення роботи з багатопотоковістю.

І це ще одна проблема Cloud Computing – необхідність в постійному підключенні до інтернету. З врахуванням вітчизняних реалій цю проблему можна назвати першорядною. Часткові вирішення вже давно існують. Наприклад, Google Gears дозволяє працювати з GoogleDocs або Gmail в оффлайн-режимі, а при першому ж підключенні синхронізує все з сервером [110]. Правда, в Google офіційно відмовилися від подальшої підтримки цієї технології і уповають на розширені можливості HTML5 по роботі з локальними даними. Microsoft в рамках парадигми Software + Services пропонує комбінований підхід – частина даних зберігається і обробляється локально, а інше віддається «хмарі». Але в даному випадку наголос все ж робиться на вирішення іншої проблеми – безпеки.

За ідеєю, хмаркові обчислення тим і хороші, що дані зберігаються в розподіленому вигляді і періодично архівуються. Однак ніхто не застрахований від збоїв.

Подібні прецеденти вже були, наприклад, з Gmail. Тоді вдалося відновити інформацію більшості акаунтів, але частина все-таки пропала назавжди. Одна справа – особисте листування, а зовсім інше – бізнес-інформація. У першому випадку це не критично, а от другий може загрожувати як мінімум вимушеним простоєм і втратою прибутку. Не випадково багато бізнес-користувачі вважають за краще зберігати все на власних серверах з налагодженою системою бекапів. Зрештою, навіть якщо відбудеться збій, і накопичувачі ушкодяться, то їх хоча б можна буде отримати і віднести на відновлення у відповідну контору. А якщо таке станеться десь в хмарі, то все залежатиме вже від совісті cloud-провайдера. Особливо приємно бачити в EULA горезвісний відмову від відповідальності – мовляв, ніщо в світі не надійно. Виходом може стати тільки підтримка актуальної копії всіх даних поза хмарками.

Отже, одна з основних проблем масового запровадження хмаркових

технологій пов’язана з транспортуванням великих обсягів даних між географічно розподіленими серверами та клієнтами. Існуючі методи транспортування досить обмежують якість надання послуг через великі часові затрати саме на транспортування, тому потрібно провести аналіз існуючих методів транспортування, щоб виявити їх вузькі місця.

1.3. Аналіз рівнів транспортування даних в розподілених системах.

Швидке збільшення пропускної здатності комп’ютерних мереж і поява нових розподілених додатків є двома мотивами досліджень і розробок мереж. З одного боку, пропускна здатність мережі розширена до 10 Гбіт/с з появою 40 Гбіт/с, що дозволяє транспортувати багато даних інтенсивним програмам і було неможливо в минулому. З іншого боку, нові програми, такі як обробка наукових розподілених даних, швидко розгортаються в високошвидкісних глобальних мережах.

Національні та міжнародні високошвидкісні мережі підєднали найбільш розвинені регіони у світі волоконними лініями зв’язку [107]. Дані можуть передаватися зі швидкістю до 10 Гбіт/с між цими мережами, і часто з більш високою швидкістю всередині самих мереж. Наприклад, у Сполучених Штатах, є національні мульти-10 Гбіт/с мережі, такі як Lambda National Rail, Internet2/Abilene, TeraGrid, ECBT і інші. Вони можуть підключатися до багатьох міжнародних мереж, таких як канадська CA Net, нідерландська SURFnet, великобританська UKLight та японська JGN/JGN2.

Між тим, об’єми даних зростають експоненціально. Старий спосіб зберігання даних на дисках і доставки їх транспортними засобами вже не ефективний. У багатьох ситуаціях це старомодний спосіб доставки дисків з даними унеможливлює задоволення вимог додатків.

Можливості високопродуктивних серверних систем і комунікаційних мереж значно прискорили розвиток розподілених обчислень. Розподілені додатки спочатку характеризувалися за базовими вимогами до зв’язку, які були в основному пов’язані з надійністю і впорядкованістю. На сьогодні,

мультимедійні додатки та хмаркові технології дуже широко використовуються і вимагають певних гарантованих характеристик часу доступу та пропускної здатності мережі. Ці нові обмеження частково вирішуються на рівнях застосувань і мереж, але адаптація транспортного рівня залишається необхідною. На транспортному рівні механізм управління перевантаженням є одним з ключових механізмів для цього.

На жаль, високошвидкісні мережі не можуть оптимально використовуватися безпосередньо програмним забезпеченням. Протокол управління передачею (TCP), який є де-факто транспортним протоколом Інтернету, недостатньо використовує пропускні здатності мережі при високошвидкісних з'єднаннях при великих затримках [44, 71, 108]. Наприклад, один потік TCP з настройками параметрів за замовчуванням на Linux 2.4 може досягати швидкості тільки близько 5 Мб/с на каналі 1 Гбіт/с.

У дев'яності роки численні дослідження в галузі мережевих технологій були націлені на поліпшення якості обслуговування (QoS), яке пропонується кінцевим користувачам. Ці дослідження привели до розробки нових мережевих архітектур.

Проблеми в розгортанні цих архітектур зробили їх недоступними для кінцевого користувача. Деякі з цих архітектурах були стандартизовані в IETF, наприклад інтегроване обслуговування (IntServ) [31], диференційоване обслуговування (DiffServ) [28] і багатопротокольна комутація за мітками (MPLS) [33]. Останнім часом деякі проекти, такі як проект EuQoS [28], запропонували нову архітектуру на основі віртуалізації мережевих ресурсів і послуг для забезпечення QoS для кінцевого користувача.

Тим не менше, всі ці пропозиції були вивчені і досліджені без врахування верхнього пласту мережевої архітектури і, зокрема, транспортного рівня, який спрямований на застосування ефективної адаптації між прикладними потребами і станом мереж та послуг. Це відсутність зв'язку між мережним і транспортним рівнем призвело до нездатності ефективно обслуговувати потреби повним стеком протоколів, які

б дозволили забезпечити суворий контроль QoS.

Комп’ютерний зв’язок часто представляють у вигляді структурованої пластової моделі [22]. Кожен пласт виконує певне завдання. Пласти покладені один на одного.

Пласт надає послуги іншому пласту, який знаходиться вище, і використовує послуги нижчого пласту. Якщо не змінювати функціональність послуг і інтерфейси між пластами, то можна змінювати самі пласти і при цьому система в цілому продовжує функціювати. Наприклад, додаток для Web-браузера буде працювати так само, коли комп’ютер підключений за допомогою кабелю Ethernet, як і при бездротовому підключенні. Це відбувається тому, що пласт обробки доступу до середовища надає ті ж послуги пласту вище, незалежно від дротового або бездротового середовища передачі даних.

Існують дві основні моделі пластів. Модель ISO Взаємодії відкритих систем (OSI) [79, 22] визначає сім пластів: програма, презентація, сесія, транспорт, мережа, передача даних і фізичний рівень. Для Інтернету використовується більш компактна рівнева модель визначена IETF. Було визначено [42, 22] чотири пласти: додаток, транспорт, мережа і доступ до мережі. У порівнянні з моделлю OSI, використовувані пласти передбачають охоплення рівнів представлення і обробки. Представлення моделі OSI знайшла в основному академічне використання, в той час як IETF (Internet Engineering Task Force) модель TCP/IP де-факто стала набором протоколів, які використовується в Інтернеті.

Розглядаючи багаторівневу архітектуру TCP/IP, можна виділити в ній, подібно до архітектури OSI, рівні, функції яких залежать від конкретної технологічної реалізації мережі, і рівні які не залежать від технологій мережі, функції яких орієнтовані на роботу тільки з додатками.

У стеку TCP/IP визначені чотири рівні – прикладний рівень, основний (транспортний) рівень, рівень міжмережевих взаємодій, рівень мережевих інтерфейсів. Кожен із них несе на собі деяку долю навантаження при

вирішенні основної задачі – організації надійної і продуктивної роботи складеної мережі, частини якої побудовані на основі різних мережевих технологій.

Стрижнем всієї архітектури є рівень міжмережевої взаємодії, або мережевий рівень, який реалізує концепцію передачі пакетів в режимі без встановлення з'єднань, тобто дейтаграмним способом. Саме цей рівень забезпечує можливість переміщення пакетів мережею, використовуючи той маршрут, який в даний момент є найбільш раціональним. Цей рівень також називають рівнем Інтернет, вказуючи, тим самим, на основну його функцію – передачу даних через складену мережу.

Протоколи двох нижніх рівнів є мереживо-залежними, а відповідно, програмні модулі протоколів міжмережевого рівня і рівня мережевих інтерфейсів встановлюються як на кінцевих вузлах складеної мережі, так і на маршрутизаторах.

Так як стек TCP/IP був розроблений до появи моделі взаємодії відкритих систем OSI, то, хоча він також має багаторівневу структуру, відповідність рівнів стеку TCP/IP рівням моделі OSI достатньо умовна (табл. 1.2).

Таблиця 1.2. Відповідність рівнів стеку TCP/IP і моделі OSI, побудовано за даними [22].

<table>
<tr><td>OSI</td><td></td><td></td><td></td><td></td><td></td><td></td><td>TCP/IP</td></tr>
<tr><td>7</td><td>WWW</td><td>SNMP</td><td>FTP</td><td>Telnet</td><td>TFTP</td><td>SMTP</td><td rowspan="2">I</td></tr>
<tr><td>6</td><td>Gopher, WAIS</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>5</td><td></td><td>TCP</td><td></td><td></td><td>UDP</td><td></td><td rowspan="2">II</td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>IP</td><td>ICMP</td><td>RIP</td><td colspan="2">OSRF</td><td>ARP</td><td>III</td></tr>
<tr><td>2</td><td colspan="6">не регламентується: Ethernet, Gigabit Ethernet,</td><td rowspan="2">IV</td></tr>
<tr><td>1</td><td colspan="6">Token Ring, PPP, FDDI, X.25, SLIP, frame relay …</td></tr>
</table>

Додатки використовують послуги транспортного рівня для зв’язку з іншими кінцевими точками. Транспортний рівень може забезпечити надійну передачу потоку даних через мережу з низькою надійністю пакетами. У моделі IETF протокол TCP забезпечує цей надійний потік передачі даних. Існує також пакет на основі ненадійного транспортного протоколу UDP, а також інші більш спеціалізовані транспортні протоколи.

Транспортний рівень поділяє потік даних на пакети і використовує мережевий рівень для доставки їх до кінцевої точки зв’язку. На приймальному кінці, транспортний рівень відповідає за збір пакетів з мережевого рівня і складанням їх у потік даних для додатку. Мережевий рівень забезпечує маршрутизацію пакетів даних від одного кінця до іншого. Він зберігає таблицю маршрутизації, яка вказує, який вузол повинен використовуватися для даного призначення, і переключає вхідні пакети в цьому напрямку. Зрештою, пакети досягають кінцевого пункту призначення і поставляються на транспортний рівень приймального вузла.

Традиційно кожен пласт має доступ до безпосереднього пласту зверху і знизу за допомогою стандартних інтерфейсів. Додаток може отримати доступ до транспортного рівня, транспортний рівень може отримати доступ до додатка і мережевого рівня, і так далі. При управлінні в середині пластів забезпечується доступ до більш гнучких способів передачі. Аналогічно може бути досягнуто при управлінні між найближчими пластами або між пластами, які є ізольовані. Цей доступ може включати в себе контроль або обмін інформацією поза пластами програмних інтерфейсів. Дослідження міжрівневої взаємодії дозволяє виявити ефекти, які існують за рахунок поперечної взаємодії пластів. Наприклад, обидва пласти транспортного і канального рівня можуть або не можуть виконувати повторні передачі, що впливає на поведінку і продуктивність іншого пласту. Одна з цілей цієї роботи полягає в дослідженні поперечної взаємодії пластів між прикладним і транспортним рівнями.

У багаторівневій архітектурі Інтернет, транспортний рівень є четвертим пластом, який знаходиться між мережевим і прикладним рівнями (рис. 1.10). На даний час існують два транспортні протоколи на цьому рівні: TCP (Transmission Control Protocol) і UDP (User Datagram Protocol), однак з'являються нові протоколи, в тому числі SCTP [98] і DCCP [45]. TCP є надійним протоколом потокових даних, орієнтований на з'єднання, тоді як UDP-з'єднання – менш ненадійний протокол обміну повідомленнями.

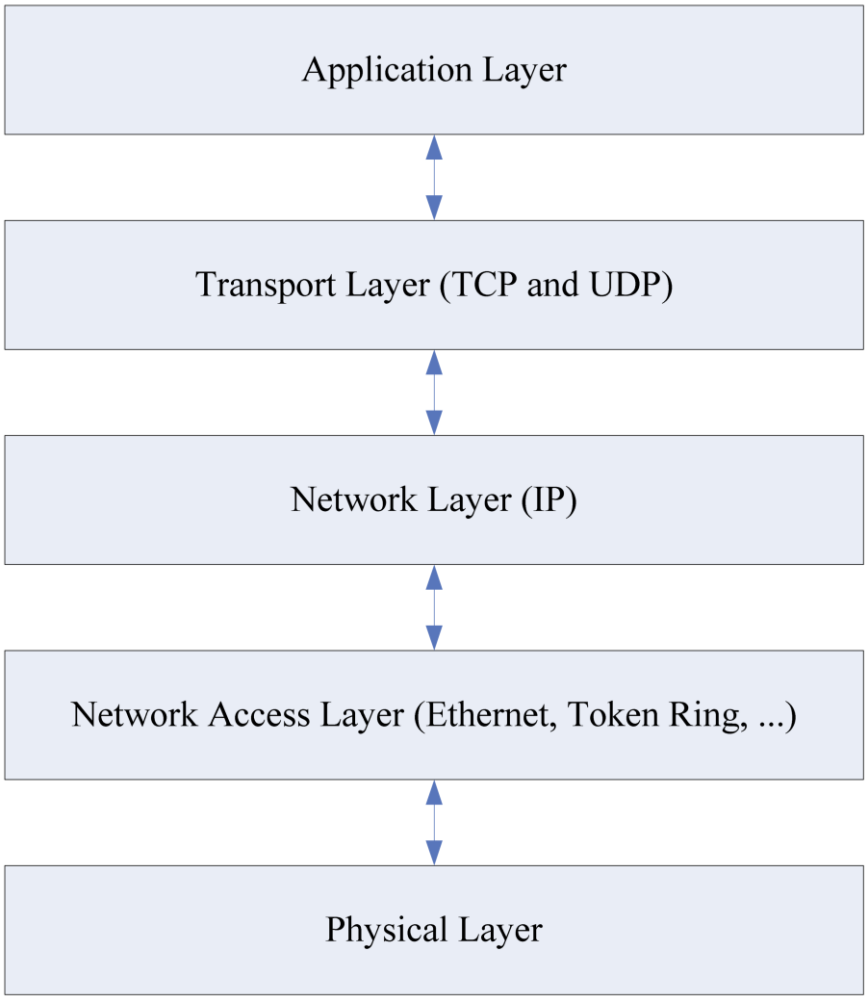


Рис. 1.10. Рівні архітектури протоколів передачі Інтернету, побудовано за [22].

Тим не менш, протягом цього часу було внесено декілька змін до основної архітектури Інтернету (наприклад, транспортний і мережевий рівень). Еволюція двох крайніх пластів протоколу Інтернет в поєднанні з незмінністю основних протоколів TCP/IP говорить про неможливість ефективної адаптації нових сервісів між додатком і фізичним рівнем.

Сучасний стан пропускної здатності стеку протоколів Інтернету можна зобразити у вигляді пісочного годинника, який демонструє вузькі місця стеку протоколів (рис. 1.11).

Транспортний рівень відіграє велику роль – адаптер між прикладним і фізичним рівнем, оскільки він локалізований в кінцевих системах а, отже, його легше розгортати ніж на мережевому рівні. У транспортному рівні механізм управління перевантаженням є одним з основних механізмів. Контроль перевантаження дозволяє уникати колапсів в Інтернеті і різні потоки поділяють доступні смуги пропускання. Проте, реальні механізми управління перевантаженням акцентують увагу тільки на перевантаженнях мережі і не беруть до уваги ні потреби додатків, ні нові мережеві послуги.

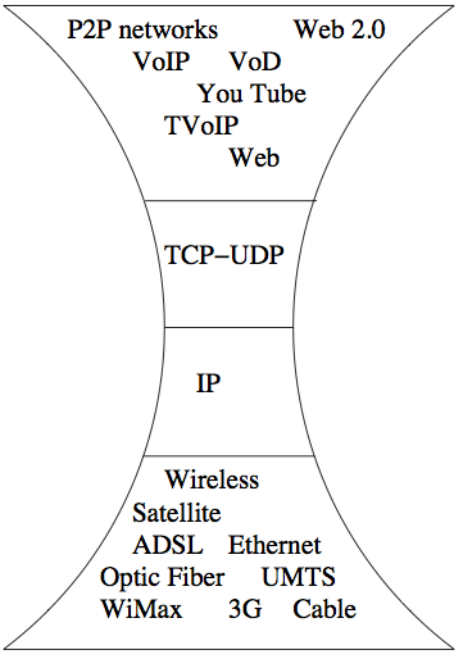


Рис. 1.11. Сучасний стано протоколів Інтернету, побудовано за [22].

Транспортні протоколи також управляють трафіком як в глобальних мережах, так і в локальних і корпоративних мережах. Згідно [106], близько 95 % всіх переданих байтів і 85-95 % всіх переданих пакетів в Інтернеті передаються за допомогою протоколу TCP. Цей протокол на сьогодні є основним протоколом транспортного рівня в архітектурі TCP/IP. Головною задачею цього протокола є забезпечення надійної передачі даних через ненадійне середовище передачі.

Транспортний рівень представляє собою групу методів і протоколів у багаторівневій архітектурі компонентів мережі, які відповідають за інкапсуляцію блоків даних, – дейтаграми і сегменти – що передаються через мережеву інфраструктуру на вузол, або управляють передачею. Таким чином, протоколи транспортного рівня встановлюють пряме транспортне середовище між віртуальними вузлами або кінцевими додатками. Цей рівень також згадується як транспортні протоколи [106].

Транспортний протокол надає різні функціональні можливості для додатків, підтримує, але не обмежує доставку даних, контролює достовірність даних і сервіс потоків або повідомлень. Транспортні протоколи загального призначення мають чотири основні цілі, які, як правило, прозорі для додатків: ефективність, справедливість, збіжність та розподільність.

Транспортний протокол має бути ефективним, або використовувати доступну смугу пропускання якомога ефективніше. Щоб бути ефективним, протокол повинен виконувати наступні два завдання в короткі терміни:

- намагатися досягнути максимально доступної смуги пропускання
- відновити швидкість до максимальної після падіння швидкості передачі через перевантаження, або втрати пакетів.

Після досягнення максимальної швидкості, він повинен залишатися в поточному стані, поки не зміниться стан мережі, тобто коливання повинні бути якомога меншими.

Очікується, що пропускна здатність мережі буде розподілятися справедливо між усіма одночасними потоками. Вимірювання справедливості може мати різні стандарти. Найбільш поширеним з них є $\min - \max$ справедливість, метою якої є забезпечити максимально мінімальну пропускну здатність. Справедливість власності серед всіх потоків, що належать до того ж протоколу називається внутрішньо-протокольна справедливість. Зокрема, незалежність RTT (часу передачі) використовується для опису окремого випадку справедливості на топологіях з різними RTT, яка не задовольняється TCP. Проблема справедливості стає все важчою, коли

співіснують гетерогенні протоколи. Нові транспортні протоколи повинні розглядати ситуації, коли вони співіснують з TCP, перш ніж вони почнуть широко використовується в Інтернеті. Справедливість між TCP і новими протоколами називається TCP дружелюбність.

Швидкість відправлення даних повинна бути врівноваженою для всіх відправних точок, для будь-якої конкретної ситуації в мережі. Прийнято, що пропускна здатність коливається навколо нерухомої точки, тому що зворотні зв'язки, як правило, використовується для повідомлення про зміни ситуації в мережі. Це глобальна властивість стабільності транспортних протоколів.

Оскільки Інтернет є великою слабозв'язаною системою, неможливо мати центральний сервер для задання пропускної здатності. Транспортні протоколи повинні контролювати швидкість своїх даних за кінцевим результатом з або без допомоги з боку маршрутизаторів, через які проходить трафік. Принцип з кінця в кінець стверджує, що, при можливості, операції транспортних протоколів повинні відбуватися тільки в кінцевих хостах. Принцип з кінця в кінець значно збільшує масштабованість системи. Більше того, навіть при існуванні операторів шлюзів, протокол як і раніше повинен мати контроль за перевантаженням на кінцевих хостах [47].

Управління перевантаженнями є важливим компонентом в транспортному протоколі для реалізації його цілей. Транспортний протокол регулює швидкість передачі даних за допомогою певного алгоритму управління перевантаженням. Контроль перевантаження мережі, як правило, це система зі зворотнім зв'язком. Зворотній зв'язок може бути або від проміжних вузлів, таких як маршрутизатори, або може бути оцінений за втратами пакетів, збільшення тенденцій у затримці пакетів, або тайм-ауту подій. Явний зворотній зв'язок з маршрутизаторами надає більш точну інформацію, але й вимагає більш високих обчислювальних затрат і дорогий при розгортанні.

Швидкість відправлення даних може бути налаштована або за часом між пакетами або кількістю розміщених пакетів. Перший спосіб – на основі

управління перевантаженням, другий – на основі управління вікном перевантаженням. Обидва методи можуть бути застосовані одночасно. Лінійна система часто застосовується в схемі управління для налаштування цих параметрів через свою простоту. Найвідоміший алгоритм управління є алгоритм AIMD TCP, або адитивне збільшення, мультиплікативне зниження.

Для вирішення проблеми адаптації були запропоновані для стандартизації нові транспортні протоколи, такі як DCCP [45] або SCTP [98]. Тим не менше, ці нові протоколи пропонують обмежені можливості адаптації і ще не реалізовані в більшості операційних систем. Ці нові протоколи характеризуються використанням дейтаграм-орієнтованого зв'язку замість потоко-орієнтованого зв'язку, як в TCP. Крім того, у випадку DCCP забезпечується новий вид контролю перевантаженням. Цей механізм управління перевантаженням названий TCP Friendly Rate Control (TFRC) дружнє управління швидкістю [48], більше не базується на вікнах, а замість цього використовує формулу для визначення управління швидкістю. TFRC можна розглядати як перший крок для адаптації до мультимедіа додатків з більш гладкою пропускною здатністю, ніж та, що отримана в таких же умовах з TCP-подібним управлінням перевантаженням. Тому цей протокол краще пристосований до потокових даних [40].

Отже, основною задачею є подолання вузьких місць в стеку протоколів TCP/IP, запропонувавши механізми управління перевантаженням, які дозволять транспортним протоколам забезпечувати кращий зв'язок, враховуючи обмеженість кінцевих пристроїв та комунікаційних засобів.

Для проведення дослідження визначення вузьких місць та їх подолання цілком очевидним є побудова математичних моделей, як самих протоколів передачі, так і повної моделі передачі «з кінця в кінець».

1.4. Висновки до 1-го розділу.

У даному розділі:

1. Хмаркова система зберігання даних, або зберігання даних як

послуга – це абстрактне поняття, яке відповідає системі зберігання даних, яку можна адмініструвати за вимогою через спеціальний інтерфейс.

2. Проведено аналіз архітектур систем зберігання даних, досліджено основні підходи до хмаркових технологій, виділено особливості формування архітектури хмаркових сховищ даних і їх проблематику та проведено аналіз транспортування даних в розподілених системах. У процесі аналізу виділено позитивні і негативні риси окремих архітектур зберігання даних. Результати аналізу винесено у Додаток А.

3. Розглянуто хмаркові інформаційні технології з точки зору їх використання для зберігання та поширення даних.

4. При розгляді хмаркових сховищ даних зосереджена увага на їх архітектурних особливостей з точки зору ефективності та доступності для користувачів. Ефективність зберігання даних – важлива характеристика хмаркової інфраструктури зберігання, особливо враховуючи її акцент на загальну економію. Продуктивність має багато аспектів, але головне завдання хмаркової системи зберігання даних – переміщення даних між користувачем і віддаленим постачальником хмаркових послуг.

5. Проведено аналіз існуючих архітектур до реалізації сховищ даних. Приведено поділ їх на приватні, публічні та гібридні.

6. Проаналізовано рівні транспортування даних в розподілених системах. Визначено, що одновний протокол, який використовується для передачі гарантованих даних є TCP, який є головним робочим протоколом Інтернету.

7. Виявленно вузькі місця в організації та транспортуванні даних в хмаркових сховищах даних, на основі чого обгрунтовано постановку задачі для дослідження.

Основні результати розділу опубліковано у працях [10, 11, 12].

РОЗДІЛ 2.
МОДЕЛЮВАННЯ ХМАРНОГО СЕРЕДОВИЩА ДАНИХ І АНАЛІЗ МЕТОДІВ ЇХ ПЕРЕДАВАННЯ

У розділі розроблено модель хмаркового сховища даних та методи передавання даних. Уведено модель гібридних протоколів передавання даних через хмаркове сховище, обгрунтовано модель мережевого трафіку, проведено моделювання завантаженості сховища даних.

2.1. Моделі передачі даних в хмаркових технологіях.

Першою моделлю передачі даних між абонентами була модель типу «точка–точка». Дана модель передбачала виділений канал передачі від абонента, який передавав, до абонента, який приймав. Дана модель застосовувалася в телефонії до 1990-тих років. Саме така модель і була використана для створення перших комп'ютерних мереж.

Мережа типу «точка–точка» – найпростіший вид комп'ютерної мережі, при якому два комп'ютери з'єднуються між собою напряму через комунікаційне обладнання (рис. 2.1). Перевагою такого виду з'єднання є простота і дешевизна, недоліком – з'єднати таким чином можна лише 2 комп'ютери і не більше.

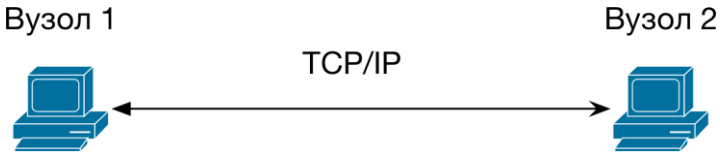


Рис. 2.1. Модель мережі «точка–точка», побудовано за даними [22].

Дана модель з'єднання часто використовується у випадках, коли потрібно швидко передати інформацію з одного комп'ютера на інший.

Варто зазначити, що дана модель може використовуватися, як найбільш узагальнена модель передачі даних між двома об'єктами.

При моделюванні передачі даних від одного абонента до іншого можна використати модель у вигляді графа $G=(V,E)$, вершинами V якого виступатимуть абоненти мережі, а ребрами E – зв'язки між ними. Ребра

матимуть відповідні ваги, наповнення яких визначатиметься необхідністю моделювання. В одному випадку це може бути швидкість передачі інформаційної одиниці, в іншому – якість обслуговування абонента. Але, у будь-якому випадку, ваги визначатимуть узагальнену характеристику тракту передачі даних.

Для подолання недоліків з'єднання типу «точка-точка» було запропонована віртуальна приватна мережа (VPN) [6] з порівняною якістю обслуговування, але при набагато менших витратах. Перемикаючи трафік для оптимального використання каналів вони мали змогу ефективніше використовувати мережу. В такій моделі було вперше використано символ хмари для позначення розмежування між користувачем і постачальником.

Модель VPN – це логічна мережа, створена поверх інших мереж, на базі загальнодоступних або віртуальних каналів інших мереж (Інтернет). Безпека передавання пакетів через загальнодоступні мережі може реалізуватися за допомогою шифрування, внаслідок чого створюється закритий для сторонніх канал обміну інформацією. VPN дозволяє об'єднати, наприклад, декілька географічно віддалених мереж організації в єдину мережу з використанням для зв'язку між ними непідконтрольних каналів.

VPN складається з двох частин: «внутрішня» (підконтрольна) мережа, яких може бути декілька, і «зовнішня» мережа, через яку проходять інкапсульовані з'єднання (зазвичай використовується Інтернет) (рис. 2.2).

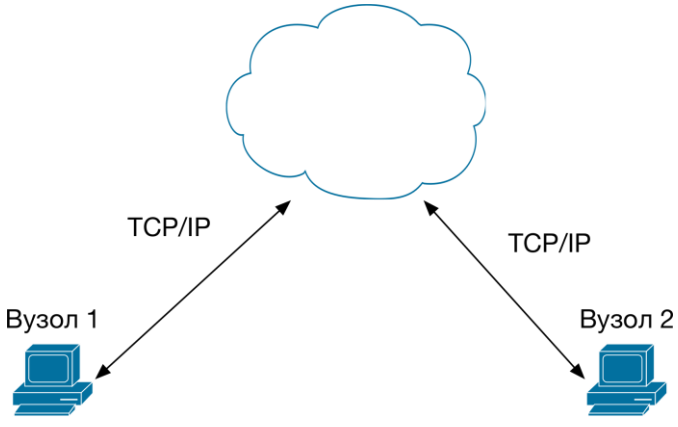


Рис. 2.2. Модель передачі даних через VPN, побудовано за даними [22].

Підключення до VPN віддаленого користувача робиться за допомогою сервера доступу, який підключений як до внутрішньої, так і до зовнішньої (загальнодоступної) мережі. При підключенні віддаленого користувача (або при установці з'єднання з іншою захищеною мережею) сервер доступу вимагає проходження процесу ідентифікації, а потім процесу аутентифікації. Після успішного проходження обох процесів, віддалений користувач (віддалена мережа) наділяється повноваженнями для роботи в мережі, тобто відбувається процес авторизації.

При об'єднанні локальних мереж в загальну VPN мережу можна отримати цілком працездатний загальний простір при мінімальних витратах і високій мірі захисту. Для створення такої мережі знадобиться встановити на одному комп'ютері з кожного сегменту спеціальний VPN-шлюз, який відповідатиме за передачу даних між філіями. Обмін інформацією в кожному відділенні здійснюється звичайним способом, проте у тому випадку, якщо необхідно передати дані на іншу ділянку VPN-мережі, то вони вирушають на шлюз. У свою чергу шлюз здійснює обробку даних, шифрує їх за допомогою надійного алгоритму і передає мережею Інтернет цільовому шлюзу в іншій філії. У точці призначення дані розшифровуються і також передаються на кінцевий комп'ютер звичайним способом.

Усе це проходить абсолютно непомітно для користувача і нічим не відрізняється від роботи в локальній мережі. Окрім цього VPN є оптимальним способом організації доступу окремого комп'ютера в локальну мережу компанії.

Одними з небагатьох недоліків, які має модель VPN, є необхідність закупівлі невеликої кількості устаткування і програмного забезпечення, а також збільшення об'ємів зовнішнього трафіку. Втім ці витрати досить невеликі і враховуючи величезну кількість переваг VPN, з ними цілком можна миритися.

Однак, такі дві моделі передачі даних між абонентами мають один суттєвий – вони вимагають безпосередньої участи абонентів у процесі

передачі даних. Необхідність постійного зв’язку дещо обмежують сфери їх використання.

Позбавлена даного недоліку модель на основі дата-центру (data center, центру зберігання та обробки даних, центру обробки даних) – спеціалізований технічний майданчик для розміщення інформації в мережі Інтернет, підключений до неї в автономну систему (або мережі в її складі) через множину каналів зв’язку.

Дата-цент представляє собою сукупність спланованих певним чином територій, зовнішніх майданчиків, будівель, приміщень, зі встановленими інженерними системами забезпечення та обслуговуючим персоналом, що утворюють загальний фізичний простір і технологічне середовище для розміщення комп’ютерів, електронних та інших засобів прийому, передачі, обробки, зберігання інформації і забезпечують задану ступінь доступності, розміщеного обладнання в заданому режимі функціонування [26].

Створення та експлуатація дата-центрів здійснюється згідно з рядом жорстких стандартів. Дата-центр може бути підрозділом телекомунікаційної компанії або ж окремою організацією. Основна діяльність дата-центру полягає у встановленні та подальшому обслуговуванні серверного та комунікаційного обладнання клієнтів згідно з умовами колокації або виділення серверів. Обладнання монтується до серверних стійок і/або шаф. Якість та пропускна здатність каналів зв’язку дата-центру напряму впливають на якість послуг, які він надає, позаяк основним критерієм її оцінки є час доступності розміщеного в дата-центрі сервера.

Відповідно до TIA/EIA-942 структура центру обробки даних складається з трьох основних підсистем [30]:

1. MDA – Main Distribution Area – головна розподільна підсистема, забезпечує інтерфейс доступу до центру і розподіляє трафік головної магістралі на внутрішні магістралі. Вона включає кінцеве обладнання операторів зв'язку, маршрутизатори, магістральні комутатори;

2. HDA – Horizontal Distribution Area – горизонтальна розподільча підсистема, направляє трафіки внутрішніх магістралей через локальні лінії, що входять в апаратні зони (стійки);

3. EDA – Equipment Distribution Area – підсистема розводки на обладнання, що доставляє трафік в робочі області до серверів, дискових масивів.

Центр обробки даних – це комплексна централізована система, що забезпечує безперервність бізнес–процесів з високим рівнем продуктивності та готовності сервісів, що включає:

- високонадійне серверне обладнання,
- систему зберігання даних,
- активне мережеве обладнання,
- архітектурно-технічні рішення щодо резервування й дублювання критично важливих сервісів інформаційних систем,
- «забезпечувальну» інженерну інфраструктуру,
- фізичний захист приміщень,
- системи управління та моніторингу,
- комплекс організаційних заходів.

При цьому дає вигоди у:

- Мінімізація часу доступу до інформації при будь-якій кількості запитів;
- Збільшення доступного для кожного користувача дискового простору;
- Збільшення доступності даних;
- Відсутність часових витрат на резервне копіювання і відновлення даних;
- Підвищення захищеності системи від збоїв і втрати інформації.

Формально модель обміну даними через дата-центр можна зобразити графічно (рис. 2.3).

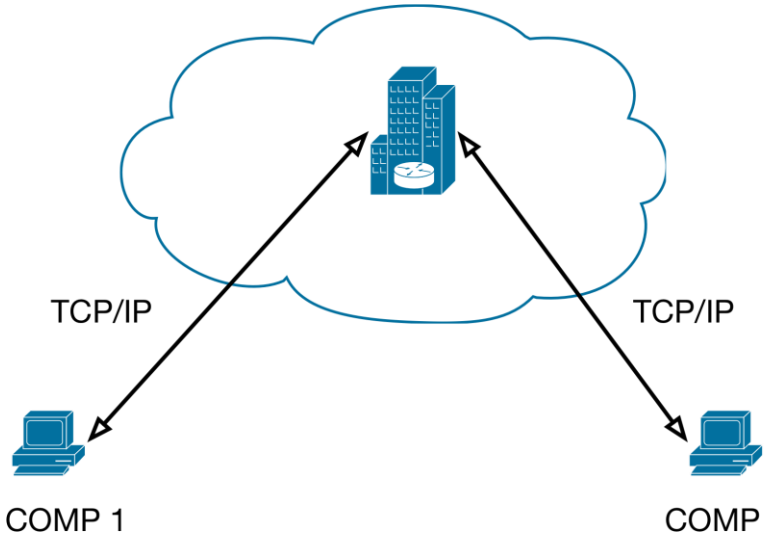


Рис. 2.3. Модель передачі даних через дата-центр, побудовано за даними [22].

Представлення у моделі всього дата-центру одним елементом нічого не говорить про його внутрішню структуру, зате дозволяє моделювати зовнішні зв’язки центру та прослідкувати можливості обміну даними між абонентами, використовуючи центр обробки даних.

Реально, дата-центр містить декілька серверів, які можуть навіть бути розділені регіонально. Але, у будь-якому випадку, є центральні сервери і сателіти.

Файли, якими обмінюються абоненти, фізично зберігаються на основних серверах. Увесь доступ до збереженої інформації відбувається через так звані сателіти.

Основний сервер – сервер на якому фізично знаходяться файли клієнтів.

Основних серверів може бути декілька. Клієнт оплачує зберігання інформації на одному з них, або за додатковою угодою, на декількох. Основна характеристика основного серверу – зберігання великих об’ємів інформації, але повільний доступ до неї.

Узагальнена модель основного сервера містить (рис. 2.4):

1. Firewall – захисний бар'єр сервера.
2. Nginx – Web-сервер для обробки запитів та перенаправлення їх на програмну аплікацію сервера (інтерфейсний модуль основного сервера).

3. Програмна реалізація сервера – програма яка забезпечує авторизацію, автентифікацію, розмежування прав доступу до файлів та сам доступ до файлів.

4. База даних – в якій знаходиться інформація про користувачів, рахунки, файли та ін.

5. Файли – документи користувачів до яких вони мають права доступу.

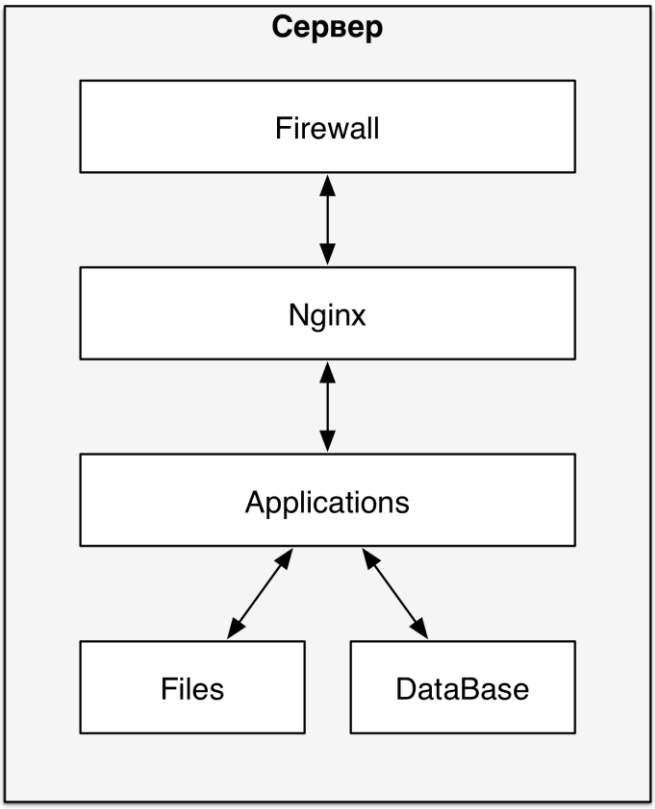


Рис. 2.4. Модель основного сервера, побудовано самостійно.

Основною проблемою є те що сервери знаходяться фізично віддалено від усіх можливих клієнтів. Через що час виконання запиту клієнта досить низький.

Існують рішення дзеркальних серверів для збільшення швидкодії доступу до файлів. Тобто беруться сервери які виконують реплікацію даних один на одний.

Великою проблемою даної архітектурної моделі є актуальність даних. Ще однією проблемою є те що клони сервера повинні бути такої ж конфігурації як і основні, що є досить дорого.

Існує інший підхід до вирішення проблем доступу до дата-центрів. Це, так звана мережа доставки і дистрибуції контенту (Content Delivery Network, або Content Distribution Network, CDN) – географічно розподілена мережна інфраструктура, що дозволяє оптимізувати доставку та дистрибуцію контенту кінцевим користувачам в мережі Інтернет [5]. Використання конвент-провайдерами CDN сприяє збільшенню швидкості завантаження інтернет–користувачами аудіо-, відео-, програмного, ігрового та інших видів цифрового контенту в точках присутності мережі CDN.

На швидкість завантаження сильно впливає те, наскільки далеко користувач знаходиться від сервера. Це відбувається через те, що при використанні технології TCP/IP, що застосовується для поширення інформації в мережі Інтернет, затримки при передачі інформації залежать від кількості маршрутизаторів, що знаходяться на шляху між джерелом і споживачем контенту. Розміщення контенту між декількома серверами засобами CDN скорочує мережевий маршрут передачі даних і робить завантаження сайту швидше з точки зору користувача.

При такій моделі організації хмаркового сервісу збереження даних час передачі між двома абонентами визначатиметься сумою двох часів передачі – від першого до центру і від центру до другого. У випадку коли абоненти і дата-центр знаходяться недалеко один від одного, цей час є невеликим і суттєво на якість обслуговування не впливає. Але враховуючи те, що зв’язок з дата-центром відбувається за протоколом TCP/IP, який є надійним протоколом передачі, але при зв’язках на великих відстанях швидкість гарантованої передачі падає суттєво. Тобто, використання такої моделі для організації передачі даних в континентальних масштабах викликає втрату якості обслуговування.

Не важливо де буде знаходитися сервер дата-центру – біля першого абонента, біля другого, чи десь по середині. У першому випадку дані будуть швидко доставлені на сервер, але довго будуть передаватися з сервера абоненту (рис. 2.5).

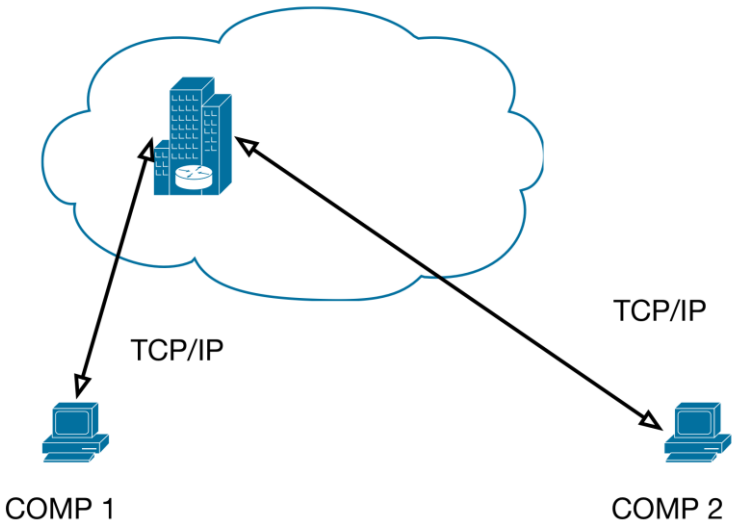


Рис. 2.5. Модель несиметричної передачі даних через дата-центр, побудовано самостійно.

У другому випадку, навпаки, дані довго будуть передаватися від абонента на сервер, але швидко будуть передані з сервера абоненту. Основною проблемою цього є низька швидкість передачі даних стандартним протоколом TCP/IP на великі відстані.

Тому можна розглядати такий підхід, як модель одноточкової дистрибуції.

На противагу цьому можна використати модель багатоточкової дистрибуції, яка предбачає наявність сателітів основних серверів у місцях найбільшої мережевої активності абонентів (рис. 2.6).

Використання такої моделі доставки даних знижує кількість хопів, що істотно збільшує швидкість завантаження контенту з мережі Інтернет. Хоп (hop, стрибок) – назва процесу передачі мережевого пакету (або датаграми) між хостами (вузлами) мережі. Зазвичай використовується для визначення «відстані» між вузлами (чим більше хопів – тим складніший шлях маршрутизації і тим «далі» знаходяться вузли один від одного) [16].

Тобто при зниженні кількості хопів, кінцеві користувачі відчувають меншу затримку при завантаженні контенту, відсутність різких змін швидкості завантаження та високу якість потоку даних. Така стабільність дозволяє операторам дата–центру доставляти відеоконтент у форматі HD,

забезпечувати швидке завантаження файлів великих розмірів або організовувати відеотрансляцію з високою якістю сервісу (QoS) і низькими витратами на мережу.

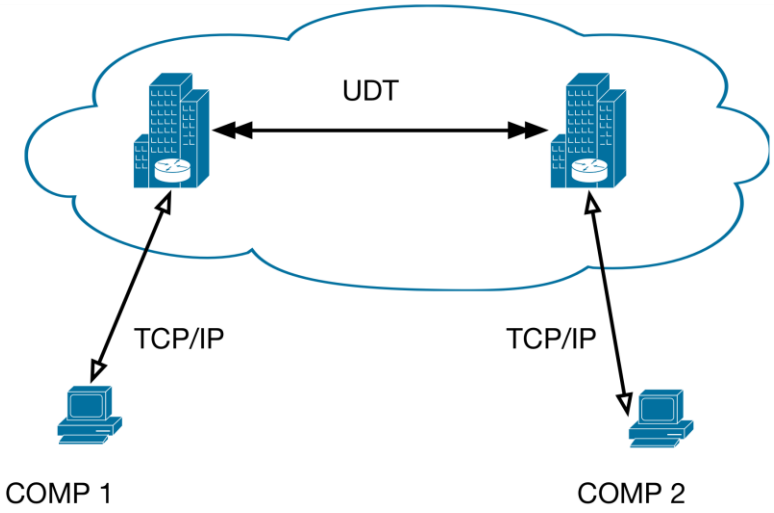


Рис. 2.6. Модель багатоточкової дистрибуції даних через дата-центр, побудовано самостійно.

Модель багатоточкової дистрибуції здатна запобігти затримці при передачі даних, можливим переривання зв'язку і втратам на перевантажених каналах і стиках між ними. Управління навантаженням при передачі мережевого трафіку дозволяє розвантажити магістраль і вузли мережі, розподілити навантаження між віддаленими серверами.

Розміщення серверів в безпосередній близькості від кінцевих користувачів може збільшити вихідну пропускну здатність всієї системи. Наприклад, наявність єдиного порту 100 Мбіт/с не означає дану швидкість на всіх ділянках мережі, так як вільна пропускна здатність магістрального каналу в момент передачі може бути всього 10 Мбіт/с. У випадку, коли використовуються 10 розподілених серверів, сумарна пропускна здатність може скласти 10-100 Мбіт/с.

При такому підході дата-центр складатиметься з географічно розподілених багатофункціональних платформ, взаємодія яких дозволяє максимально ефективно обробляти і задовольняти запити користувачів для отримання контенту.

Однак і тут можливі дві моделі реалізації. При першому підході дані центрального сервера реплікуються на периферійні платформи. Кожна платформа підтримує в актуальному стані повну або часткову копію розповсюджуваних даних. В іншому випадку дані кешуються на сателітах і зберігаються там декілька днів.

Вузол мережі, що входить до складу платформи, взаємодіє з локальними мережами інтернет-провайдерів і поширює контент кінцевим користувачам через найкоротший мережевий маршрут з оптимального за завантаженості сервера. Довжина мережевого маршруту залежить від географічної або топологічної віддаленості користувача комп'ютера від сервера або вартості передачі трафіку в регіоні присутності.

Великі дата-центри можуть складатися з величезної кількості розподілених вузлів і розміщувати свої сервери безпосередньо у мережі кожного локального Інтернет-провайдера. Багато операторів роблять акцент на пропускній спроможності сполучних каналів і мінімальній кількості точок приєднання в регіоні присутності. Незалежно від використовуваної архітектури, головним призначенням подібних мереж є прискорення передачі як статичного контенту, так і безперервного потоку даних.

При такому підході на ключових територіях (зонах) (там де багато клієнтів) розміщують сервери сателіти. Які не зберігають дані і файли, а лише надають швидкий доступ через себе на основні сервери. Для пришвидшення обслуговування клієнтів та додаткового захисту створюються проміжні сервери, час доступу до яких зі сторони клієнтів є меншим, а, отже більша їх швидкодія. Такі сервери називають сателітами.

Сателіт – проміжний сервер на якому немає файлів клієнтів. Він створений для оптимізації передачі даних від основного сервера до клієнта і навпаки. Кількість серверів-сателітів набагато перевищує кількість основних серверів.

Усі сервери розміщені у певних IP-зонах, щоб забезпечити мінімальний час надання послуг клієнтам. Клієнти можуть знаходитися у будь-якій IP-

зоні. Крім того, клієнти можуть бути мобільними – змінювати свої IP адреси, переміщатися з зони в зону. Остання обставина вимагає динамічності у визначені часу доступу до їхньої інформації.

Структурно сателіт складається з трьох частин (Рис. 2.7):

1. Firewall – захисний бар'єр.

2. Nginx – веб сервер для обробки запитів та перенаправлення їх на програмну аплікацію сателіта.

3. Application – Програмна реалізація сателіта – програма яка забезпечує визначення на якому із серверів фізично знаходяться файли користувача та надає доступ до них.

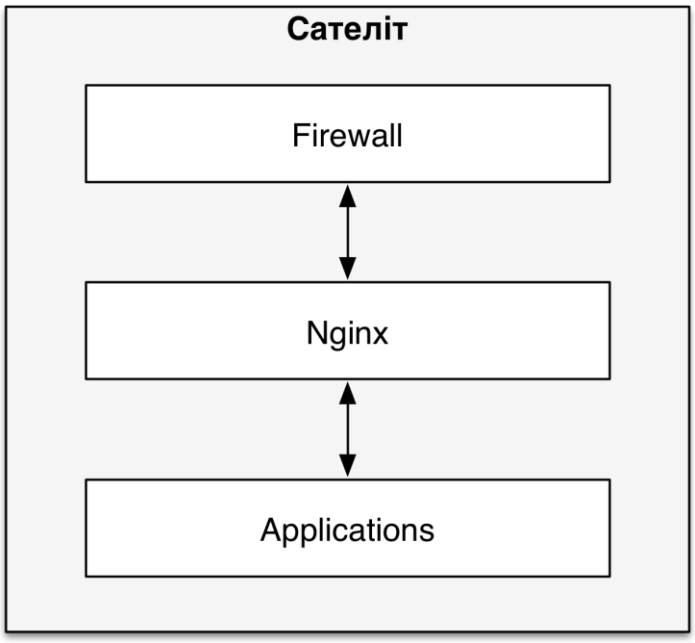


Рис. 2.7. Модель сателіта, побудовано самостійно.

Таким чином, клієнт усю взаємодію з розміщення та отримання своєї інформації проводить безпосередньо через сателіт, незалежно від того на якому основному сервері у нього визначене місце для інформації. Звичайно, якщо клієнт статичний, то для нього визначається сателіт, який найближче розміщений до нього (мінімальна кількість хопів). Але ситуація зміниться при переміщення клієнта в іншу зону, або при технічних неполадках в мережі, або при дуже великому навантаженні на сателіт від різних клієнтів. При цьому втрачається перевага даної моделі.

За основу моделі сховища даних доцільно використати модель сховища даних, запропоновану Робінсоном:

$$S = \langle F, D, G, C, L \rangle, \tag{2.1}$$

де $F = \{f_1, f_2, ..., f_n\}$ - множина елементів даних,

$f = \{p_1, p_2, ..., p_m\}$ – множина пакетів даних,

$D = \{d_1, d_2, ..., d_k\}$ – множина пристроїв зберігання,

$G : F \to D$ – розміщення пристроїв зберігання,

$C : D \to Z$ – ємність пристроїв зберігання,

$L : D \to Z$ – завантаженість пристроїв зберігання.

Для потреб моделювання хмаркових сховищ даних краще використати її масштабований варіант, уведений Петровим:

$$S_m = \langle F, D(t), G, C, L \rangle. \tag{2.2}$$

Тоді модель хмаркового сховища даних подана як:

$$S_{cloud} = \langle D, D_{free}, S_{ms} \rangle, \tag{2.3}$$

де $D_{free} \subseteq D$ – підмножина вільних пристроїв зберігання,

$S_{ms} = \{S_{m1}, S_{m2}, ..., S_{ml}\}$ – множина масштабованих сховищ.

Масштабовані пристрої у даному випадку – це пристрої з множини загальних пристроїв, які не включають в себе підмножину вільних пристроїв $D_i(t) = D \setminus D_{free}$. Масштабовані пристрої не мають спільних пристроїв зберігання: $\forall t, i, j, i \neq j \Rightarrow D_i(t) \cap D_j(t) = \varnothing$.

Удосконалено модель хмаркового сховища як алгебраїчну систему:

$$C_{dw} = \langle S_{cloud\_m}; Y; L \rangle,$$

$$S_{cloud\_m} = \langle D, D_{free}, S_{ms}, PR \rangle,$$

$$Y = \{I_{cc,} I_{mpp}, I_{mpd}\},$$

де I_{cc} – метод вибору шлюзу за складністю запиту,

I_{mpp} – метод мультипротокольної передачі потокових даних,

I_{mpd} – метод мультиплексування різних джерел даних, для одночасної передачі,

PR – протокол передачі даних,

L – предикат завантаженості S_{cloud}.

Елемент даних $f_i \in St \cup SemSt \cup UnSt$ може бути представлений структурованими, слабо-структурованими та неструктурованими даними [95].

Предикат завантаженості хмаркового сховища поданий як відношення завантаженості хмаркового сховища даних у моменти часу t_1 та t_2. Для її визначення досліджуються трафіки даних у хмаркових сховищах, аналізуються об'єднані потоки даних, встановлюється залежність рівня фрактальності сумарного потоку:

$$L\left(S_{cloud\_m_{t1}}, S_{cloud\_m_{t2}}\right) \to Z\,. \tag{2.4}$$

Параметри хмаркового сховища даних S_{cloud_m}: вхідний/вихідний трафік, кількість запущених процесів, завантаженість і простій процесорів, середнє навантаження на процесор та об'єм кеш-пам'яті.

2.2. Організація доступу до хмаркового сховища.

Складність систем зв’язку для передачі даних через хмаркові сховища даних на сьогоднішній день в значній мірі визначається складністю протоколів та їх поєднанням [89], які вони реалізують. Протоколи представляють собою набір правил, у відповідності до яких взаємодіють системи. При розробці систем передачі даних протокол реалізується апаратно або програмно. У зв’язку з тим, що впровадження хмаркових технологій відбувається поверх наявного апаратного обладнання і створювати та замінювати його досить проблематично, для хмаркових технологій, в основному використовують програмну реалізацію протоколів. Специфікації протоколів представлені в стандартах, які є найбільш критичною інформацією при розробці систем і повинні бути коректними та

забезпечувати ефективну взаємодію. Процес розробки протоколів та їх комбінацій стає все більш динамічним (кількість відомих протоколів подвоюється кожні п’ять років). Крім того, зростає складність самих протоколів, що побічно підтверджується зростанням об’єму їх стандартних специфікацій. Таким чином, формальне доведення коректності протоколів та їх поєднань представляє собою важливу наукову проблему.

Формально процес передачі за допомогою гібридного протоколу, або мультипротокольної передачі, можна представити у вигляді переходів:

$$\begin{aligned}
&\langle d_1 f_i \rangle \to \{\langle d_1 p_1 \rangle, \langle d_1 p_2 \rangle, \ldots, \langle d_1 p_m \rangle\} \to \\
&\{\langle c_1, p_1, pr_1 \rangle, \langle c_2, p_2, pr_1 \rangle, \ldots, \langle c_n, p_m, pr_1 \rangle\} \to \\
&\{\langle d_2 p_1 \rangle, \langle d_2 p_2 \rangle, \ldots, \langle d_2 p_m \rangle\} \to \{\langle d_2 p_1 \rangle, \langle d_2 p_2 \rangle, \ldots, \langle d_2 p_m \rangle\} \to, \\
&\{\langle c_1, p_1, pr_2 \rangle, \langle c_2, p_2, pr_2 \rangle, \ldots, \langle c_n, p_m, pr_2 \rangle\} \to \\
&\{\langle d_3 p_1 \rangle, \langle d_3 p_2 \rangle, \ldots, \langle d_3 p_m \rangle\} \to \langle d_3 f_i \rangle
\end{aligned} \qquad (2.5)$$

де $C = \{c_1, c_2, ..., c_m\}$ — множина каналів передачі даних.

Для цілей утворення ефективних хмаркових сховищ даних необхідно забезпечити:

- організацію асинхронного передавання файлів багатьма каналами зв’язку;
- організацію потокового читання файлу і його передавання;
- організацію прийому файлу через декілька каналів зв’язку та його кешування для подальшого запису;
- організацію потокового запису файлу з кешу;
- організацію синхронного підтвердження для завершення передавання файлу.

Як відомо, для організації надання будь-якого сервісу в хмаркових технологіях і доступу до хмаркового сховище, зокрема, необхідна наявність відповідного сховища. Тобто, це сервер, чи мережа серверів, через які сервіс доступу до сховища надається клієнтам.

Самим звичайним спосіб надання сервісу клієнту – це обслуговування

його запитів. Обслужити запит – означає отримати запит і направити стороні, яка створила, відповідь на нього.

Сервер хмаркового сховища має обмежені обчислювальні ресурси, тобто може виконувати лише обмежену кількість операцій за одиницю часу. Але для створення відповіді на запит потрібно провести деяку кількість операцій. Відповідно, сервер сховища може обслужити за одиницю часу лише обмежену кількість запитів, яка визначається обчислювальною потужністю сервера.

Якщо за одиницю часу кількість запитів, які поступають від клієнтів, перевищує обчислювальні можливості сервера, то певна кількість запитів залишиться не обслуженими. Щоб зменшити кількість необслужених запитів, необхідно збільшити обчислювальну потужність сервера сховища.

Найбільш прямолінійний підхід до збільшення потужності сервера сховища – використовувати більш потужні комп’ютери в якості серверів. Це рішення має недоліки: по-перше, в буль-якому випадку потужність комп’ютерів обмежена, а по-друге – вартість такого сховища за рахунок використовуваних комп’ютерів зростає швидше ніж його продуктивність, тобто вартість в два рази більш потужного комп’ютера для сховища зросте більше ніж в два рази.

Інший підхід, який виключає вказані недоліки, полягає в тому, щоб використовувати для обслуговування клієнтських запитів декілька серверів сховища. При цьому вартість рішення буде прямо пропорційна його потужності, і верхня межа на обчислювальну потужність визначається не рівнем апаратних технологій, а кількістю серверів сховища. Проте, при великій розподіленості сховища зростають затрати на передачу даних між сереверами сховища, дані яких повинні копіюватися одночасно на всі сервери сховища.

Згідно до принципів організації глобальної мережі Інтернет, запит, який направляється на певну адресу, може отримати лише один комп’ютер в мережі. Таким чином, для того, щоб запити на доступ до даних сховища, які

направляються на адресу сервера сховища, могли б бути опрацьовані декількома реальними серверами на комп'ютері, який приймає запити на дані, повинен бути запущений спеціальний сервіс сателіта. Основна мета цього сервісу – виконання наступних основних задач:

- отримання запитів від клієнта;
- вибір реального сервера сховища, який буде опрацьовувати даний запит від клієнта;
- перенаправлення запиту від клієнта до реального сервера сховища;
- отримання відповіді від реального сервера сховища на запит клієнта;
- перенаправлення відповіді реального сервера на запит до клієнта.

Таким чином, усі запити, які попадають у сховище, проходять спочатку через сервіс сателіта. Відповідно, на опрацювання одного запиту сервісу сателіта потрібен процесорний час. Тому потужність сервера сховища в цілому обмежена не тільки потужністю комп'ютерів сховища, але й потужністю комп'ютера, на якому запущений сервіс сателіта. Звідси випливає, наскільки важливо реалізувати сервіс сателіта максимально ефективно.

Можна представити таку модель хмаркового сховища у вигляді графової моделі. Нехай хмаркове сховище даних включає N серверів даних $D = \{d_1,...,d_N\}$, які об'єднані між собою каналами зв'язку. Специфіка організації надійних сховищ даних передбачає, що усі сервери сховища мають безпосередній зв'язок з будь-яким іншим. Таким чином така мережа серверів сховища представляє собою повнозв'язний граф $G_D = (D, L_D)$, в якому L_D представляють дуги графа, що символізують канали зв'язку між серверами даних.

Окрім серверів даних до хмаркового сховища належать також K сателітів $St = \{St_1,...,St_K\}$, які також об'єднані між собою каналами зв'язку. І, як у випадку з серверами, усі сателіти мають безпосередні зв'язки з кожним сателітом (L_{St}). Окрім того, усі сателіти пов'язані окремими зв'язками (L_Z) з

серверами даних, утворюючи повну модель мережі сховища даних, яка також представляє собою повнозв'язний граф $G = (P, L)$, в якому $P = St \bigcup D$, $L = L_D \bigcup L_{St} \bigcup L_Z$.

Для користувача хмаркове сховище даних представлено, як мережа сателітів $G_{St} = G \bigcap G_D$, через які він може отримувати доступ до сховища даних.

Дана модель представлення хмаркового сховища даних не обмежується його типом – приватна, публічна чи гібридна. Вона дозволяє абстрагуватися від типу сховища, а сконцентруватися на організації доступу до нього.

Проблема полягає у тому, щоб забезпечити користувачу якісний сервіс. З технічної точки зору – при отриманні запиту від користувача переадресувати його на той сателіт (точку підключення до хмаркового сховища), який забезпечить для клієнта якісне надання послуги доступу до даних.

Рішенням даної проблеми може бути два варіанти.

При першому варіанті сервіс сателіта не повинен визначати до яких даних хоче отримати доступ клієнт, який сателіт буде оптимальнішим у даній ситуації, так як усі сателіти мають доступ до необхідних серверів даних. Тому сервіс сателіту повинен прийняти з'єднання і, не очікуючи надходження запиту, вибрати довільний сателіт та довільний сервер даних для його обслуговування.

Хоча даний підхід доволі простий, його складно обслуговувати. Адміністратор повинен постійно дбати про те, щоб всі сервери даних мали однаковий вміст – проводити реплікацію на N серверів даних. Це достатньо складно з технічної точки зору. Крім того, це приводить до додаткових витрат довготермінової пам'яті, так як копія кожного ресурсу повинна бути присутня на кожному сервері даних. Ще одним недоліком такого підходу є те, що він не дозволяє його масштабувати на регіонально-розподілені сховища даних, враховувати віддаленість клієнтів від сателітів та серверів даних.

Другий підхід повинен виправити наведені недоліки першого підходу, тобто врахувати регіональне розподілення серверів даних, що викликає додаткові передачі даних між серверами. Взаємне розміщення клієнтів і сателітів, викликає вибору оптимальної точки підключення до хмаркового сховища.

Використовуючи дану модель можна оцінити ефективний час обміну між користувачами та сховищем даних.

Так для отримання даних з хмаркового сховища, незалежно на якому сервері хмарки ці дані знаходяться у вигляді файлу f потрібно затратити час:

$$T_{down}(f) = \frac{F}{V_{down\_k}(f)}, \tag{2.6}$$

де k – користувач даних,

T_{down} – час завантаження,

F – розмір файлу даних f.

Швидкість отримання файлу f користувачем $V_{down\_k}(f)$ визначається:

$V_{down\_k}(f) = \min(V_{down\_k}, V_{down\_St_j}(f))$,

де V_{down_k} – швидкість отримання даних клієнтом,

$V_{down\_St_j}(f)$ – швидкість передачі файлу f з сховища до сателіту St_j.

А швидкість передачі файлу f зі сховища до сателіту St_j визначається за формулою, враховуючи, що файл може бути розміщений на декількох серверах і зчитуватися паралельно з них:

$$V_{down\_St_j}(f) = \min\{V_{upl\_St_j}, V_{down\_St_j}, \sum_{i=1}^{K} V_{down\_d_i}(St_j, f)\}, \tag{2.7}$$

де S_{t_j} – сателіт j,

d_j – сховище i,

$V_{upl\_St_i}$ – швидкість надання даних сателітом,

$V_{down\_St_i}$ – швидкість отримання даних сателітом зі сховища.

Отже сумарна формула часу отримання файлу зі сховища визначатиметься:

$$T_{down}(f) = \frac{F}{\min\{V_{down\_k}, V_{upl\_St_j}, V_{down\_St_j}, \sum_{i=1}^{K} V_{down\_d_i}(St_j, f)\}}. \quad (2.8)$$

Оцінка часу завантаження даних від клієнта в хмаркове сховище може бути проведена аналогічно, враховуючи той факт, що завантажувати потрібно не на декілька серверів сховища, а лише на одне:

$$T_{upl}(f) = \frac{F}{V_{upl\_k}(f)}. \quad (2.9)$$

Швидкість завантаження даних від k-го клієнта:

$$V_{upl\_k}(f) = \min\{V_{upl\_k}, V_{upl\_St_j}(f)\}, \quad (2.10)$$

$$V_{upl\_St_j}(f) = \min\{V_{down\_St_j}, V_{upl\_St_j}, \max(V_{down\_d_i}(St_i, f))\}.$$

Отже, час завантаження в сховище визначатиметься:

$$T_{upl}(f) = \frac{F}{\min\{V_{upl\_k}, V_{down\_St_j}, V_{upl\_St_j}, \max\{V_{down\_d_i}(St_j, f)\}\}}. \quad (2.11)$$

Отже, запропонована модель представлення організації хмаркового сховища даних дозволила оцінити час завантаження і зчитування файлів в сховищі даних.

2.3. Модифікація методу доступу через хмаркове сховище.

При моделюванні передачі даних можливі як варіанти детального моделювання, так і моделювання на макро рівні. Останній є більш узагальненим методом моделювання і дозволяє побудувати цілісну модель передачі «з кінця в кінець». Для досліджуваної теми це є досить важливо, так як в такій моделі враховуються усі нюанси передачі, які узагальнюються у відповідні параметри моделі.

Традиційна модель передачі файлів між двома абонентами відбувається за наступними принципами (рис. 2.8):

1. Файл поділяється на блоки.
2. Створюється черга блоків для передачі.
3. За допомогою мультиплексування проводиться передача окремих блоків через N каналів зв'язку.
4. За допомогою демультиплексування приймаються передані блоки файлу.
5. Прийняті блоки записуються у пріоритетну чергу (пріоритет визначається порядком блоків у файлі).
6. Об'єднані блоки записуються на диск приймаючого абонента.

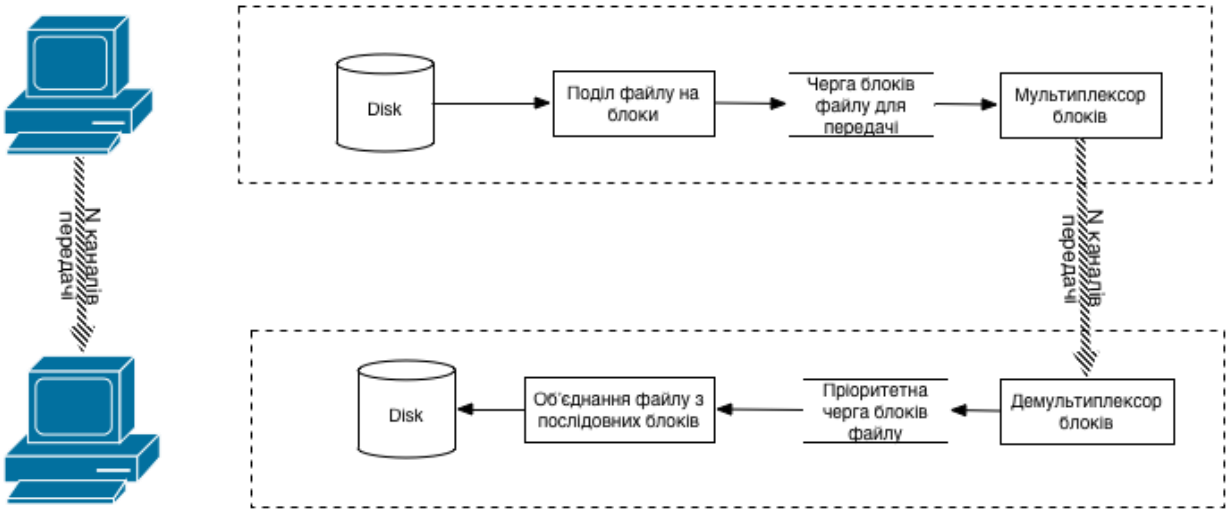


Рис. 2.8. Модель передачі файлів між двома абонентами, побудована самостійно.

Такий класичний підхід використовується як класична модель передачі та закладена у програмне забезпечення передачі файлів через мережу. Модель складалася історично і підтримується усім наявним програмним і апаратним забезпеченням мереж, тому при внесенні змін до такої моделі потрібно враховувати, що на кінцевих сторонах приймання-передачі повинна забезпечуватись прийнята ідеологія передачі.

Складність і розгалуженість сучасних мереж вимагають для передачі файлів використовувати проміжні елементи. Це можуть бути виділені сервери, чи інше обладнання.

У такому випадку, в загальному модель передачі з кінця в кінець залишається такою самою (див. рис. 2.8), але при її деталізації появляться

проміжні елементи приймання-передачі. Чим більше таких проміжних елементів буде на шляху передачі, тим складніша буде модель, яка фактично складатиметься з послідовностей класичних моделей приймання передачі.

Структурна модель змінюється при використанні проміжного елемента при передачі. У нашому випадку це хмарне сховище даних, яке на шляху передачі може бути й не одне. У моделі передачі появляється приймально-передаючий елемент (рис. 2.9). Дана модель вказує на те, що час передачі буде збільшуватися за рахунок того, що проміжний елемент повинен виконати дії з прийняття повного файлу, а потім подальшій його передачі до адресата. Неефективність такого підходу стає більш ніж очевидно.

Такий підхід зумовлений особливостями передачі інформації з використанням протоколів класу TCP.

Дану проблему можна обійти, коли в проміжному елементі не очікувати прийняття усіх блоків файлу, а передавати далі в процесі надходження блоків і кінцеве об'єднання файлів виконувати на кінцевій стороні передачі (рис. 2.10).

У даному випадку потрібно змінити логіку роботи проміжного елемента.

При прийнятті блоків файлу у проміжному елементі проводиться їх запис у пріоритетну чергу блоків. Пріоритет блоку визначається його порядковим номером у файлі. Таким чином у пріоритетній черзі вибудовується вся структура файлу. При повному заповнені черги блоків в ній буде розміщений увесь файл, причому з правильною послідовністю блоків.

Окрім пріоритетної черги на проміжному елементі міститься лічильник переданих блоків, який відслідковує номер останнього переданого блоку. У початковий момент він рівний 0. Передача блоку відбувається в той момент, коли номер блоку файлу у черзі на одиницю більший за лічильник переданих блоків. Тобто, фактично, передача блоків файлу почнеться у момент, коли в голові черги появиться блок з номером 1. Після передачі чергового блоку –

лічильник збільшує своє значення на одиницю. Такий процес триває дотого моменту, поки усі блоки файлу не будуть передані далі.

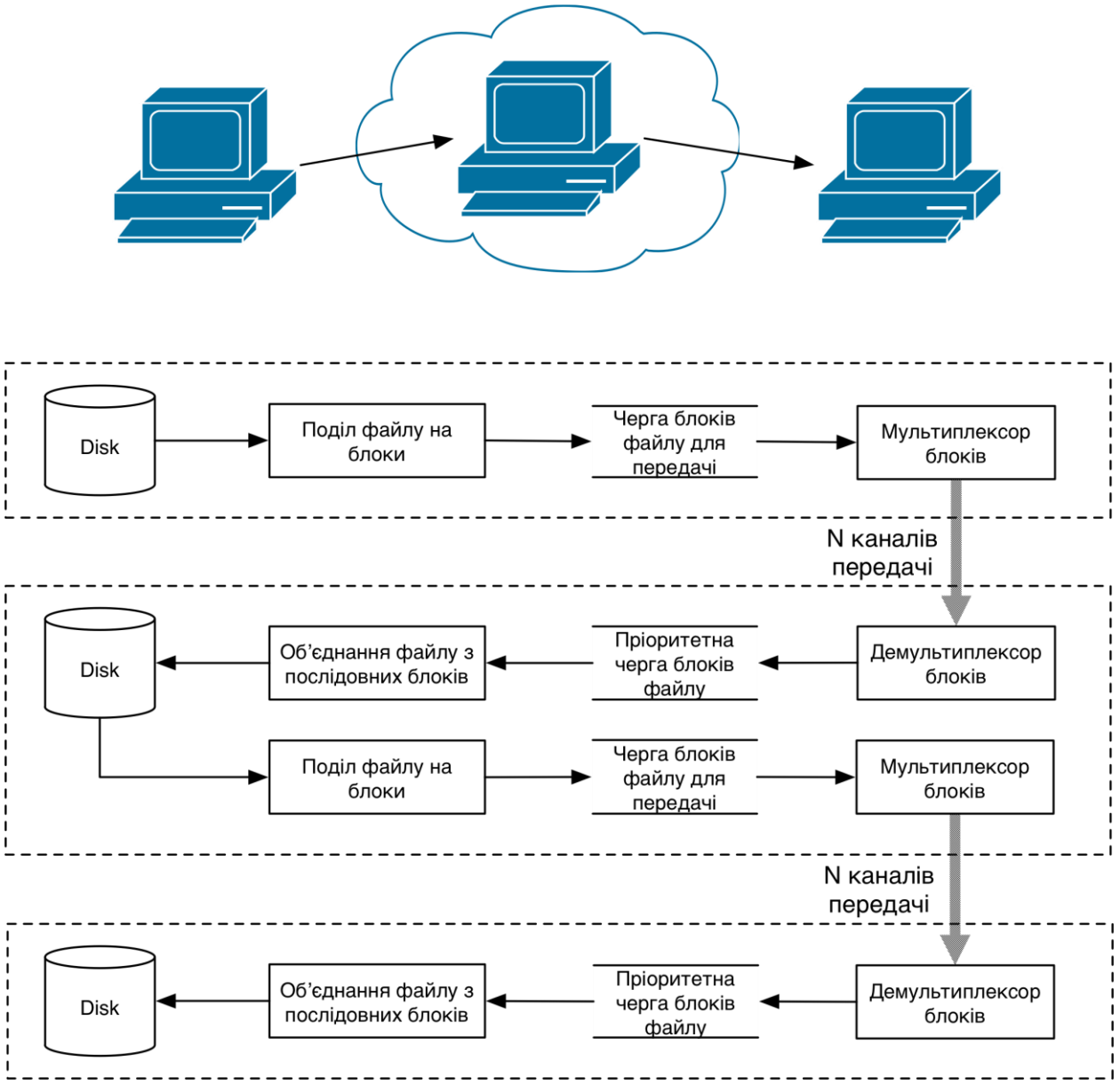


Рис. 2.9. Модель передачі файлів через проміжний елемент, побудована самостійно.

Для кінцевих користувачів запропонована модель нічим не буде відрізнятися від старого підходу. Очевидний виграш у повній швидкості передачі забезпечується виключенням з моделі передачі процесів збору повного файлу на проміжних етапах.

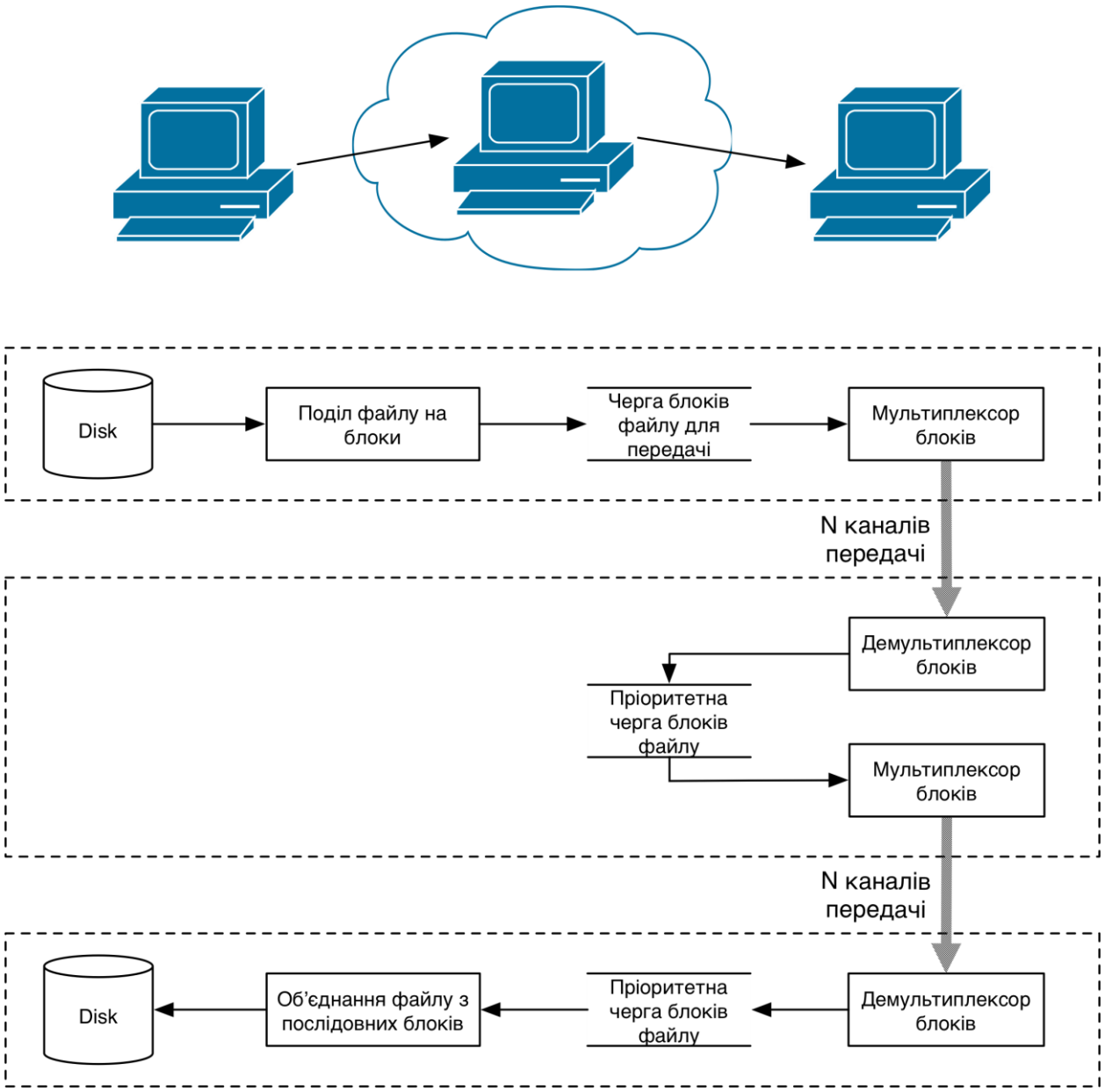


Рис. 2.10. Модифікована модель передачі файлів між двома абонентами, побудована самостійно.

Затримка, яка буде виникати при передачі за запропонованою моделлю, викликана очікуванням чергового блоку файлу для передачі на проміжному елементі. Нерівномірність надходження блоків файлу на проміжний елемент викликана їх передачі різними каналами, кожен з яких може мати свої характеристики передачі. Варто зауважити, що імовірність надходження чергового блоку файлу також описується законами розподілу, що використовуються при моделюванні класичної моделі передачі.

2.5. Моделювання завантаженості хмаркових сховищ даних.

У процесі обробки та зберігання інформації неминуче виникає необхідність в обміні даними між учасниками цього процесу. З кінця 70-тих років почався бурхливий розвиток комп'ютерних мереж і відповідного мережевого обладнання. Локальні та глобальні мережі продовжують розвиватися, виникають нові протоколи передачі даних, розширюються апаратні можливості мережевого обладнання, зростає кількість підключених абонентів і сумарний об'єм трафіку.

Існуючі сучасні хмаркові сховища даних за своєю внутрішньою структурою не використовують максимально свої потенційні функціональні можливості. Одна з причин – складність у поведінці мережного трафіку, як в середині хмарки, так і зовнішній, що впливає на якість обслуговування.

З точки зору сучасних тенденцій розвитку телекомунікацій, і не тільки хмаркових сховищ даних, актуальною задачею є побудова конвергентної мультисервісної мережі. Така мережа повинна забезпечувати необмежений набір послуг, надавати гнучкі можливості для управління і створення нових видів сервісів. Останнє вимагає реалізації універсальної транспортної мережі з розподіленою комутацією, де взаємодія між пристроями і додатками здійснюється за допомогою створення віртуальних з'єднань, на управління якими помітно впливають особливості стохастичної динаміки процесів пакетної комутації.

З іншого боку, інтенсивний розвиток галузі тягне за собою ряд проблем. Одна з них полягає в тому, що при зростальній кількості споживачів інформаційних послуг зростають вимоги до мережевого та серверного обладнання, яке необхідне для підтримування належного рівня якості обслуговування.

Розвиток мережевого обладнання і транспортних протоколів повинні базуватися на адекватних математичних моделях параметрів трафіку та інструменти моделювання мережевих процесів [97, 101]. Характер мережевого трафіку визначається рядом факторів – від поведінки

користувачів або прикладного програмного забезпечення, до протоколів передачі та використовуваного обладнання. Очевидно, що макропараметри мережевого трафіку на відносно великих часових інтервалах) визначаються людиною. Проте, характер трафіку на інтервалах порядку мікросекунд визначається, в основному, транспортними протоколами, мережевим обладнанням і серверним програмним забезпеченням. Таким чином, дослідження основних характеристик хмаркового сервера, таких як виділення оперативної пам'яті, завантаження центрального процесора, стан процесів операційної системи на фоні інтенсивного мережевого трафіку, є актуальною задачею.

Модель повина ідентифікувати предикат завантаженості хмаркового сховища, що реально зробити лише використовуючи реальне сховище даних. Для цього необхідно: дослідити трафіки даних у хмаркових сховищах, проаналізувати об'єднаний потік даних, встановити залежність рівня фрактальності сумарного потоку. Відповідні параметри сховища даних S_{cloud} для моделювання визначаються з практичних значень – вхідний/вихідний трафік, кількість запущених процесів, завантаженість і простій процесорів, середнє навантаження на процесор та об'єм кеш-пам'яті.

$$L(CH(S_{cloud_i}), CH(S_{cloud_k})) \to Z\ . \tag{2.12}$$

Предикат завантаженості хмаркового сховища поданий як відношення завантаженості ХСД у моменти часу t_1 та t_2. Завантаженість ХСД – функція значень параметрів ХСД:

$$CH(S_{cloud}) = \frac{IT \cdot OT \cdot LCPU \cdot MC}{V \cdot PN \cdot (FCPU + LCPU) \cdot ALCPU} \tag{2.13}$$

де, IT – вхідний трафік, OT – вихідний трафік, V – канал, PN – кількість запущених процесів, $LCPU$ – завантаженість, $FCPU$ – простій процесорів, $ALCPU$ – середнє навантаження на процесор та MC – об'єм кеш-пам'яті.

Однією з найбільш актуальних проблем дослідження імовірнісно-

часових характеристик хмаркових сховищ даних є адекватне врахування особливостей мережного трафіку. Доцільним є розгляд різних моделей мережевого трафіку і аналіз найбільш перспективної моделі для хмаркових сховищ даних, яка враховує самоподібні властивості трафіку як часового ряду.

На прикладі реального хмаркового сховища даних побудована динамічна модель характеристик вхідного та вихідного трафіку, а також розподіл апаратних потужностей хмаркового сервера. Для всіх процесів було встановлено самоподібність, що підтверджує можливість застосування фрактальних моделей для роботи з хмарковими сховищами даних, зокрема, для прогнозування поведінки серверів хмаркових сховищ [99].

Класичні підходи теорії віддаленого трафіку базуються на припущенні, що вхідні потоки є стаціонарними потоками Пуасонівського типу, тобто являють собою суперпозицію великої кількості незалежних стаціонарних ординарних потоків без післядії рівномірно малої інтенсивності. Для телефонних мереж з канальною комутацією таке припущення є справедливим. Проте дослідження показують, що трафік телекомунікаційних мереж сучасних хмаркових сховищ даних з комутацією пакетів має особливу структуру, яка не дозволяє використовувати при проектуванні звичні методи, які базуються на марківських моделях і формулах Ерланга. Мова йде про прояв ефекту самоподібності віддаленого трафіку, тобто в реалізації завжди присутня деяка кількість досить сильних викидів на фоні відносно низького середнього рівня. Це явище значно погіршує характеристики (збільшення втрат, затримки, джитер) при проходженні самоподібного трафіку через мережу.

До недавнього часу теоретичною базою для проектування систем розподілу інформації забезпечувала теорія телетрафіку, яка є однією з гілок теорії масового обслуговування.

Дана теорія добре описує процеси, що відбуваються в таких системах розподілу інформації, як телефонні мережі, побудованих за принципом

комутації каналів. Найбільш поширеною моделлю потоку викликів (даних) в теорії телетрафіку є найпростіший потік (стаціонарний ординарний потік без післядії), який також називають стаціонарним пуасонівським потоком.

Сучасний стан бурхливого розвитку високих технологій привів до появи і повсюдного поширенню мереж з пакетною передачею даних, які поступово стали витісняти системи з комутацією каналів, але, як і раніше, вони проектувалися на основі загальних положень теорії телетрафіку.

Таким чином, утворилася «проблема самоподібності телетрафіку», якій за останні роки присвячено більше тисячі робіт, і яка до цих пір не втратила своєї актуальності.

Незважаючи на значну популярність цієї тематики і тривалий період її активного вивчення, доводиться констатувати, що до цих пір залишається безліч питань і невирішених завдань.

Основні з них:

- фактично відсутня теоретична база, яка прийшла б на зміну класичній теорії масового обслуговування при проектуванні сучасних систем розподілу інформації з самоподібним трафіком; нема єдиної загальновизнаної моделі самоподібного трафіку;
- не існує достовірної та визнаної методики розрахунку коефіцієнта пачечності для заданого потоку, що відповідає відношенню пікової інтенсивності процесу надходження заявок обслуговування до його середнього значення;
- параметрів та показників якості систем розподілу інформації за умови впливу ефекту самоподібності;
- відсутні алгоритми і механізми, які забезпечують якість обслуговування в умовах самоподібного трафіку.

Тому є доцільним визначення характеристик фрактальних процесів різних потоків даних у хмаркових сховищах даних для подальшого прийняття рішення про спосіб управління ними.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- дослідити трафіки даних у хмаркових сховищах;
- проаналізувати об'єднаний потік даних;
- зробити висновок про залежність рівня фрактальності сумарного потоку від самоподібних властивостей окремих потоків, які він в собі містить.

Стохастичні моделі трафіку, які широко використовувалися раніше [84], в основному представляли марківські процеси, тобто мали короткотермінові залежності. Такі моделі описувалися розподілом Пуасона з змінною довжиною повідомлення за експоненційним законом, і базувалися на теорії масового обслуговування. Ці моделі сформувалися в період ранніх мереж ARPANET. Результати моделювання трафіку на основі теорії масового обслуговування відповідали розподілу часу обробки викликів в телефонних мережах.

Проте, пізніше стали все частіше появлятися роботи, в яких вказувалося про зростання тенденції міжмережевого обміну даними і об'єму сумарного трафіку. Також на характер трафіку стали впливати нові протоколи передачі даних в мережі, особливо у внутрішньому середовищі хмарних сховищ даних. На основі подібних досліджень у 1986 році була розроблена концепція «ланцюжка пакетів» (packettrain). В рамках моделі вважається, що пакети в мережі передаються разом, в той же час як в пуасонівських моделях кожен пакет опрацьовувався окремо.

Не так давно також стала популярною модель, в рамках якої об'єм трафіку має довготермінову залежність. Також було встановлено, що трафік самоподібний і характеризується розподілом з важким хвостом [93].

В рамках класичної моделі трафіку вважається, що джерела даних працюють почергово [63]. Тобто періоди високої активності змінюються тривалими затримками. Таким чином, було встановлено, що час надходження повідомлення і довжина повідомлень підкоряється експоненційному розподілу, а процес надходження повідомлень від джерел

даних – пуасонівський процес. Усі процеси стаціонарні і незалежні.

Пуасонівська модель не враховує, що реальний мережевий трафік має періоди сильних сплесків активності. Для класичної моделі автокореляційна функція прямує до нуля для великих відліків, у той же час як наявність сплесків активності в реальному досліджуваному трафіку приводить до додатньої автокореляції.

Модель трафіку як ланцюжка повідомлень була сформульована і стала популярною в 80-ті роки [67]. В рамках моделі вважається, що пакети трафіку передаються разом і можуть опрацьовуватися як одне ціле. Мережеве обладнання в кожній точці мережі може приймати рішення про подальшу обробку ланцюжка за першим повідомленням. Подібний алгоритм запобігав би мережу від даремних операцій для аналізу кадрів. Проте, варто зауважити, що це модель джерел повідомлень. Модель може бути застосована тільки для повідомлень, які мають один пунктом призначення. Очевидно, що реалізації транспортних протоколів і мережевого обладнання для моделі ланцюжка повідомлень і класичної моделі будуть кардинально відрізнятися.

У багатьох сучасних роботах відмічається [83, 93], що об'єднання трафіку від декількох змінних джерел приводить до того, що трафік стає сильно автокорельованим з довготерміновою залежністю [103]. Це приводить до того, що усталеність кореляційних структур не зчезає навіть для великих значень лагу. Іншими словами, сукупність множини джерел даних, які проявляють синдром нескінченної дисперсії, в результаті дає самоподібний об'єднаний мережевий трафік, який наближається до фрактального броунівського руху. Крім того, дослідження різних джерел трафіку показують, що сильно змінна поведінка – це властивість, яка властива для архітектури клієнт/сервер.

Проблемами самоподібності мережного трафіку займалося багато вчених. Зокрема в роботі [1] проведено дослідження властивостей реального трафіку в мережах з пакетною комутацією. З використанням методу R/S

аналізу показано самоподібну природу мережного трафіку в інформаційних мережах. На основі такого підходу була розроблена модель генератора трафіку, який реалізовує мультифрактальну поведінку потоків даних в реальних інформаційних системах, що дозволяє імітувати трафік з заданими показниками самоподібності. У роботі [9] показано, що в мережах стандарту 802.16b самоподібні властивості трафіку проявляються як на канальному, так і на транспортному рівнях. Отримані значення основних показників ступеня фрактальності мережевого трафіку та запропоновано методи агрегування вихідної статистики [2].

Робота [83] присвячена експерименту зі зняття трафіку мережі одного з великих Інтренет-провайдерів, а також наводяться результати аналізу структурних особливостей данного трафіку. Авторами доведено, що самоподібні властивості проявляють себе як на канальному, так і на транспортному рівнях. У роботі [60] американські вчені вивчали процеси з довготривалою залежністю. Для генерації таких процесів автори пропонують використання фрактальної моделі інтегрованого змінного середнього.

Самоподібність – це властивість об'єкту, частини якого подібні до всього об'єкту в цілому. Багато об'єктів у природі мають такі властивості, наприклад узбережжя, хмари, кровоносна система людини чи тварин.

Неформально самоподібний (фрактальний) процес можна визначити як випадковий процес, статистичні характеристики якого проявляють властивість масштабування. Самоподібний процес суттєво не змінює виду при розгляді в різних масштабах за шкалою часу. Зокрема, на відміну від процесів, які не мають фрактальних властивостей, не відбувається швидкого «згладжування» процесу при усередненні за шкалою часу – процес зберігає схильність до сплесків.

Якщо вважати процес передачі даних через хмаркове сховище $\{X_k; k = 0,1,2,\ldots\}$ стаціонарним випадковим процесом і, враховуючи стаціонарність і припущення про існування і скінченність двох перших моментів, можна використати:

$m = E[X_t]$ – середнє значення, або математичне очікування;

$\sigma^2 = E[X_t - m^2]$ – дисперсія;

$R(k) = E[(X_{t+k} - m)(X_t - m)]_\infty$ – кореляційна функція,

$r(k) = R(k) \big/ R(0) = R(k) \big/ \sigma^2$ – коефіцієнт кореляції.

Під усередненням за часовою шкалою розуміють перехід до процесу $\{X^{(m)}\}$, такому, що

$$X_k^{(m)} = \frac{1}{m} \sum_{i=k_{m-m+1}}^{k_m} X_i \,. \tag{2.14}$$

При моделюванні мережевого трафіку значення X_k інтерпретуються як кількість пакетів (рідше – як сумарний об'єм даних в байтах), які поступили в канал, або мережу протягом k-го інтервалу часу. Вихідний процес при цьому вже є усередненим. У деяких випадках, коли є необхідність уникнути такого початкового усереднення, розглядається точечний процес, або потік подій, тобто послідовність моментів надходження одиничних пакетів в мережу.

Не існує єдиного причинного фактору, що викликає самоподібність. Різні кореляції, які існують в самоподібному мережевому трафіку, які впливають на різних часових масштабах, можуть виникати через різні причини, проявляючи себе в характеристиках на конкретних часових масштабах.

Причиною довгострокової залежності в мережевому трафіку можуть бути наступні фактори:

- поведінка користувача і прикладного програмного забезпечення;
- генерація, структура і пошук даних;
- об'єднання трафіку;
- засоби адміністрування мережі;
- механізми оптимізації, які базуються на зворотніх зв'язках;
- ускладнення структури мережі, збільшення кількості абонентів.

Процес X називається самоподібним з параметром $H = 1 - (\beta/2)$, якщо його коефіцієнт автокореляції

$$r(k) = \frac{1}{2}\left[(k+1)^{2-\beta} - 2k^{2-\beta} + (k-1)^{2-\beta}\right] = g(k), \quad k \in N, \tag{2.15}$$

де функція

$$g(k) = \frac{1}{2}\delta^2 k^{2-\beta}. \tag{2.16}$$

виражена через центральний різницевий оператор 2-го порядку $\delta^2(f(x))$, який діє на функцію $f(x) = x^{2-\beta}$ так, що $\delta(f(x)) = f(x+1/2) - f(x-1/2)$.

Самоподібність проявляється в тому, що для процесу, який задовольняє першу умову, виконується рівність $rm(k) = r(k)$, тобто в такому процесі не змінюється коефіцієнт автокореляції після усереднення за блоками будь-якої довжини m. Таким чином, для самоподібного процесу статистичні характеристики другого порядку нормованого агрегованого процесу $X(m)$ не відрізняється від характеристик вихідного процесу X при значному інтервалі змін m.

Параметр H є індикатором міри самоподібності процесу, а також свідчить про наявність у нього таких властивостей як персистентність / антиперсистентність і тривала пам’ять [66]. Параметр може приймає значення від 0 до 1. Для білого шуму (марківський процес) параметр Херста рівний 0,5, що означає повну відсутність довгострокової або короткострокової залежності і процес є повністю випадковим, відповідно, найпростіший (Пуасонівський) потік ще називають «потоком чистої випадковості першого роду».

У випадку $H \in [0.5,1]$ процесс є персистентним, має довгострокову залежність [65]: якщо протягом якогось часу в минулому спостерігалося збільшення параметрів процесу, то і в майбутньому в середньому буде відбуватися їх зростання. Іншими словами, ймовірність того, що на кроці

$k+1$ процес відхилиться від середнього в тому ж напрямку, що і на k кроці, настільки велика, наскільки параметр H близький до 1.

При $H \in [0,0.5]$ процесу властива антиперсистентність, він має короткострокову залежність [66]: високі значення процесу йдуть за низькими і навпаки. Тобто, ймовірність того, що на кроці $k+1$ процес відхилиться від середнього в протилежному напрямку (щодо відхилення на k кроці), настільки велика, наскільки параметр H близький до 0.

Довгострокова залежність є причиною різко виражених пульсацій процесу, проте дозволяє говорити про деяку передбачуваність в невеликих межах часу. З точки зору теорії черг, важливим наслідком корельованості потоку є неприйнятність оцінок параметрів черг, які базуються на передбаченні про однаковий і незалежний розподіл інтервалів у вхідному потоці.

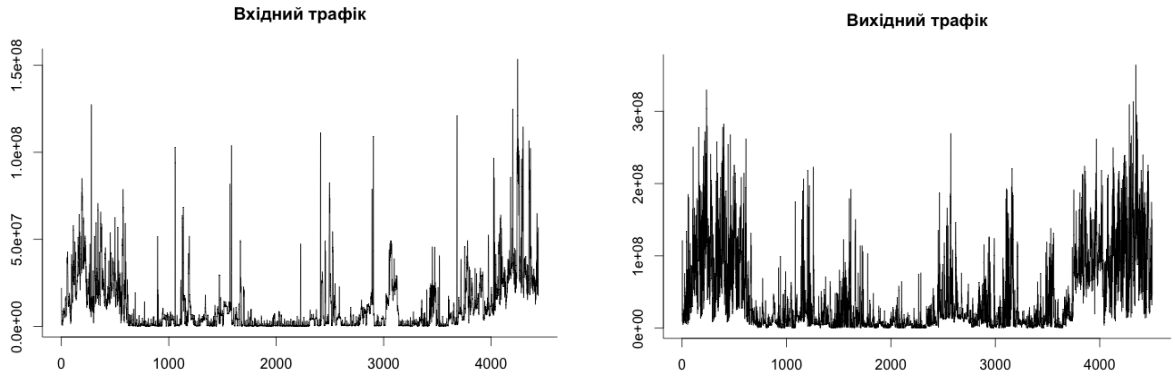
Для того щоб підтвердити існування властивості самоподібності для різних потоків даних мультисервісної мережі, необхідно провести вимірювання деяких характеристик різних видів мережного трафіку. Для цього необхідні статистичні дані про потоки і трафік даних, а також потрібно провести дослідження об'єднаного потоку та змінних характеристик сервера хмаркового сховища даних [21].

Для дослідження було використано сервер хмаркового сховища даних. Фізичний сервер поділено на декілька віртуальних зон з використанням операційної системи Solaris, кожна з яких використовується для виконання ряду задач. Більша частина трафіку передається за протоколами HTTP/HTTPS, FTP/FTPS та SFTP.

Для віддаленого моніторингу параметрів сховища даних в реальному часі використовується додаток Zabbix [109]. Zabbix – це додаток типу клієнт-сервер, який використовується для збору, зберігання та обробки інформації про стан мережі, мережеві навантаження, а також стану операційної системи сервера сховища даних в реальному часі. Для подальшої обробки були використані наступні параметри сховища даних:

- вхідний/вихідний трафік;
- кількість запущених процесів;
- завантаженість і простій процесорів;
- середнє навантаження на процесор;
- об'єм кеш-пам'яті.

Отримані дані консолідувалися на протязі тижня, тому можна вважати, що вони представляють реальну картину використання хмаркового сховища. Часова залежність об'єму трафіку приведена на рис. 2.11 для вхідного (а) і вихідного (б).



a) b)

Рис. 2.11. Часові залежності вхідного (а) і вихідного (б) трафіків.

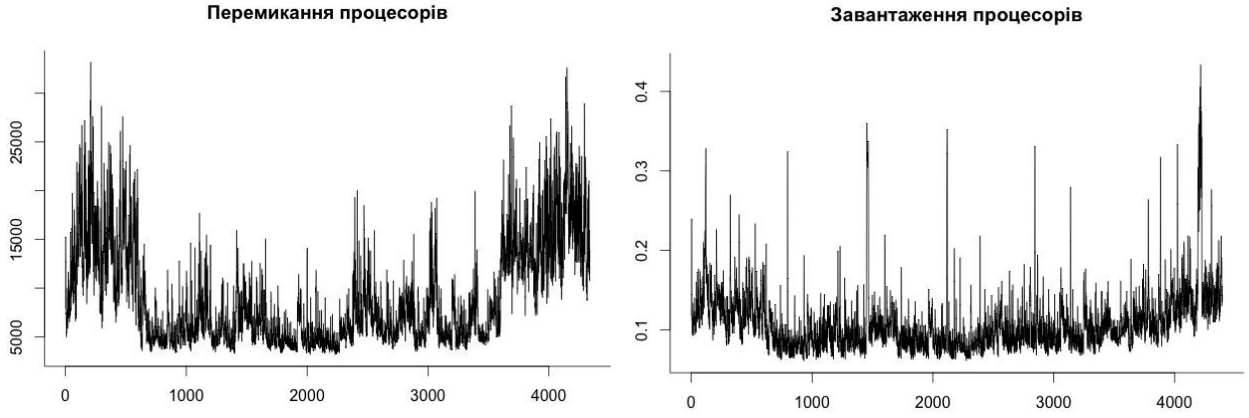
Так як дані аналізувалися на протязі тижня, графіки чітко демонструють певну добову періодичність.

Наступний рисунок (рис. 2.12) відображає міру завантаження процесорів та їх перемикання.

Завантаженість процесорів системою і користувачами представлена на рис. 2.13.

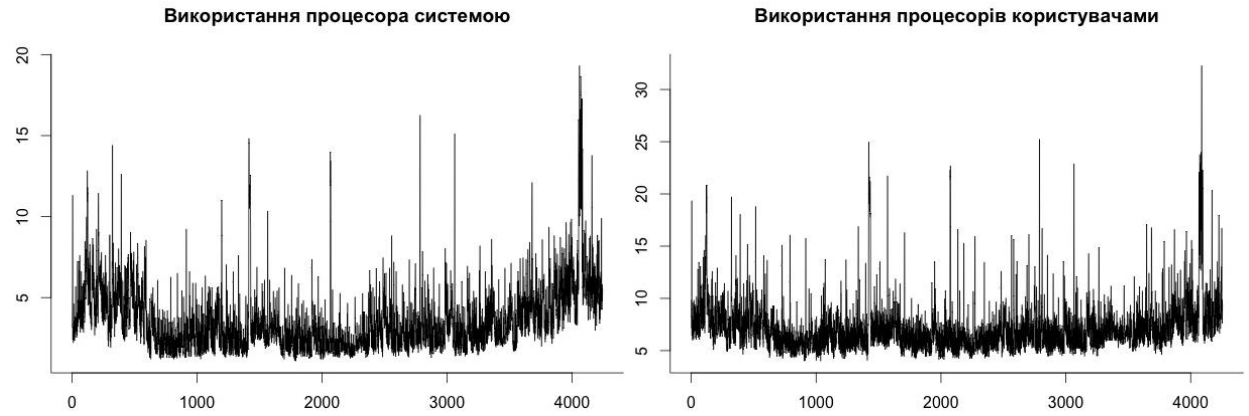
Аналогічно до даних графіку залежності вхідного і вихідного трафіку (див. рис. 2.11), інші залежності повторюють динаміку перших. Це пов'язано з тим, що основна функція хмаркового сховища даних – це ввід, зберігання та вивід даних користувачів. Тобто, основне навантаження на сервери сховищ даних представляє вхідний та вихідний трафіки. Тому при аналізі та

моделювання хмаркових сховищ можна обмежитися тільки моделями трафіку, або моделлю завантаженості процесора.



a) b)

Рис. 2.12. Часові залежності перемикання (а) і завантаженості (б) процесорів.



a) b)

Рис. 2.13. Часові залежності використання процесорів системою (а) і користувачами (б), розроблено самостійно.

Графічне представлення коефіцієнта автокореляції дозволяє візуально переконатися в тому, що досліджуваний трафік має довготермінову залежність. На рис. 2.14 приведено графік коефіцієнта кореляції для процесу, який відповідає вхідному трафіку в логарифмічному масштабі. Очевидно, що точки на рисунку в цілому групуються навколо прямої, кутовий коефіцієнт якої може бути визначений шляхом лінійної регресії.

Якщо процес є самоподібний, то у відповідності до (2.8) кутовий коефіцієнт $\beta = 2(H-1)$. При отриманому значенні $\beta = -0{,}2125$ параметр Херста виявився рівним $0{,}8937$.

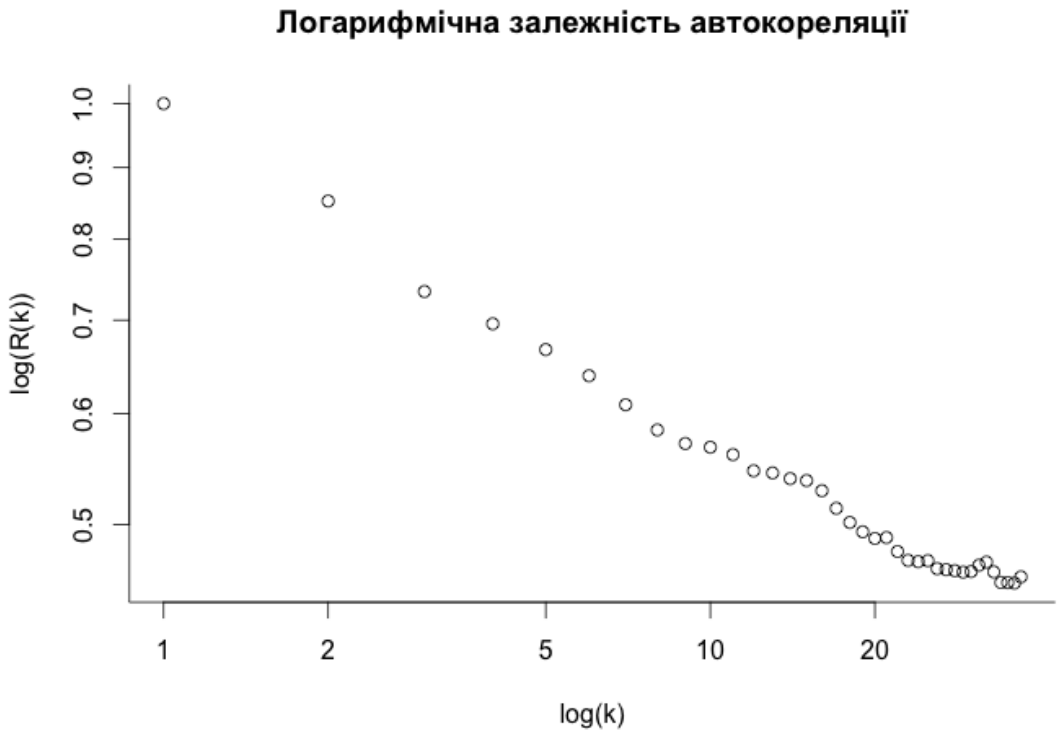


Рис. 2.14. Логарифмічна залежність коефіцієнту автокореляції вхідного трафіку, розроблено самостійно.

Для вихідного трафіку графік коефіцієнта кореляції – рис. 2.15. Так само як і для вхідного трафіку точки в цілому групуються навколо прямої, кутовий коефіцієнт якої може бути визначений шляхом лінійної регресії. Для самоподібного процесу – кутовий коефіцієнт $\beta = -0{,}1986$, параметр Херста виявився рівним $0{,}9007$.

Традиційно самоподібність у будь-якому стохастичному процесі виявляється через визначення параметру Херста H. Той факт, що $0{,}5 < H < 1$, тобто значення параметру Херста більше $0{,}5$, вважається достатньою основою для визнання процесу самоподібним. Варто зауважити, що значення H, яке наближається до одиниці, може означати, що процес є детермінованим, тобто не випадковим: для ряду строго детермінованих процесів структура строго повторяється на будь-якому масштабі, що приводить до одиничного значення параметра Херста.

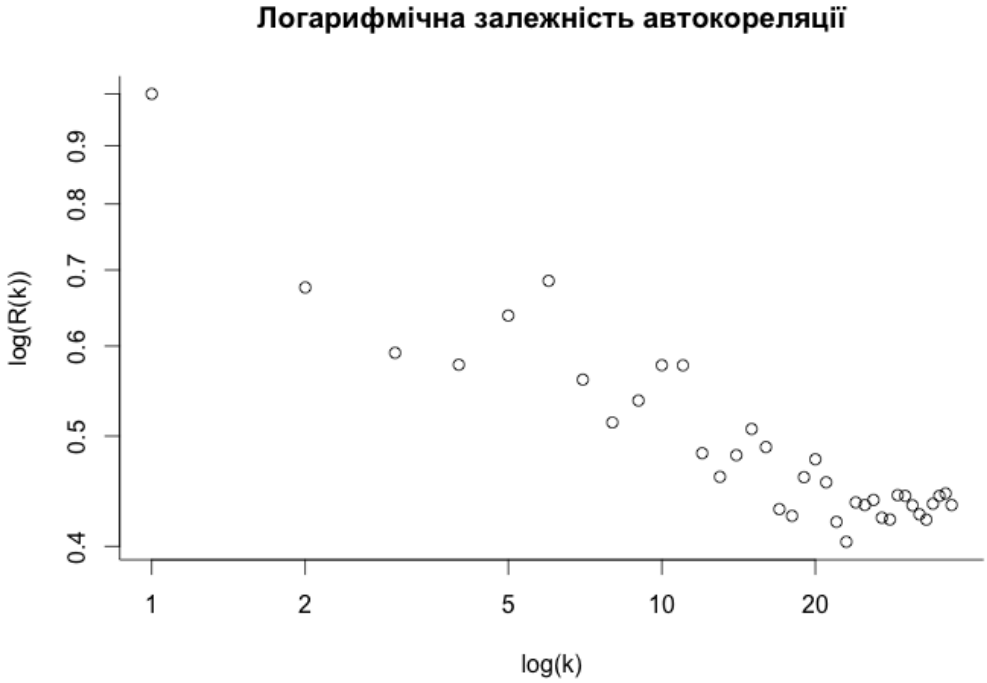


Рис. 2.15. Логарифмічна залежність коефіцієнту автокореляції вхідного трафіку, розроблено самостійно.

При розгляді часової залежності трафіку було відзначено присутність періодичної складової в ньому, що й приводить до такого великого значення параметра Херста. Звичайно, наближеність параметру Херста до одиниці дозволяє більш точно виконувати прогнозування. Для того, щоб дослідити трафік детальніше потрібно розглянути окремо часові залежності в межах одного дня.

Виключення періодичної добової складової з трафіку та окремий аналіз його в межах одного дня (рис. 2.16) показує на дещо менше значення параметру Херста, воно коливається в межах 0,70 – 0,87.

У результаті побудови моделі завантаженості хмаркового сховища даних було встановлено періодичність трафіку хмаркового сховища даних, що має добовий характер. Інтенсивність завантаження сховища в основному залежить від вхідного і вихідного трафіку. Достатнє високе значення параметру Херста вказує на потенціну можливість моделювання і прогнозування завантаженості хмаркового сховища даних в довгостроковий період.

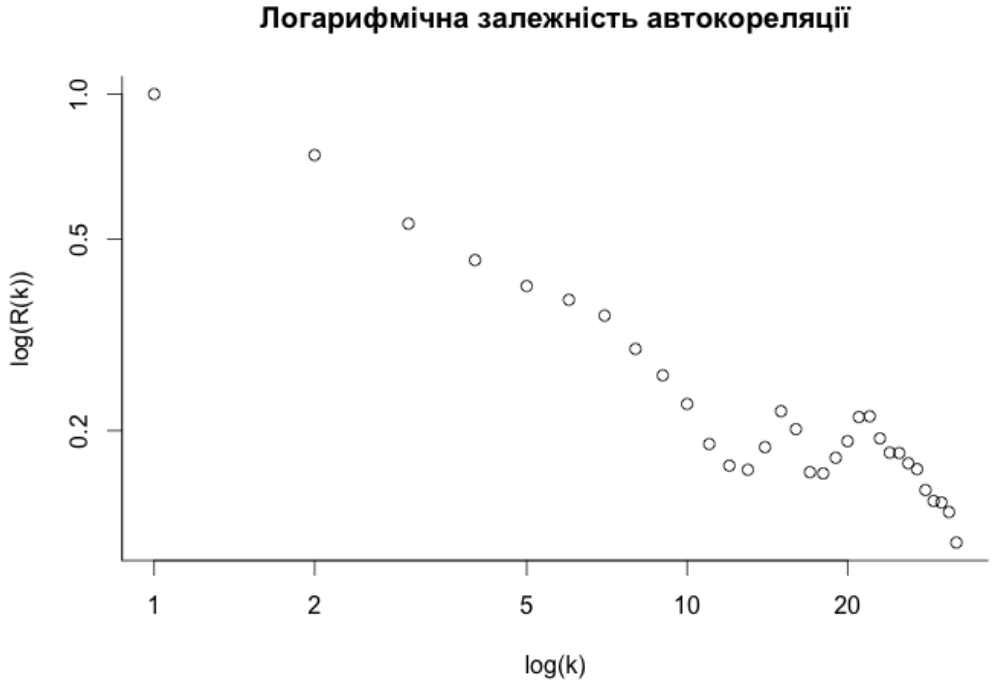


Рис. 2.16. Логарифмічна залежність коефіцієнту автокореляції для одноденного трафіку, розроблено самостійно.

2.6. Висновки до 2-го розділу.

У результаті роботи у даному розділі:

1. На основі існуючих підходів та їх недоліків, розроблено модель хмаркового сховища даних та методи передачі даних.

2. Уведено модель гібридних протоколів передачі даних через хмаркове сховище, обгрунтовано модель мережевого трафіку, проведено моделювання завантаженості сховища даних.

3. Удосконалено модель хмаркового сховища як системи алгебраїчних рівнянь.

4. Побудовано моделей протоколів передачі даних так повну модель передачі «з кінця в кінець».

5. Проведено моделлювання завантаженості хмаркового сховища, яке дозволило виявити процеси самоподібності з передачею даних через нього.

6. Побудовано модель хмаркового сховища зберігання даних. Запропоновано методи покращення ефективності хмаркових сховищ даних:

- метод мультипротокольної передачі потокових даних, який на відміну від методу вибору протоколу на всій ділянці мережі характеризується вищою продуктивністю, адаптуючись під кожну ділянку мережі окремо;
- метод мультиплексування різних джерел даних для хмаркового сховища даних, який на відміну від методу балансування навантаження в режимі реального часу визначає завантаженість каналів передачі, що дає змогу задіяти вільні канали передачі даних;
- метод вибору шлюзу за складністю запиту, який базується на нечітких даних про швидкість обміну з клієнтом і дозволяє вибрати оптимальний за швидкістю маршрут і шлюз передачі даних на поточний момент.

Основні результати розділу опубліковані у працях [13, 14, 16, 21, 95, 99].

РОЗДІЛ 3.
РОЗРОБЛЕННЯ МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОЗПОДІЛЕНИХ ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ ХМАРНИХ СХОВИЩ ДАНИХ

У розділі – розроблено методи підвищення ефективності обчислень у хмаркових сховищ даних. Основна увага зосереджена на розробці протоколу сеансового рівня, методу мультипротокольного передавання потокових даних, методу мультиплексування різних джерел для одночасного передавання та методу вибору шлюху за складністю виконання запиту.

3.1. Розроблення протоколу сеансового рівня для високошвидкісних регіонально-розподілених мереж.

Швидке збільшення пропускної здатності комп'ютерних мереж дозволило розробляти численні додатки, які передбачають інтенсивну обробку даних. Ці нові додатки можуть виконувати задачі, починаючи від масової передачі даних (SDSS [104] та e-VLBI [87]), до інтерактивних систем високої пропускної здатності (GeoWall [70]).

Однак, різні програми ставлять різні вимоги до послуг передачі даних. Наприклад, додаток GeoWall може віддати перевагу плавній зміні швидкості передачі даних, тоді як для додатку SDSS бажано, щоб дані передавалися з максимально можливою швидкістю в приватних мережах.

Проте, нинішня система Інтернет призначена для забезпечення підтримки цілої множини різного типу додатків. Ця філософія дизайну Інтернету має великий вплив на розвиток транспортних протоколів. Більшість трафіку в Інтернеті формується за рахунок потоків TCP, але існують додатки для яких протокол TCP не забезпечує достатнього рівня ефективності. У контексті високопродуктивних обчислень, TCP добре відомий своєю низькою ефективністю та справедливим поділом ресурсів в мережах з високою затримкою пропускної здатності [44].

За останні декілька років дослідники комп'ютерних мереж запропонували багато нових алгоритмів керування перевантаженням для загальної та специфічної

передачі даних [46, 73, 75]. Тим не менше, більшість з цих запропонованих алгоритмів завершуються моделюванням та обмежуються умовами лабораторних експериментів; мало з них пройшли практичне тестування і випробування в реальних високопродуктивних мережах. З іншого боку, модифікації мережевого стеку ядра протоколу (наприклад, нові варіанти TCP) зазвичай вимагають кількох років для стандартизації, впровадження та широкого розгортання. Справді, з часів появи протоколу TCP, близько трьох десятиліть тому, тільки його чотири версії були широко розгорнуті, а саме Tahoe, Reno, NewReno, і SACK [44, 73, 108].

Хоча на сьогоднішній день все більше мереж отримують швидкість передачі даних 1 Гбіт/с і вище, але як і раніше актуальною проблемою для грід-додатків залишається використання широкої смуги пропускної здатності, через обмеження існуючих транспортних протоколів мережі [108]. Обмеження впроваджених мережевих транспортних протоколів є однією з головних причин, через яку так важко масштабувати додатки з інтенсивним використанням мережевих з'єднань від місцевих кластерів до глобальних мереж [36, 41, 49, 108].

Протокол управління передачею (TCP) успішно використовується протягом десятиліть, як основний протокол транспортного рівня стеку мережних протоколів. Проте останнім часом було показано, що TCP має деякі втрати продуктивності при його використанні для високошвидкісних мереж Wide Area, особливо для географічно віддалених мереж. Алгоритм управління перевантаженням AIMD, який використовується протоколом TCP, є досить «бідним» у розкритті доступної смуги пропускної здатності і у випадку великої втрати пакетів у високопродуктивних мережах з досить великими часами затримки [44].

Дослідники комп'ютерних мереж працюють над новими транспортними протоколами і алгоритмами контролю насичення для підтримки високошвидкісних мереж наступного покоління. Багато робіт, у тому числі варіантів TCP (FAST [43], BiC [46], Scalable [73], і HighSpeed [115]) і XCP [71] показали більш високу продуктивність при їх моделюванні. Однак практичне використання в реальних додатках цих протоколів все ще дуже обмежена через

труднощі їх реалізації, встановлення та обмеження технічного рівня. Користувачі мережі, яким потрібно передавати великі масиви даних зазвичай звертаються до рішень рівня додатків, серед яких дуже популярні протоколи на основі UDP, наприклад SABUL [57], UDT [108], Tsunami [77], RBUDP [86], FOBS [38] і GTP [114].

Протоколи на основі UDP забезпечують набагато кращу переносимість і прості в при їх встановленні. Однак, незважаючи на простоту впровадження протоколів на рівні користувача, досить важко налаштувати їх в ядрі, щоб зробити їх максимально ефективними. Оскільки реалізації рівня користувач не можуть змінити код ядра, можуть бути додаткові перемикання контексту і копіюванням ділянок пам'яті між рівнем користувача та рівнем ядра. На високих швидкостях передачі даних, ці операції дуже чутливі до завантаження процесора і продуктивності протоколу.

Враховуючи перераховані утруднення та недоліки існуючого стану передачі даних великого обсягу через високопродуктивні регіонально-розподілені мережі видається доцільним з практичної точки зору розробка протоколу сеансового рівня для таких мереж.

Новий протокол сеансового рівня повинен бути сумісним з стандартними протоколами стеку протоколів OSI – TCP та UDP. Крім того, він повинен бути адаптований для сучасних мереж, тобто ефективно використовувати пропускну здатність мережі, не залежно від її характеристик. При використанні такого протоколу не повинні збільшуватися затримки при переході передачі між різними типами мереж. Тим самим протокол повинен забезпечити ефективне використання ресурсів крайніх вузлів мережі. У питанні захищеності --- він повинен підтримувати шифрування даних, тобто він повинен дозволяти підключення протоколів шифрування [20].

Доцільно провести оптимізацію ефективності реалізації протоколу на базі UDP і показати, що за цими ідеями можна реалізувати ефективні та практичні додатки на базі протоколів UDP. Наприклад, використання середовища протоколу UDT (базований на UDP) [108], може легко підтримувати різні алгоритми

управління перевантаженням, наприклад, високошвидкісні TCP [36, 46, 73, 115] або вибуховий RBUDP [115].

Перевагами даного середовища протоколу є:

- нові протоколи на базі UDP можуть бути швидко прототиповані і випробувані;
- різноманітні алгоритми управління перевантаженням можна легко порівняти експериментально;
- конкретні прикладні протоколи, що використовують UDP, можуть бути розроблені відносно легко.

Також, з використанням даного середовища можуть розроблятися конкретні протоколи для розподілених даних або потокових медіа-додатків.

Основним недоліком даного середовища протоколу є додаткове навантаження самого середовища, які накладають додаткові витрати протоколу прикладного рівня у порівнянні з рівнем протоколу ядра.

Для досягнення високої продуктивності, керуються двома принципами:

1. Не повинно бути піків завантаження процесора, особливо на стороні отримання даних. Сплески завантаження CPU можуть призвести до невчасної обрабки прийнятих пакетів, що у свою чергу приведе до втрати даних передачі. Це може привести до серйозних проблем, які спостерігаються в TCP з AIMD, у якому при втраті пакету, зазвичай, витрачається багато часу для відновлення передачі даних при з'єднаннях на великих відстанях.

2. Загальне завантаження процесора повинно бути якомога меншим. Проста реалізація додатку може запобігти надмірному використанню центрального процесора. Крім того, інші операції з обробки даних, які також потребують процесорного часу, часто виконуються паралельно з передачею даних.

Для перевірки завантаженості центрального процесора (CPU), системних затримок та впливу розмірів пакетів та розмірів буферів на швидкість передачі даних для протоколу UDP було проведено декілька практичних експериментів.

Експерименти проводилися на п'яти різних системах (табл. 3.1). Усі вони були з’єднані з іншим віддаленим вузлом з аналогічною конфігурацією, через мережу з пропускною здатністю не менше ніж швидкість NIC. MTU на стендах становила 1500 байт.

Таблиця 3.1. Середовище експерименту.

Ім’я	CPU	Memory	NIC	OS
onno	Dual Itanium2 1.5GHz	8 GB	10 GbE	Linux 2.6.0
sara77	Dual Xeon 2.4GHz	2 GB	1 GbE	Linux 2.4.18
ncdm171	Dual PowerPC G4 1GHz	2 GB	1 GbE	Mac OS X
win91	Dual Xeon 2.4GHz	2 GB	1 GbE	Windows 7
ncdm87	Dual Opteron 2.4GHz	4 GB	1 GbE	Linux 2.6.8

При використанні розміру пакету UDP 1500 байт і розміру буфера UDP 1 МБ були отримані результати наведені у табл. 3.2. Використання процесора вимірювалося в одиницях МГц/Мбіт, що є відношенням частоти використання процесора до швидкості передачі даних. Експеримент проводився в локальних мережах. (Тактова частота процесора визначалась, як сума тактових частот всіх процесорів, які використовувалися додатком при тестуванні. У деяких системах кількість процесорів була менше 0,5, в яких використовувалася технологія Hyper-Threading.)

Таблиця 3.2. Використання центрального процесора і час затримки.

Ім’я	UDP			TCP		
	CPU Util. (MHz/Mbps)		Delay (ms)	CPU Util. (MHz/Mbps)		Delay (ms)
	Відправка	Отримання		Відправка	Отримання	
onno	0,22	0,35	0,062	0,23	0,50	0,068
sara77	0,40	0,45	0,070	0,51	0,51	0,086
ncdm171	1,22	1,45	0,202	2,22	2,73	0,245
win91	1,03	1,09	0,203	1,14	1,28	0,302
ncdm87	0,26	0,40	0,065	0,25	0,56	0,087

Для усунення впливу помилок RTT на результат, було використано локальні мережеві з'єднання з дуже малим значеннях RTT, тому що, в іншому випадку, великі значення RTT можуть порушити точність моделювання.

Результати експериментальних досліджень є усередненням результатів 100 випробувань, що дає можливість побачити більш реалістичні результати.

Результати (табл. 3.2) демонструють, що протокол UDP створює менше завантаження центрального процесора і дає менші часові затримки, ніж TCP, в основному за рахунок того, що цей протокол має менші накладні витрати на обробку даних, які передаються протоколом. Це означає, що ефективна реалізація протоколу на основі UDP може мати таку ж продуктивність, як і вбудована в ядро системи реалізація TCP.

Для визначення факторів впливу на продуктивність протоколу було проведено ряд додаткових експериментів зі зміною двох параметрів:

- розмір буфера UDP;
- розмір пакету протоколу.

Результати порівняння між двома локальними машинами зображено на рисунках (рис. 3.1, 3.2).

Рисунок 3.1 демонструє, що розмір буфера на стороні отримувача є вирішальним фактором пропускної здатності. Розмір буфера повинен бути достатньо великим для досягнення оптимальної пропускної здатності.

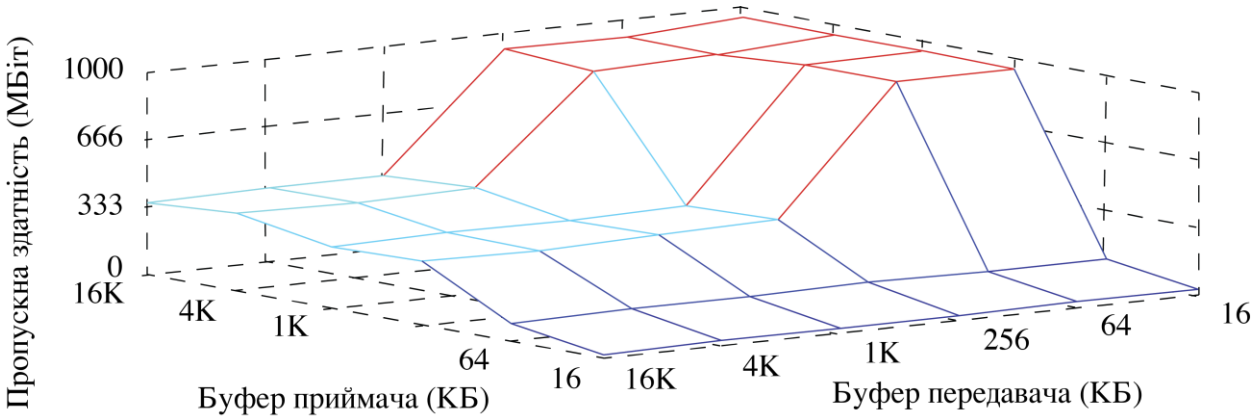


Рис. 3.1. Залежність продуктивності UDP від розміру буфера, побудована самостійно.

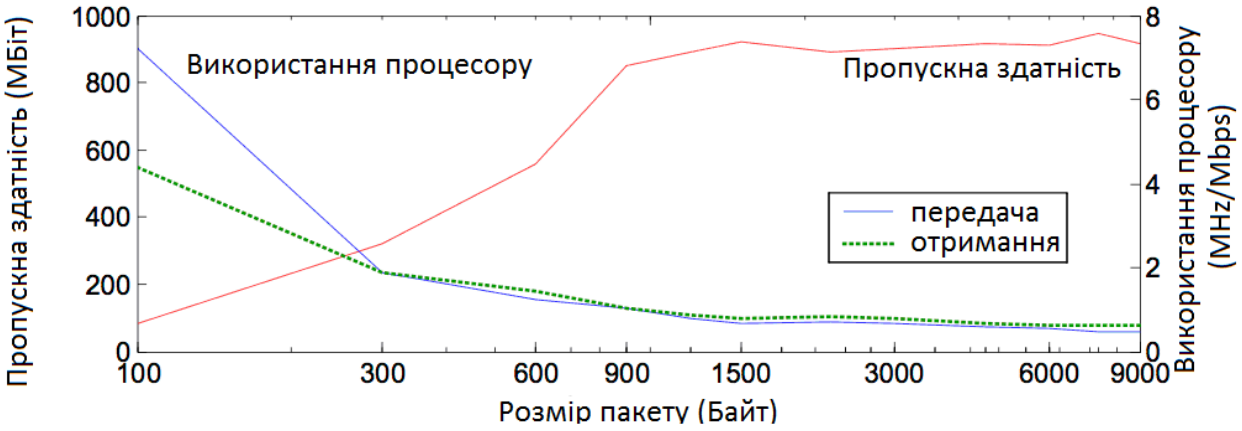


Рис. 3.2. Залежність продуктивності UDP від розміру пакету, побудована самостійно.

Графік залежності пропускної здатності від розміру буфера (рис. 3.2) демонструє, що є деяке оптимальне значення розміру буфера, і подальше збільшення його розмірів не дають приросту пропускної здатності, а навпаки, відбувається зниження. Практично ж, розмір буферу на стороні передавача може бути значно меншим, ніж на стороні отримувача тому що:

- великий буфер збільшує RTT;
- великий буфер збільшує можливе перевантаження мережі.

Перша ситуація викликає більш серйозні проблеми, коли відбувається втрата пакетів, в той час, як друга викликає більше втратів пакетів. Розмір буфера на стороні відправника повинен бути достатньо великим, щоб не було обмежень на швидкість передачі пакетів. Це мінімальне значення зв'язане зі значенням RTT.

Залежність на рисунку (див. рис. 3.2) демонструє, що розмір пакету повинен бути рівний розміру MTU. Хоча видно, що більший розмір пакетів приводить до меншої завантаженості процесора (оскільки є менше накладних витрат на обробку).

Також, підвищення продуктивності забезпечується зменшенням розмірів пакетів. Цей метод може бути використаний, щоб уникнути додаткового копіювання пам'яті, уникаючи використання тимчасового буферу для упаковки і розпаковування пакетів і даних користувача.

На відміну від стандартних протоколів транспортного рівня TCP і UDP, протокол UDT не входить до стеку протоколів OSI і тому ще не є стандартом де-

факто для комп’ютерних мереж. Це обмеження потребує додаткової програмної реалізації цього протоколу. Його базування на стандартних протоколах TCP і UDP дещо спрощує цю задачу.

Протокол UDT – надійний транспортний протокол передачі потокових даних орієнтований на з'єднання на рівні додатків, який реалізований мовою програмування C++. Архітектура протоколу зображена на рис. 3.3. Протокол UDT містить п'ять функціональних компонентів: модуль API, відправник, одержувач, слухач, а також канал UDP, дана функціональна структура представлена на рис. 3.4. Протокол UDT також включає в себе чотири компоненти даних: буфер протоколу відправника, буфер протоколу одержувача, список втрат відправника і список втрат одержувача. Протокол UDT є двостороннім промисловим протоколом, усі його елементи мають однакову структуру [56].

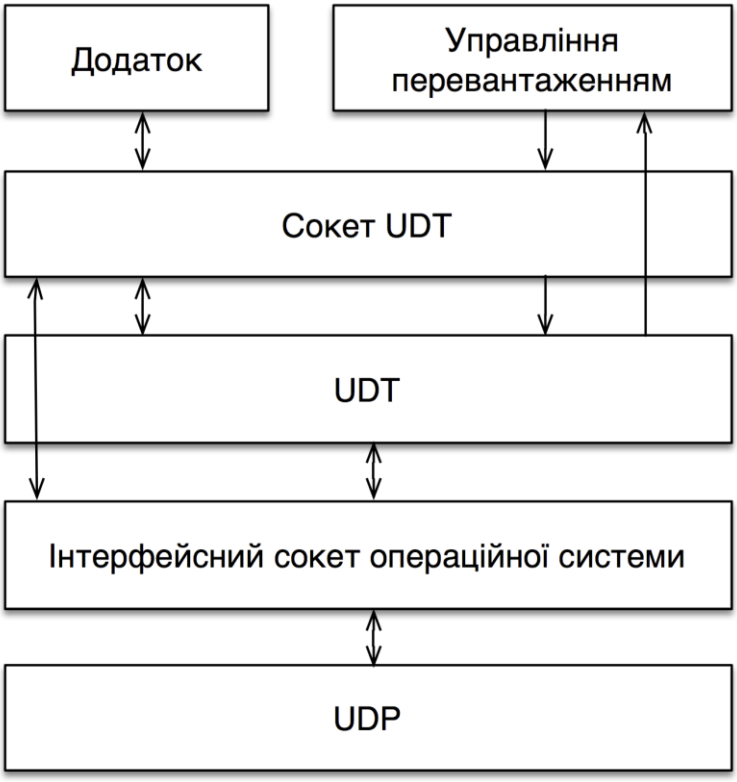


Рис. 3.3. Архітектура UDT.

При роботі з протоколом користувач використовує два типи пакетів: пакети даних і пакети управління. Вони відрізняються першим бітом в заголовку пакету. Пакет даних UDT включає в себе пакетний порядковий номер, порядковий номер повідомлення, і штамп часу з моменту ініціювання з'єднання [58].

Модуль API відповідає за зв'язок з додатками. Дані для відправки передаються в буфері відправника і передається відправником у канал UDP. Приймач зчитує дані з каналу UDP і розміщує їх в буфері приймача, сортує дані і перевіряє втрати пакетів. Додатки можуть читати отримані дані з буфера приймача. Приймач також зберігає отриману інформацію управління. Він оновлює список втрат відправника, коли NAK прийнято і список втрати приймача при виявленні втрати. Деякі події управління ініціюють приймач оновити модуль управління перевантаженням [55].

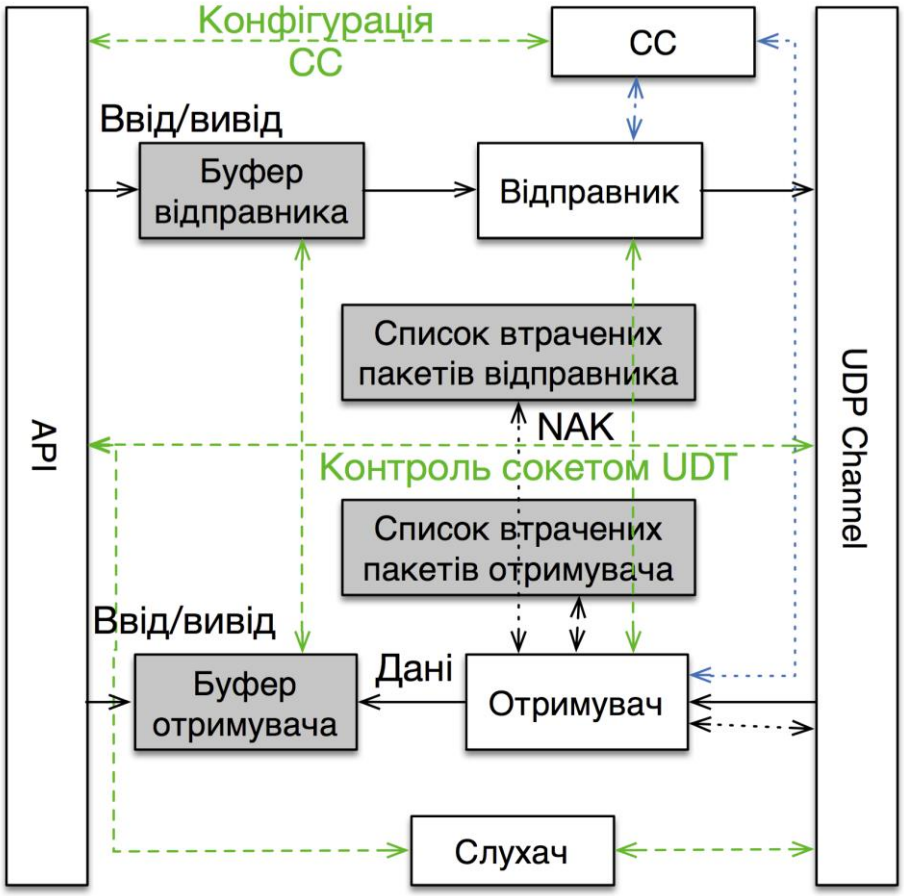


Рис. 3.4. Функціональна структура протоколу UDT.

Протокол UDT використовує керуванням перевантаженням на основі зміни швидкості та зміною розмірів вікна на основі управляння вихідним трафіком. Управління швидкістю оновлює період відправки пакету кожен інтервал часу, в той час, як управління потоком оновлює розмір вікна щоразу після отримання підтвердження. Протокол UDT використовує таймер на основі підтверджень (АК), створює підтвердження з інформацією про те, чи є нові безперервно отримані

пакети даних. UDT відправляє набір пакетів даних з фіксованим розміром пакетів, якщо немає достатніх даних для відправки. Фіксований розмір можна налаштувати додатково і оптимальне значення є максимальний розмір блоку (MTU) [59].

Для практичного використання протоколу транспортного рівня UDP на сеансовому рівні доречно створити надбудову, яка проводить структуризацію інформаційних потоків і спрощує використання даного протоколу в практичних додатках.

Надбудова над протоколом UDT, що фактично стає протоколом сеансового рівня, розширює функціонал протоколу, забезпечуючи користувачу використання не тільки передачі даних в сирому вигляді, але й для передачі файлових структур даних.

Заголовок пакету протоколу має наступну структуру (рис. 3.5):

- 1 байт – номер версії протоколу (для можливості модифікації протоколу, та зворотньої сумісності);
- 1 байт – ідентифікатор команди яка передається;
- 2 байти – ідентифікатор каналу (для мультиплексування);
- 3 байти – кількість 16-байтних блоків у пакеті;
- 1 байт – довжина додаткового заповнення пакету (для забезпечення кратності довжині ключа шифрування);
- 8 байт – службова інформація пакету даних.

Рис. 3.5. Структура заголовку пакету даних протоколу, розроблено самостійно.

Для забезпечення можливості шифрування даних, довжина пакету повинна бути кратна довжині ключа шифрування. Дана вимога не є вимогою протоколу передачі – це вимоги криптографічних алгоритмів, а тому приймається аксіоматично.

Для забезпечення повноти функціоналу протоколу сеансового рівня вводяться наступні типи команд, що повністю відсутні в первинній версії протоколу:

- віддалене створення директорії;
- віддалене створення файлу;
- передача блоку файлу;
- передача закінчення файлу.

Ефективність використання ресурсів вузлів передачі та отримання досягається передачею блоків файлу. Розмір блоків повинен бути кратний розміру буфера файлової системи.

Згідно з запропонованою надбудовою над транспортним протоколом і рекомендаціями щодо практичного використання, передача одного файлу (при встановленому з'єднанні) проводиться за наступним алгоритмом (рис. 3.6):

1. Відправка пакету «віддалене створення файлу».
2. Передача змісту файлу:

- Вибір розмір блоку файлу.
- Поділ файлу на блоки.
- Послідовна відправка пакету «передача блоку файлу» для кожного блоку.

3. Відправка пакету «передача закінчення файлу».
4. Очікування підтвердження закінчення отримання файлу.

З приймальної сторони алгоритм роботи наступний (при встановленому з'єднанні):

1. Отримання пакету «віддалене створення файлу.
2. Створення файлу на файловій системі.

3. Отримання змісту файлу:

- Отримання пакету «передача блоку файлу» по блочно.
- Запис блоку файлу в файлову систему.

4. Отримання пакету «передача закінчення файлу».
5. Закриття файлу.
6. Відправка підтвердження закінчення отримання файлу.

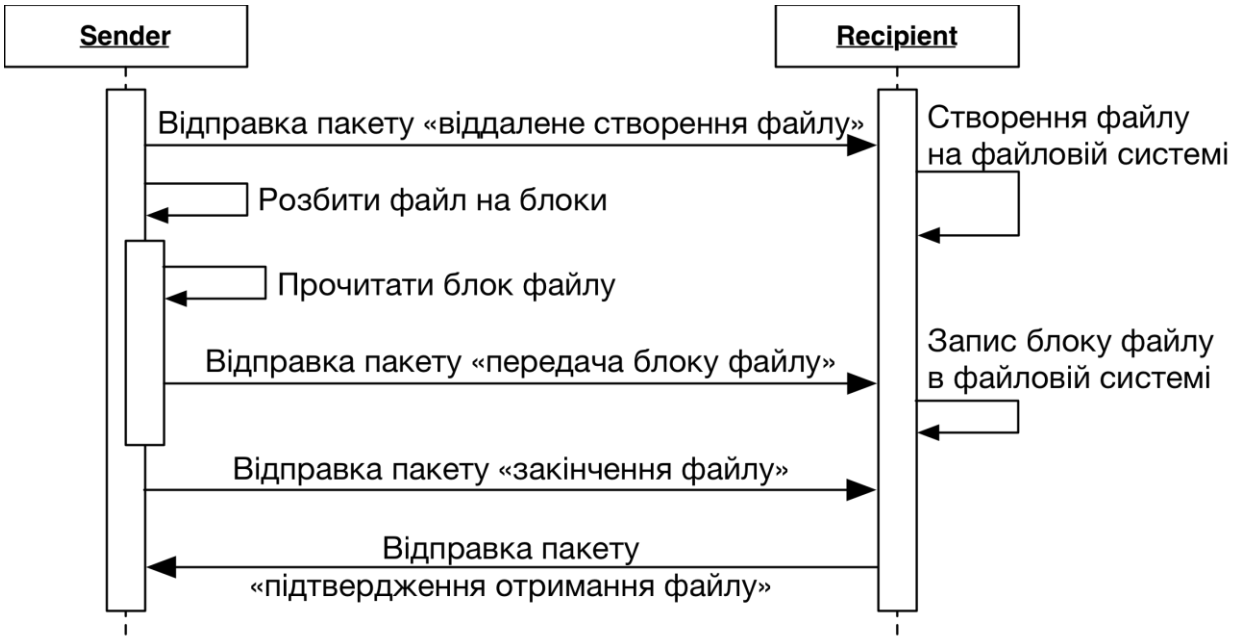


Рис. 3.6. Діаграма послідовності передачі даних протоколом сеансового рівня, розроблено самостійно.

Використання запропонованої методики дає можливість оптимальної та ефективної реалізації протоколу на базі UDP, для передачі команд файлової системи в хмарковому сховищі даних. На базі чого можна реалізувати ефективні та практичні додатки.

Перевагою даного протоколу є:

- вбудовані команди файлової системи;
- використання високорівневих команд;
- можливість мультиплексування різних даних по одному каналу зв’язку;
- можливість передачі одного файлу по багатьох каналах зв’язку одночасно.

3.2. Розробка методу мультипротокольного доступу до потокових даних.

Сучасні додатки виробляють дані з дуже великою швидкістю, і ця швидкість зростає з кожним роком, особливо при переході до хмаркових технологій. Недивлячись на постійне підвищення пропускної здатності мереж зв'язку, зокрема Internet, користувачам потрібна можливість доступу до даних з все меншими затримками. Цьому, зокрема сприяють досягнення в галузі апаратури комп'ютерів (здешевлення основної і дискової пам'яті, збільшення кількості ядер процесорів), але цього недостатньо для задоволення потреб підвищення пропускної здатності при одночасному зниженні часу затримки.

Методи нарощування потужності додатків для роботи з хмарковими технологіями повинні бути достатньо простими, по-перше, оскільки повинна забезпечуватися можливість їх швидкої зміни та масштабного впровадження з мінімальними зусиллями, а по-друге, тому що люди, які створюють хмаркові додатки, є спеціалістами широкого профілю, і вони не готові вивчати складні технології, що використовують системні програмісти і важкі в налаштуванні.

Виконання різноманітних дій над потоками даних – це нова технологія, яка забезпечує обробку даних, що дуже швидко надходять, і виробляє результати з невеликими затримками. В основному, такий підхід почав використовуватися в спільноті дослідників баз даних, і тому він має деякі характеристики, що характерні для реляційних систем баз даних. В традиційних системах баз даних спочатку дані поступають в базу даних і зберігаються на дисках, а потім вже користувачі застосовують до цих збережених даних запити. Аналогічно відбувається і при роботі з хмарковими сховищами даних – спочатку дані поступають на один з сателітів, а вже потім проводиться їх переміщення на сховища даних та реплікація.

Послідовна передача блоків файлів, навіть при паралельній їх фізичній передачі високошвидкісними каналами зв'язку, стає вузьким місцем використання хмаркових сховищ. Приймач не може використовувати файл до того моменту, поки він повністю не буде переданий. Нічого страшного в цьому немає у

випадках, коли, або файл невеликий, або файл передається безпосередньо від одного абонента іншому. При існуванні в системі передачі достатньої кількості проміжних елементів приймання-передавання, на кожному з них буде відбуватися затримка на передачу до того моменту, поки весь файл не буде прийнятий проміжним елементом. Можна сказати, що при традиційному підході дані обробляються в стані спокою, а в системах обробки запитів над потоковими даними – на льоту.

Переваги таких методів обробки, що проявили себе з позитивного боку в системах баз даних, доцільно розширити на хмаркові сховища, враховуючи особливості функціонування останніх.

Так для цілей хмаркових сховищ даних необхідно:

- Організація асинхронної передачі файлів багатьма каналами зв’язку;
- Організація потокового читання файлу і його передача;
- Організація прийому файлу через декілька каналів зв’язку та кешування його для подальшого запису;
- Організація потокового запису файлу з кешу;
- Організація синхронного підтвердження для завершення передачі файлу.

Так як методику зіставлення даних на льоту переносимо з існуючих методів обробки запитів до баз даних на льоту, то варто провести аналогію термінів і операцій.

У реляційних базах даних основним примітивом є таблиці. Таблиця заповнюється записами, кожна з яких має один і той же тип запису, який задається декількома іменованими строго типізованими стовпцями. В середині записів відсутній будь-який внутрішній порядок стовпців. При виконанні запитів, які звичайно представляються мовою SQL, вибираються записи з однієї або декількох таблиць, і вони перетворюються з використанням невеликого набору потужних реляційних операцій.

Під час використання методу обробки запитів над потоковими даними відповідними примітивами вже виступають потоки. Як і в таблиці, у потока є деякий тип запису, але записи не зберігаються, а поступають в потоці. У потоковій системі записи впорядковані природнім чином, і в кожного запису є часова мітка, що вказує, коли цей запис був створений. Для реляційних операцій, які підтримуються в реляційних базах даних, є аналоги і в потокових системах, і ці аналоги достатньо схожі на реляційні операції, щоб для формулювання потокових запитів можна було використовувати SQL. Аналогічний підхід, але вже не з використанням мови структурованих запитів, а «мови передачі та обробки файлів у хмаркових сховищах» дозволяє використати усі переваги потокового підходу, який, як показала практика використання, є набагато ефективніший.

Тобто, процес передачі файлу F від одного абонента іншому полягає у поділі його на K блоків $F = \{F_1, \ldots, F_K\}$ та послідовній передачі цих блоків через мережу. У свою чергу, приймальна сторона забезпечує приймання чітко визначеної послідовності блоків, зібрання з них файлу та надання доступу до нього користувачу (рис. 3.7). Послідовність передачі визначається тим, що наступний блок не може бути переданий, поки не надійшло підтвердження про приймання попереднього. Тобто, якщо система передачі і має декілька каналів передачі, така дисципліна передачі обмежує швидкість передачі файлу.

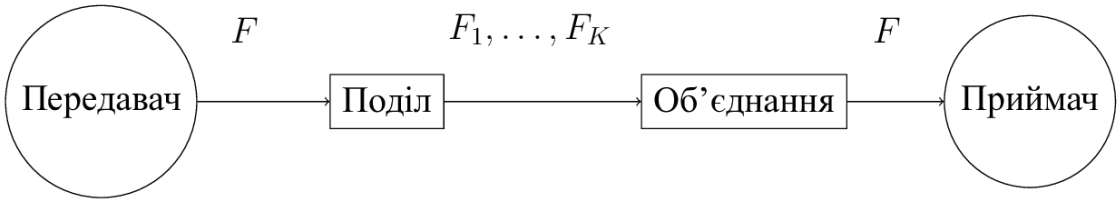


Рис. 3.7. Послідовна передача файлу.

Тобто, час через який кінцевий користувач отримає доступ до файлу визначається сумарним часом передачі усіх блоків файлу. При великих файлах він достатньо суттєвий.

Особливо зростає час доступу до файлу, коли в ланці передачі розміщені

проміжні сервери, які буферизують передачу (даний підхід використовується в хмаркових сховищах). Тоді час доступу до інформації замітно зростає.

Усі системи для роботи в мережі використовують аналогічний принцип передачі. І змінити цей принцип – означатиме зміна всього програмного забезпечення, що виглядає абсурдним.

Інша справа, коли файли передаються через хмаркоі сховища даних, для яких створюється нове програмне забезпечення. Для сумісністю з існуючими методами потрібно лише забезпечити, щоб системи приймання-передачі на сателітах:

- дозволяли отримувати послідовний потік блоків файлів для приймання від передавального абонента, і
- генерували відповідний послідовний потік блоків файлів для приймального абонента.

Усе, що відбуватиметься в середині хмарки для зовнішніх абонентів не повинно бути доступним. Тобто задача модифікації полягає у зміні передачі файлів між хмарковими серверами (сателітами та сховищами) для підвищення швидкості обміну.

Для доступу до даних в такому мультипротокольному середовищі при поділі файлу на блоки, кожен з них містить offset, що визначається місце з якого починається блок даних у файлі.

Передача між хмарковими абонентами проводиться паралельно за декількома каналами зв'язку. При чому різні канали можуть мати різну швидкість передачі. Відповідно на приймальній стороні хмаркового елемента блоки файлу можуть надходити не у впорядкованому порядку, а випадковим чином. Для відновлення з цих блоків кінцевого файлу прийняті блоки поміщаються в кеш блоків, який організовуємо за структурою пріоритетної черги (пріоритет визначається offset). Таким чином в процесі приймання блоків у кеші формується кінцевий файл.

Тобто, між двома точками мережі передача даних здійснюється через N TCP-з'єднань.

Оскільки, як було наголошено, потрібно використати асинхронну передачу, то варто використати IOLOOP (цикл активних вхідних/вихідних/помилкових з'єднань). Алгоритм роботи IOLOOP є:

1. Створення циклу IOLOOP.
2. Створення необхідних з'єднань (sockets).
3. Призначити події кожного socket з вказанням, які саме події цікавлять (читання/запис/помилка сокета).
4. Звернення до IOLOOP (з вказанням таймауту), щоб отримати доступний сокет, який відповідає призначеним умовам.
5. Якщо протягом певного часу (таймауту) жодний socket не відповідає умовам, керування повертається до програми. Якщо якийсь готовий, то IOLOOP разом з керування повертає список готових з'єднань і з вказанням умови (читання і/або запис і/або помилка сокету)

Такий принцип достатньо легко реалізувати за структурою (рис. 3.8).

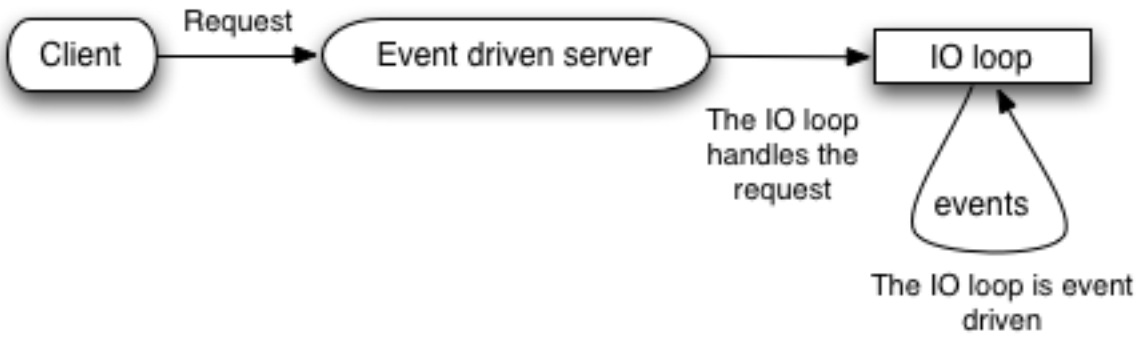


Рис. 3.8. Метод отримання доступу до асинхронної передачі.

Використовую цей підхід, дало змогу реалізувати метод мультипротокольного доступу до потокових даних.

1. Звернення до циклу IOLOOP, яке повертає готовність до читання з керуючого сокету (сокет керування програмою).
2. Зчитування сокету керування та отримання команди відправки файлу.
3. Зчитування з файлу M байт.

4. Упаковка даних у пакет.

5. Додавання пакету у чергу.

6. Якщо черга не заповнена -- повернутися до пункту 2.

7. Перевірка на існування N з'єднаннь. Якщо вони закриті, не створені -- то створити і виконати необхідні дії для з'єднання. Після з'єднання сокети добавляються в цикл IOLOOP з вказівкою очікування: «готовності запису», «готовність читання», «помилки сокета».

8. Звернення до IOLOOP за списком «готових» сокетів.

9. Якщо помилка сокету: закривають сокет, видаляють його з IOLOOP, помічають пакети, що були передані цим сокетом і не підтверджені, як не відправлені, створюють новий сокет, утворюють новий сокет та добавляють його в IOLOOP.

10. Якщо готовність сокету до запису, для кожного з сокетів: отримати пакет з черги, шифрування його та запис у буфер обміну сокету, а сам пакет помічають як відправлений через даний сокет.

11. Якщо готовність сокету до читання, тоді для кожного сокету: прочитати пакет, розшифрувати, отримати підтвердження пакету; видалити даний пакет з черги.

12. Якщо файл ще не весь прочитаний, та є місце в черзі, тоді перейти до 3.

13. Якщо файл прочитаний та черга пуска, закрити з'єднання та видалити їх з IOLOOP та перейти до 1.

14. Перейти до 8.

Окрім того, що пріоритетна черга дозволяє робити збір файлу з блоків в процесі їх приймання, не зважаючи на послідовність поступлення блоків, такий метод дозволяє паралельно проводити послідовну подальшу передачу файлу в момент надходження відповідного послідовного блоку.

Узагальнена схема роботи цього методу може бути представлена графічно (рис. 3.9).

До основної переваги запропонованого методу слід віднести те, що транспортування даних використовується більш швидший протокол UDP, а для гарантування доставки – TCP.

Рис. 3.9. Діаграма послідовності мультипротокольного доступу до потокових даних, розроблено самостійно.

Фактично, для транспортування даних використовується засіб, який при створенні був призначений для контролю передачі, а для контролю – той, що призначений для самої передачі. Таким чином, використовуючи існуючі протоколи транспортного рівня, було підвищено швидкість передачі на великі відстані потокових даних.

Враховуючи все вище викладене, можна сказати, що перевагами такого модифікованого методу є:

- Можливість потокової передачі даних через N-вузлів (серверів) хмаркових сховищ, без очікування завершення передачі одиночного файлу.
- Однак, запропонований підхід має певні недоліки:
- Динамічне збільшення кешу, при очікуванні правильної послідовності блоків файлу, для подальшого потокового запису;
- Синхронне підтвердження завершення запису файлу.

3.3. Розроблення методу мультиплексування різних джерел даних для одночасної передачі.

Обсяги інформації та бази даних, з якими працюють сьогоднішні підприємства і компанії з кожним роком зростають у геометричній прогресії. Зростають запити керівників підприємств не тільки до швидкості роботи машин з величезними обсягами інформації, але й до якості, надійності зберігання даних, накопичувачів інформації. Саме тому правильно обрана система зберігання даних – дуже важлива складова успіху в бізнесі. Використання хмаркового сховища даних, дозволяє не тільки забезпечити резервування даних, але й розміщення їх поблизу споживача даних.

При виборі високошвидкісних протоколів передачі даних, слід звернути увагу на можливість мультиплексування документів, та отримання одного документа з різних джерел.

Мультиплексування документів необхідне для одночасного отримання декількох документів з одного джерела. При використанні почергового отримання документів, користувач може стикнутися із проблемою, коли отримання досить малих робочих документів, може відкластися на великий проміжок часу через попадання в чергу великих документів [4].

Існують декілька видів забезпечення мультиплексування:

- Канальне мультиплексування – документи діляться на пакети, кожний пакет отримує ідентифікатор документу (каналу передачі даних), який однозначно з’язує декілька пакетів, що стосуються одного документу.

- Мультиплексування з розподілом у часі – мультиплексор в кожний момент часу видає в загальний канал дані одного документу, віддаючи йому всю смугу пропускання, але по чергово для документів через рівні проміжки часу.
- Мультиплексування з розподілом за з’єднаннями – між двома точками передачі даних створюється стільки з’єднань, скільки потрібно одночасно передавати документів.

Оскільки використання хмаркових сховищ даних повинно забезпечувати швидкий доступ до клієнтських документів з різних куточків світу, а вузли хмаркового сховища можуть знаходитися достатньо «далеко» (поганий/повільний канал з’язку) доцільним є використання методів отримання документу з різних джерел та зіставлення його на клієнтській частині.

Існує два підходи до забезпечення даного режиму роботи:

1. Отримання одного файлу з одного вузла – клієнтське програмне забезпечення робить запити до вузла для отримання цілого документу (рис. 3.10). Таким чином документи приходять цілісно. Даний метод ефективніший коли на сховищі розміщено багато файлів невеликого розміру.

2. Отримання одного документу з різних вузлів – у даному випадку клієнтське програмне забезпечення само вирішує яку частину документу з якого вузла отримати (рис. 3.11). Таким чином з «ближчих» (з якими є кращий/швидший зв’язок) вузлів можна отримати більші частини документу. Даний метод ефективніший у випадку коли необхідно отримати один достатньо великий файл, а пропускна здатність вхідного каналу передачі даних значно перевищує передачу даних з одним вузлом.

Для забезпечення одночасного доступу до даних через один і той же сателіт різних користувачів, виникла необхідність у створенні методу мультиплексування даних в межах протоколу передачі даних. Даний метод забезпечує одночасну незалежну передачу різних даних.

$$\begin{aligned}&\langle d_1 f_i\rangle \to \{\langle d_1, c_1, f_i\rangle, \langle d_1, c_2, f_i\rangle, \ldots, \langle d_1, c_n, f_i\rangle\} \to \\ &\{\langle c_1, p_1\rangle, \langle c_2, p_2\rangle, \ldots, \langle c_n, p_m\rangle\} \to \{p_1, p_2, \ldots, p_m\} \to \langle d_2 f_i\rangle\end{aligned}. \quad (3.1)$$

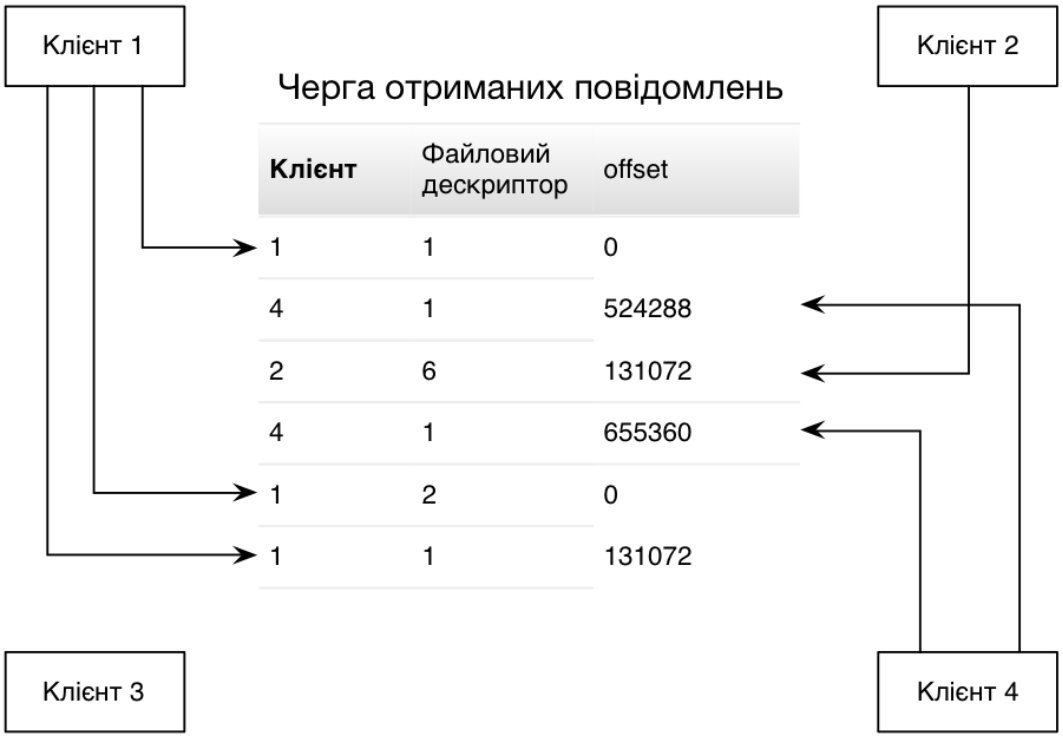


Рис. 3.10. Мультиплексування з одним вузлом, розроблено самостійно.

Для вирішення даної проблеми без мультиплексування даних існують наступні варіанти реалізації:

- Створення багатьох каналіз зв'язку, що створює накладні ресурси та втрати швидкодії системи.
- Реалізація почергової передачі даних, що може створити складкості коли один клієнт для отримання достатньо малої кількості даних очікує поки канал зв'язку звільниться від передачі даних клієнтом з більш великим об'ємом трафіку.

$$\begin{aligned}&\{\langle d_1, f_i\rangle, \langle d_2, f_i\rangle, \ldots, \langle d_l, f_i\rangle\} \to \\ &\{\langle d_1, c_1, f_i\rangle, \langle d_1, c_2, f_i\rangle, \ldots, \langle d_l, c_n, f_i\rangle\} \to \\ &\{\langle d_1, c_1, p_1\rangle, \langle d_1, c_2, p_2\rangle, \ldots, \langle d_l, c_n, p_m\rangle\} \to \{p_1, p_2, \ldots, p_m\} \to \langle d_k, f_i\rangle\end{aligned}. \quad (3.2)$$

Метод мультиплексування для протоколу UDT розроблено на основі базових принципів мультиплексування, та його складових: мультиплексор та демультиплексор.

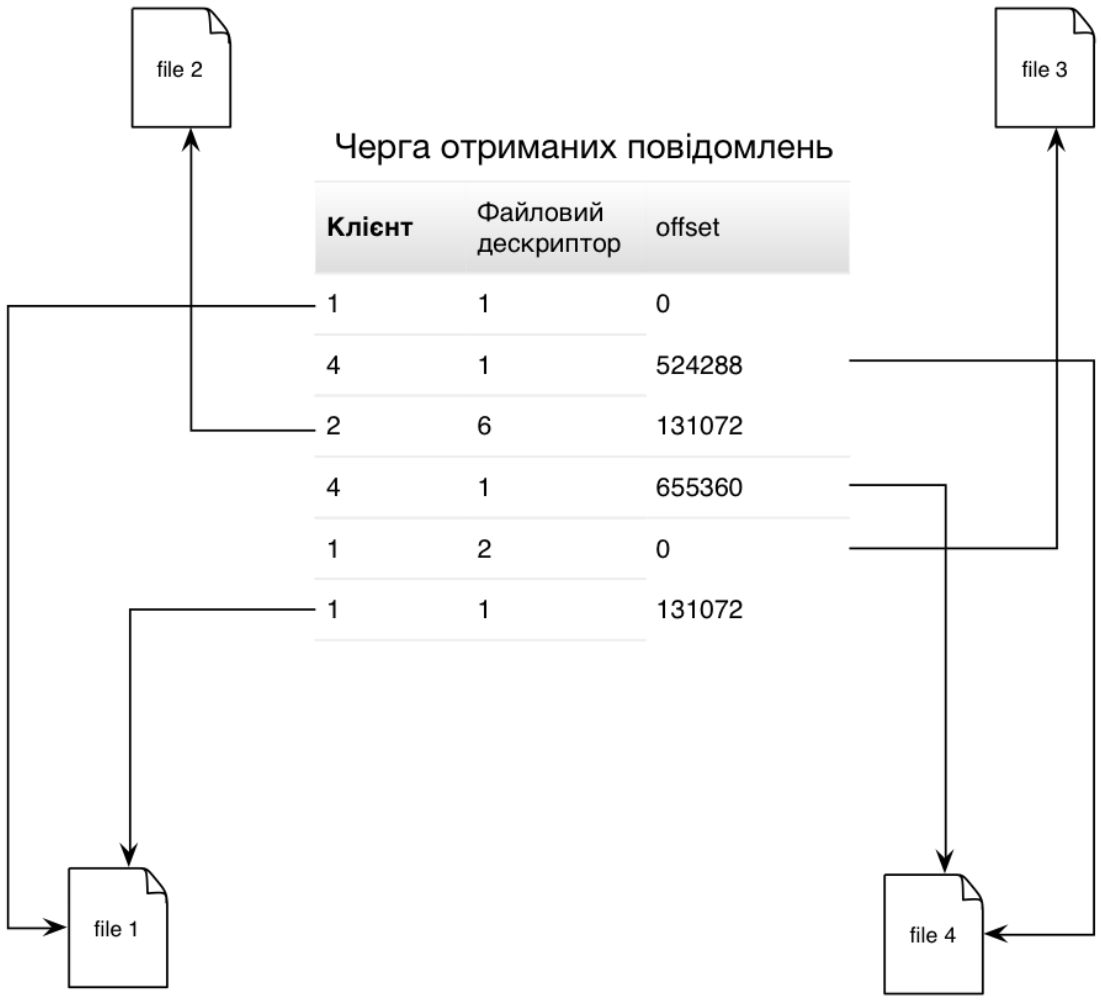


Рис. 3.11. Мультиплексування з різними вузлами, розроблено самостійно.

В результаті застосування даного методу ми отримуємо гнучку системи передачі даних з конкурентним доступом до них. Та можливість ефективно використовувати ресурси при передачі даних.

3.4. Розроблення методу вибору шлюзу за складністю запиту.

Сучасні мережі доставки та дистрибуції контента здатні здійснювати автоматичний контроль цілісності даних на кожному з серверів мережі. При цьому гарантується 100 % доступність контенту для кінцевого користувача в разі втрати зв'язності між вузлами мережі, виходу з ладу центрального або віддаленого сервера.

Найбільш розвинені комерційні CDN надають статистичний контроль процесів доставки та дистрибуції контенту. Контент-провайдер в реальному часі може отримати всю необхідну інформацію про завантаження, доступності і популярності свого контенту в кожному регіоні присутності. Але статичний

контроль не враховує мобільність клієнтів, які можуть змінювати своє географічне розміщення в часі.

Тобто, реально, модель хмаркового сховища представляється моделлю дистрибуції засобами CDN (рис. 3.12). Сателіти знаходяться поблизу отримувачів даних і призначені для збільшення швидкості доступу до них. Коли клієнт робить DNS запит на отримання IP адреси системи для обміну даними, інтелектуальна система вибору маршруту оцінює фактори (геолокація користувача, шлях проходження запиту від клієнта до DNS-сервера, навантаженість мережі та сателітів, і т.д.) та надає клієнтові IP-адрес сателіта, який находиться «найближче» до споживача послуг.

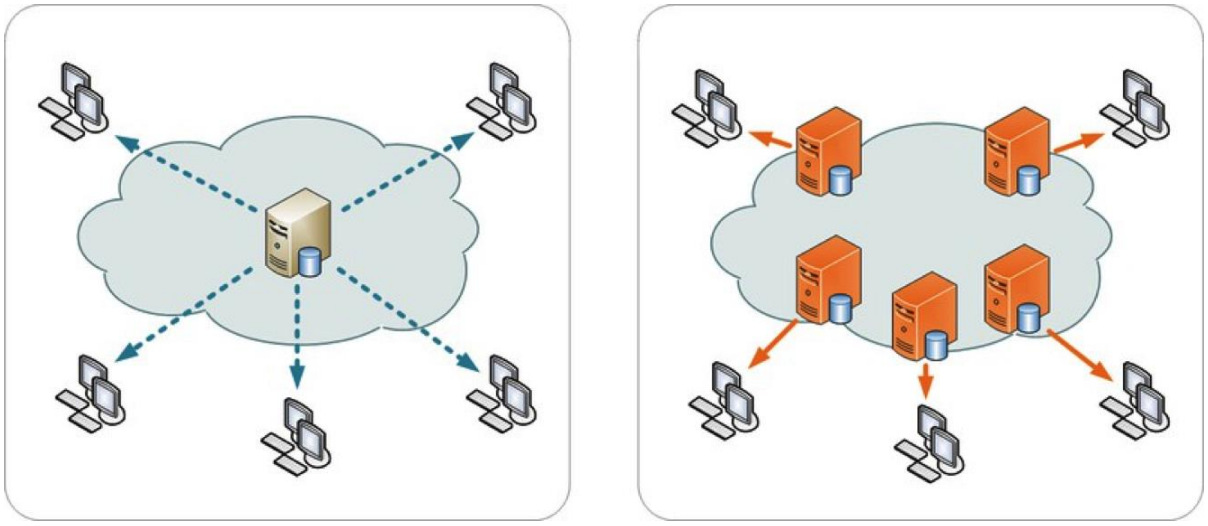


Рис. 3.12. Одноточковий дистрибуція (ліворуч) і дистрибуція засобами CDN (праворуч).

Метод вибору шлюзу за складністю запиту (I_{cc}) базується на понятті складності запиту – ступеню невизначеності інформації про реальну швидкість обміну даними між клієнтом і хмаркою.

Складність виконання запиту – добуток завантаженості хмаркового сховища даних на суму RTT для всього маршруту запиту (BGP_PATH).

$$FQ(q) = CH(S_{cloud}) \cdot \sum_{i=1}^{K} RTT_i \,. \qquad (3.3)$$

Для обчислення використовуємо: RTT (Round-trip delay time) - час від моменту посилки запиту до моменту отримання відповіді, BGP_PATH - шлях

проходження запиту.

При надходженні запиту від клієнта до сателіту, сервер отримує наступні дані про клієнта:

- IP-адресу клієнта (в даному випадку унікальний ідентифікатор клієнта. У випадку якщо 2 клієнти мають однаковий IP-адрес вважається, що це один і той самий клієнт), і
- локальний час системи користувача.

На основі цих даних отримано:

- RTT (Round-trip delay time) – час від моменту посилки запиту до моменту отримання відповіді;
- BGP_PATH – мережний шлях проходження запиту;
- географічне місце положення клієнта (за допомогою GeoIP DB).

Формально задача описується:

$$I_{cc}(q) = fi(FQ(q), BGP\_PATH, U, PDB_{sat}, PDB_{stor}), \quad (3.4)$$

де U – інформація про розміщення користувацьких даних з запиту q,

PDB_{stat} – база даних розміщень стателітів,

PDB_{stor} – база даних розміщень сховищ зберігання даних.

Для вирішення даної проблеми необхідно при авторизації клієнта до одного із серверів визначити який саме сателіт знаходиться до нього наийближче.

При надходженні запиту від клієнта до сателіту, nginx отримує наступні дані про клієнта:

1. IP-адреса клієнта (в даному випадку унікальний ідентифікатор клієнта. У випадку якщо 2 клієнти мають однаковий IP-адрес вважаємо, що це один і той самий клієнт).

2. RTT (Round-trip delay time) – час від моменту посилки запиту до моменту отримання відповіді.

3. BGP-PATH – шлях проходження запиту (шлях – список провайдерів інтернету через які проходить запит).

Nginx передає ці дані на програмну реалізацію сателіта.

У програмній реалізації клієнта на основі отриманих даних можна отримати шляхи від себе і кожного із сателітів до клієнта (BGP-PATH). Тобто, отримати список усіх можливих шляхів від клієнта до потрібної інформації. Також можна визначити географічне місце положення клієнта маючи GEOIP (рис. 3.13).

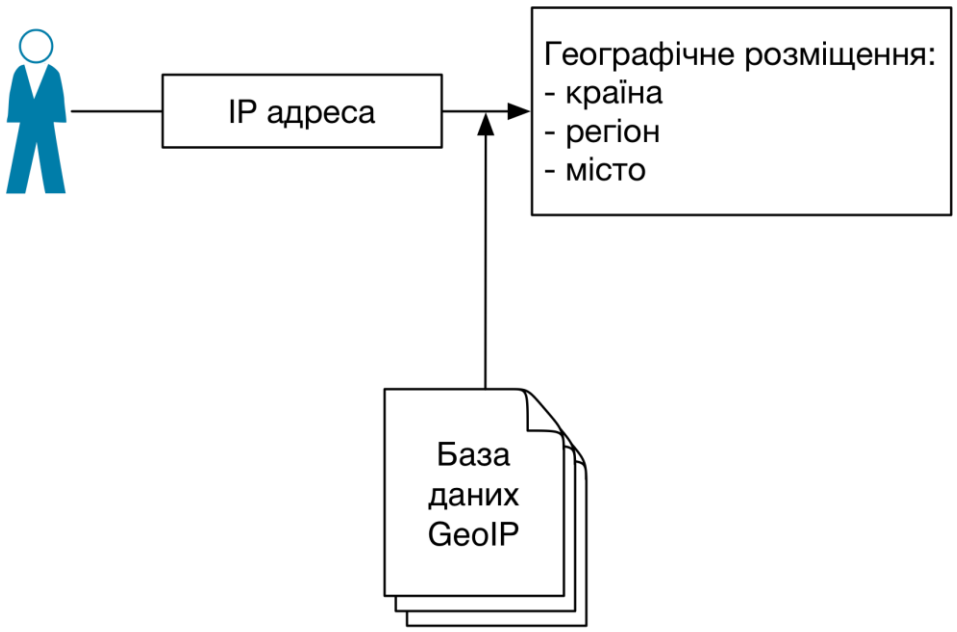


Рис. 3.13. Метод визначення географічного розміщення користувача.

Але отримати RTT від інших серверів до клієнта можна лише в тому випадку якщо клієнт заходив коли небуть на них і ця інформація була збережена. Тому потрібно вести статистику відвідування клієнтів з наступними даними:

1. IP-адрес клієнта;
2. IP-адрес сателіту;
3. BGP-PATH;
4. RTT.

Крім того потрібно враховувати завантаженість сателітів на відповідний час роботи з клієнтами.

При відсутності статистичних даних, на основі яких можна прийняти рішення про оптимальний шлях від клієнта до сателіта, можна спробувати оцінити цей шлях за іншою доступною інформацією. Якщо клієнт знаходиться досить близько (GEOIP) до іншого клієнта та їхні BGP-PATH (до одного конкретного сателіту) з достатнім наближенням можна вважати що їхні RTT до цього сателіту рівні.

Маючи ці дані можна визначити мінімальний RTT від клієнта до сателіту. На основі цього відбувається підключення клієнта до його інформаційного сховища через мінімальний шлях, перенаправивши його на ближчий і незавантажений на даний момент сателі.

Зберіганням та обробкою інформації при даному методі займається основний сервер. Основні задача сателітів забезпечити обмін інформацією з користувачем таким чином, щоб він виявився якомога «ближче» до споживача (рис. 3.14).

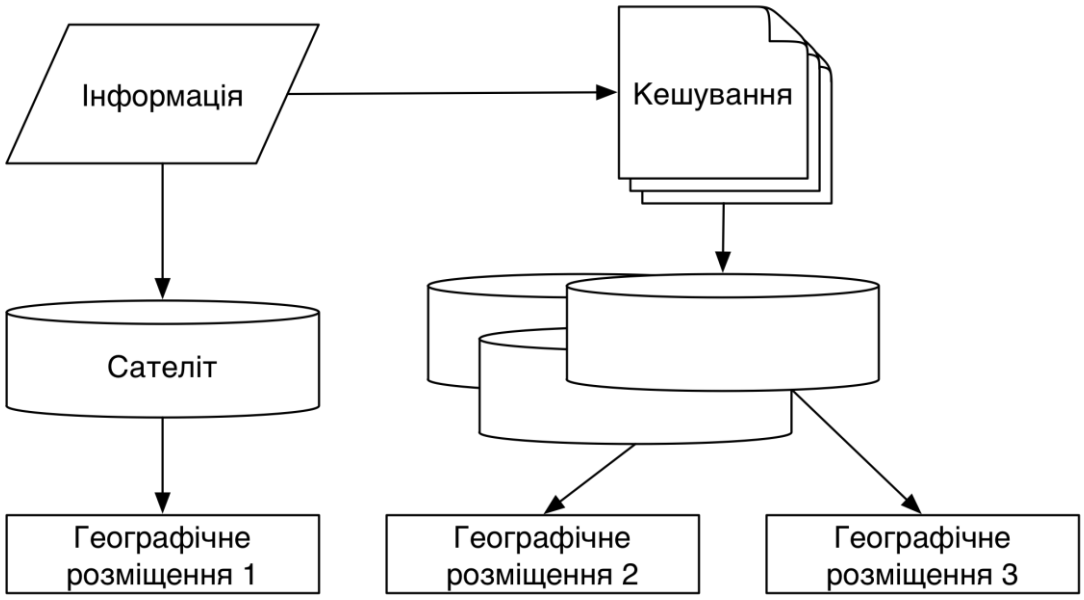


Рис. 3.14. Метод наближення даних до клієнта, розроблено самостійно.

Крім самого факту, що інформація буде доставлена користувачеві чи прийнята від нього, потрібно забезпечити якість доставки. Це передбачає максимально можливу швидкість обміну цією інформацією.

Метод також повинен передбачати можливість “наблизити” інформацію до користувача географічно. Це пов'язано з пропускною здатністю каналів (відсутністю іноді хороших магістральних каналів), а також з різницею у вартості локального і зовнішнього трафіку для кінцевого користувача.

Таким чином, усвідомивши необхідність географічного розподілення для інформації, встановлюють сервери-сателіти в безпосередній близькості від споживача.

У абстрактному варіанті в процесі доставки інформації беруть участь дві сутності: основний сервер і користувач. Необхідно для початку дізнатися, де знаходиться користувач інформації. Сподіватися, що він сам розповість цю інформацію про себе, не доводиться, тому беруть IP-адресу користувача і виконують пошук в базі даних GeoIP (яких сьогодні досить багато, як платних, так і безкоштовних), і отримують на виході інформацію про місцезнаходження користувача: його країну, регіон, місто, назва його провайдера.

Інформація користувача розташована завжди на конкретному основному сервері, а сервер має своє фізичне місце розташування, заздалегідь нам відоме. Тому інформація через сервер теж має своє географічне місце розташування: ті ж країна, регіон, місто і т.п. Крім того, при бажанні користувача, можна створювати копії інформації на інших основних серверах, таким чином одна й та ж інформація буде доступна на декількох основних серверах, а значить, в кількох географічних точках.

Тепер є клієнт, його місце розташування, інформація, з якою він хоче працювати, а також сервери-сателіти з місцями їх розташування. Тепер необхідно вибрати саме той сателіт через який буде відбуватися обмін інформацією з користувачем, який найближче до нього.

Логічно було б порахувати відстань від користувача до інформації і всіх сателітів, що забезпечать до неї доступ, і вибрати самий близький до користувача сателіт.

Ця відстань не завжди збігається з відстанню на карті між двома точками, і є швидше мірою якості і пропускної спроможності каналів між регіонами, країнами або містами (або навіть між окремими провайдерами).

Таким чином, отримують зважений орієнтований граф, на якому необхідно вирішити задачу пошуку найкоротшого шляху (мінімізація суми ваг ребер графа, що входять в цей шлях). Це класична задача, яка може бути легко вирішена за поліноміальний час. Граф хоч і є слабозв'язаний, але все-таки його розміри досить великі, тому в реальному часі для кожного відвідувача виконувати такий розрахунок не є найбільш ефективним рішенням. Але можна трохи «схитрувати»: відомо, що

при всіх пошуках найкоротшого шляху кінцевим пунктом шляху будуть місця, де розташовані сателіти, таким чином, число різних кінцевий точок шуканого шляху фіксоване і відносно невелике. Тепер нам достатньо розрахувати і закешувати довжини найкоротших шляхів від всіх вершин графа (де можуть розташовуватися користувачі) до місць розташування сателітів.

Уже ця оброблена інформація буде використана в реальному часі, коли за інформацією звернеться користувач.

Але розрахована один раз і збережена інформацію про оптимальні шляхи не є динамічною, що в реальних умовах роботи відобразиться на якості надання послуг. Тому такий підхід зі статично розрахованими шляхами не є добрим вирішення.

Крім такого підходу до пошуку мінімального шляху за географією можливі ще два підходи:

1. На основі даних про мережеву топологію:

- BGP-маршрути в режимі реального часу;
- База даних RIPE;
- Таблиці локальних адресів регіональних провайдерів;

2. За інтегральними метриками:

- мережева затримка;
- кількість хопів;
- AS (автономних систем) на шляху.

У результаті метод вибору шлюзу за складністю запиту можна зобразити у фигляді діаграми потоків даних (рис. 3.15).

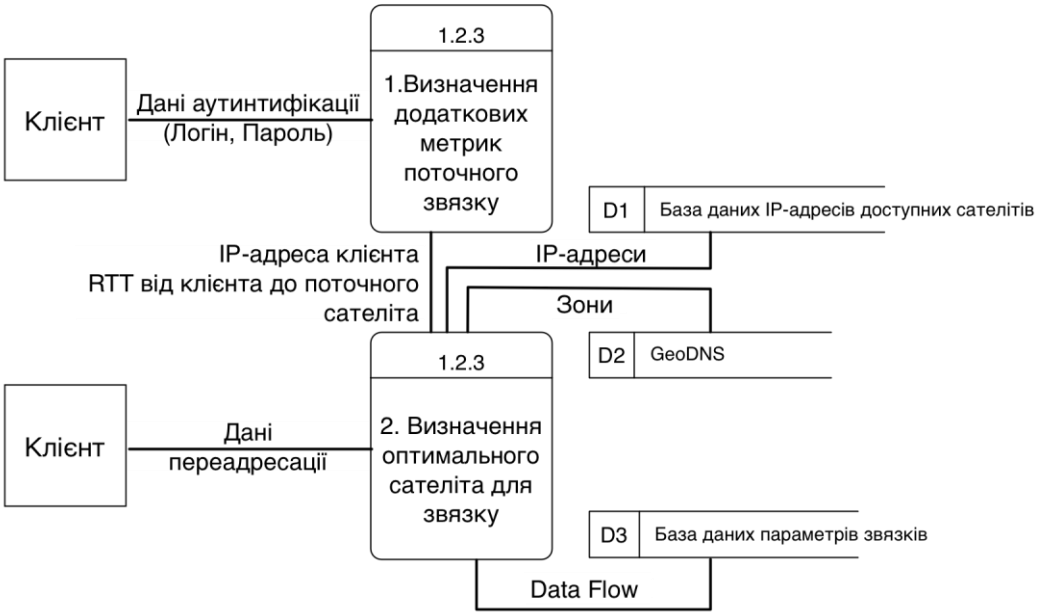


Рис. 3.15. Діаграма потоків даних методу вибору шлюзу доступу, розроблено самостійно.

3.5. Висновки до 3-го розділу.

У результаті роботи у даному розділі:

1. Розроблено метод сеансового рівня для високошвидкісних регіональних мереж.

2. Розроблено метод мультипротокольного доступу до потокових даних.

3. Розроблено метод мультиплексування даних з різних джерел для одночасної передачі через хмаркові сховища.

4. Розроблено метод вибору шлюзу доступу до хмаркового сховища даних.

Основні результати розділу опубліковані у працях [17, 19, 20].

РОЗДІЛ 4.
МОДЕЛЮВАННЯ ТА АПРОБАЦІЯ РОЗПОДІЛЕНОЇ МЕРЕЖНОЇ АРХІТЕКТУРИ ХМАРНОГО СХОВИЩА ДАНИХ

Розділ присвячений комплексній реалізації практичного використання запропонованих підходів. Зокрема у розділі подано архітектуру системи, програмну реалізацію системи обміну через хмаркове сховище даних у розподіленій мережі та приведено порівняльний аналіз ефективності запропонованих методів.

4.1. Проектування архітектури системи.

Для практичного застосування розроблених методів необхідно спроектувати та розробити розподілену систему зберігання даних.

Як було показано в попередніх розділах, специфіка сучасних сховищ даних у переході їх до розподілених систем. У таких умовах необхідною умовою є забезпечення резервування критичних вузлів мережі. Також важливою умовою є те, щоб вузли мережі володіли інформацією про наявність інших вузлів та даних на них.

В умовах розподіленої роботи, необхідним функціоналом роботи є забезпечення вчасне виявлення непрацездатного вузла обміну/збереження даних та вилучення їх з роботи. Також після відновлення роботи вузла, він має якомога швидше долучитися до обміну/збереження даних. Проектування такої системи вимагає попереднього аналізу вимог та планування її внутрішніх процесів.

Програмне забезпечення системи повинно забезпечувати виконання всіх функцій і мати засоби організації всіх необхідних процесів оброблення, передавання та зберігання даних в усіх регламентованих режимах функціонування.

Програмне забезпечення системи повинно бути:

- універсальним;
- функціонально достатнім;
- надійним;

- адаптивним;
- придатним до модернізації та масштабування;
- мати інтуїтивно зрозумілий для користувача інтерфейс;
- захищеним від зовнішніх впливів;
- здійснювати документування усіх дій користувачів програмного забезпечення.

Програмне забезпечення повинно розроблятися із застосуванням принципів структурного і модульного програмування. Кожна із задач, яка входить в систему повинна бути максимально незалежною від інших.

Контроль якості програмних засобів, які розробляються, повинен бути забезпечений тестуванням та проведенням дослідної експлуатації.

Архітектура системи повинна базуватися на засадах високо-навантажених та відмовостійких систем. Вона повинна відповідати наступним основним вимогам:

- система повинна підтримувати горизонтальне масштабування серверів для збільшення продуктивності системи в цілому (як вузлів для зберігання даних, так і сателітів для покращення ефективності доступу до даних);
- система повинна мати можливість дублювання всіх критично важливих вузлів мережі та зберігання даних, для забезпечення відмовостійкості системи.

До складу системи повинні входити наступні функціональні компоненти:

1. Сховище даних – компонент інформаційної системи, який забезпечує зберігання даних для вирішення наступних задач:
    - зберігання даних;
    - доступу до даних;
    - облік дій користувачів.
2. Сателіт – компонент інформаційної системи, який забезпечує швидку інформаційну та технічну взаємодію Системи із користувачами системи.

3. Адаптивний DNS сервер – компонент системи який забезпечує логістику запитів від користувача до системи.

Загальними вимогами до надійності системи є:

- Програмно-технічний комплекс повинен функціонувати цілодобово, у безперервному режимі, крім форс-мажорних обставин.
- Повинно проводитися регулярне (не рідше одного разу на добу) резервне копіювання баз даних. Необхідна наявність як мінімум двох резервних копій всіх даних. Резервні копії повинні зберігатися в фізично віддалених місцях.
- Відмови і збої в роботі робочих станцій і мережевих пристроїв не повинні призводити до руйнування даних і позначатися на працездатності системи в цілому.
- Вихід з ладу однієї з підсистем не повинен призводити до припинення функціонування інших підсистем, тобто при цьому повинна забезпечуватися можливість виконання функцій всіх інших підсистем.
- Планова зупинка або збій інформаційного ресурсу не повинні призводити до збою в роботі програмного забезпечення.
- Неправильні дії користувачів не повинні призводити до виникнення аварійної ситуації.
- Повинні бути мінімізовані помилки технічного персоналу, в тому числі шляхом чіткого розмежування прав доступу до системи, а також ведення журналу подій системи.

Під надійністю інформаційної системи розуміють комплексну властивість системи зберігати в часі основні властивості системи визначені у встановлених нормативно-технічних документах. При такому розумінні програмне забезпечення повинне:

- бути стійким до хибних дій користувача (помилки у діях персоналу не повинні приводити до збоїв (відмов) у роботі програмного забезпечення інформаційної системи);
- забезпечувати гарантований контроль вхідної та вихідної інформації;

- забезпечувати швидке відновлення після відмови (збоїв).

Програмне забезпечення розробляється на основі розповсюджених операційних систем, інструментальних засобів програмування і СУБД (система управління базами даних).

Система повинна використовувати стандартні рішення, що базуються на застосуванні типових протоколів та інтерфейсів взаємодії, що передбачають можливість сполучення та спільної роботи обладнання та програмного забезпечення різних виробників, а також для сполучення з інформаційними системами інших організацій.

Усі технічні рішення, використовувані в проекті системи, повинні відповідати вимогам національних стандартів або (за відсутності) міжнародних стандартів. Технічні засоби, що застосовуються в складі ІС, повинні мати сертифікати або інші документи підприємства-постачальника, що підтверджують їх відповідність технічним умовам.

У силу великої соціальної значущості проекту, та стислих термінів введення в експлуатацію перевагу слід віддавати уніфікованим рішенням. Такі рішення повинні мати наступні властивості:

- доступ до Системи повинен надаватися за допомогою глобальної мережі Internet;
- модульність (компонентний рішення);
- Мати можливість інтеграції з зовнішніми автоматизованими системами.

Система повинна забезпечувати виконання наступних функцій:

- реєстрація клієнтів системи:
    - створення обікового запису компанії в системі;
    - створення облікового запису адміністратора для компанії;
- адміністративна частина:
    - реєстрація користувачів системи для компанії;
    - редагування профілю користувача;
- забезпечення доступу до даних по протоколах:

- HTTP;
- HTTPS;
- FTP;
- FTPS;
- SFTP;

- оброблення помилок:
  - помилки сховища;
  - помилки сателітів;
  - помилки сценарію;
  - обриви зв’язку;
  - велике навантаження;
- реєстрація дій користувачів.

Проаналізувавши перелік необхідних функцій і взаємозв'язки між ними, доцільно виконати їх представлення у вигляді варантів використання системи (рис. 4.1, 4.2).

Сиситема повинна забезпечувати ефективну організацію обміну інформацією між внутрішніми компонентами. Інформаційний обмін між компонентами системи має здійснюватися з використанням локальних обчислювальних мереж і глобальних мереж передачі даних. Склад, структура, обсяг і частота передачі повідомлень повинні визначатися відповідними протоколами інформаційного обміну, певними на стадії технічного проектування. У протоколах інформаційного обміну повинні бути передбачені заходи по виключенню можливості несанкціонованого доступу до даних.

Також повинні бути передбачені засоби контролю переданих вхідних/вихідних даних та засоби по контролю інформації в базах даних.

Вимоги до інформаційного обміну між компонентами системи повинні бути визначені на етапі розробки, виходячи з можливостей платформи реалізації.

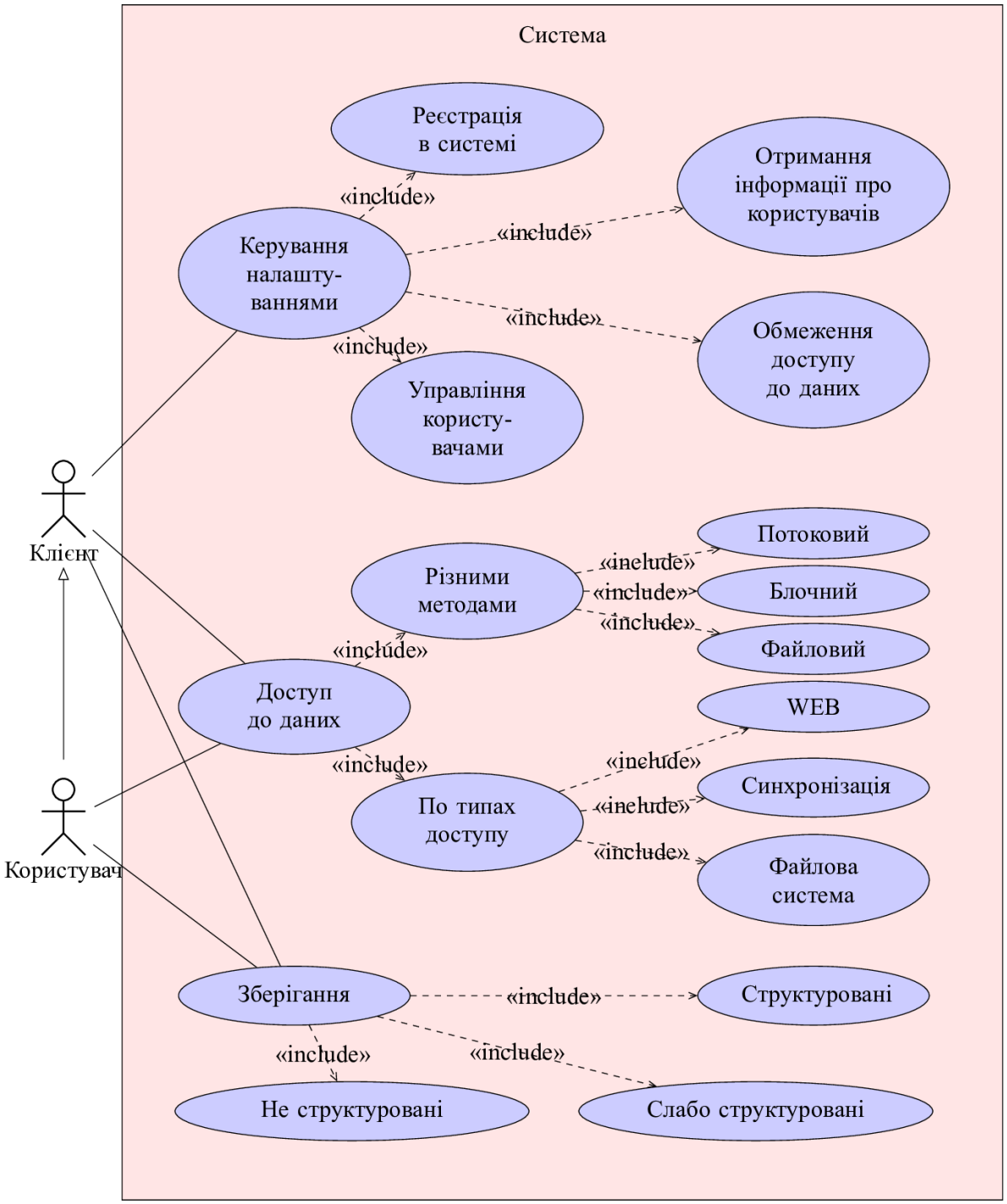


Рис. 4.1. Варіанти використання системи з точки зору Користувача та Клієнта системи, розроблено самостійно.

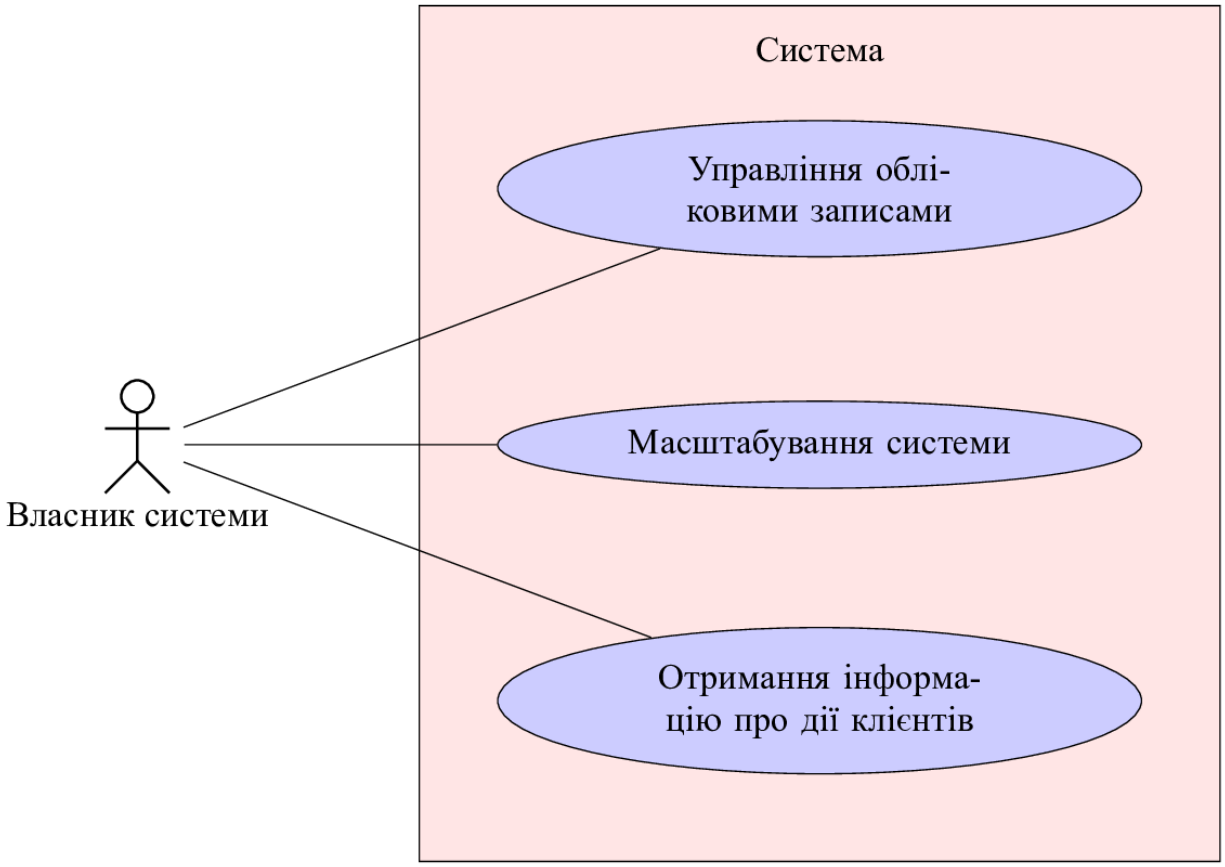


Рис. 4.2. Варіанти використання системи з точки зору Власника системи, розроблено самостіно.

Загальна архітектура системи складається із адаптивного DNS сервера, декількох сховищ даних (DC) та багатьох сателітів (рис. 4.3).

Кожене сховище даних повинно сладатися із (рис. 4.4):

- двох брандмауерів;
- двох серверів додатків;
- двох сховищ з дисками;
- така архітектура необхідна для забезпечення відмовостійкості системи у випадку втрати фізичного вузла мережі. На першому етапі запити приходять на рівень брандмауерів, які для зовнішніх вузлів відображаються з одим і тим же IP-адресом з індексами 0 та 1. У випадку виходу зладу пристрою з індексом 0, всі запити починають попадати на пристрій з індексом 1.

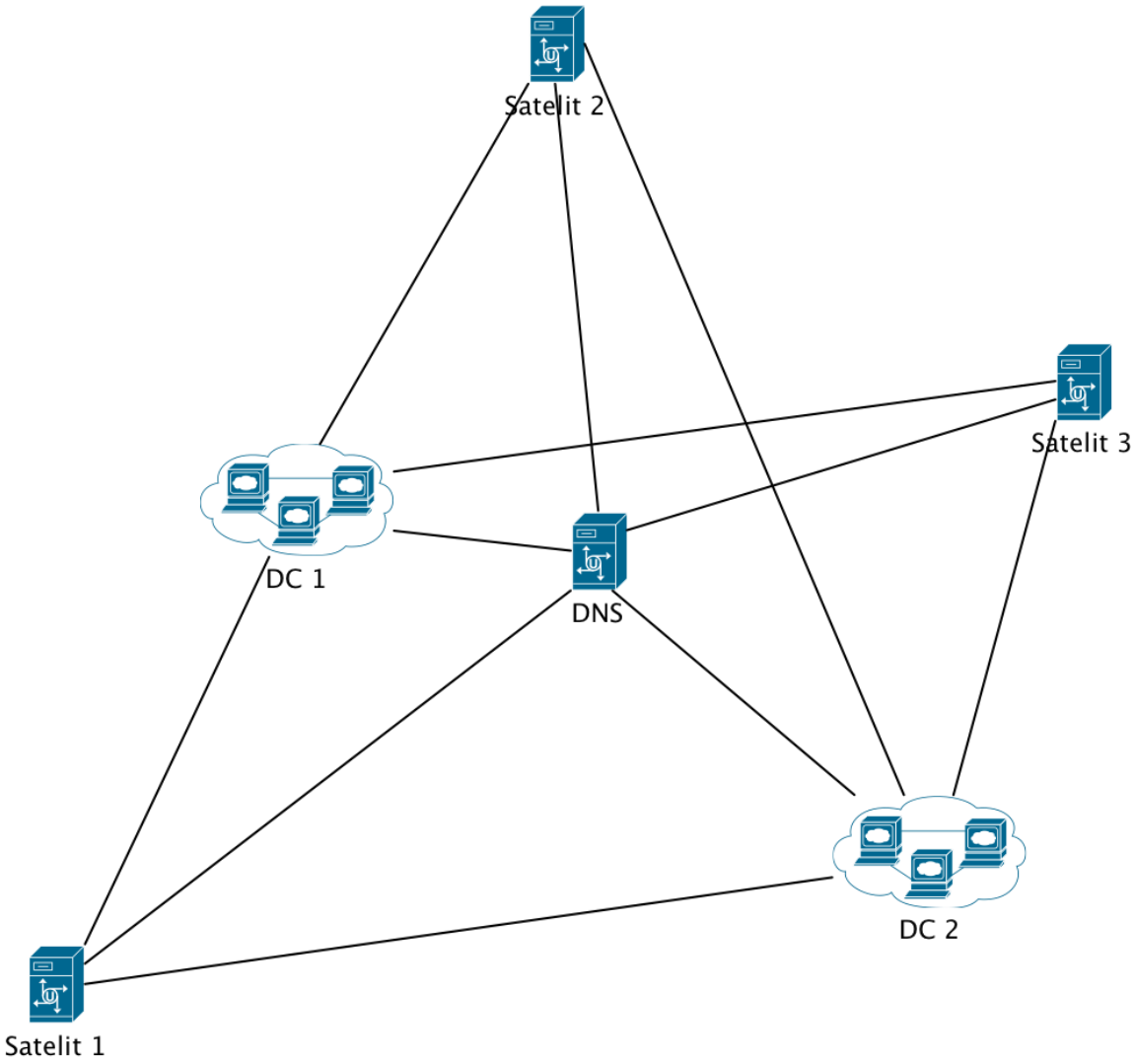


Рис. 4.3. Загальна архітектура системи, розроблена самостійно.

Після отримання брандмауером запиту від користува, запит попадає почергово на доступний сервер додатків, статус доступності якого постійно прослідковується на першому рівні.

Два сховища після апаратного підключення, на апаратному рівні підключаються як єдине сховище. Також на програмному рівні налаштована синхронізація даних між сховищами, для забезпечення резервування даних.

Оскільки, надзвичайно швидко збільшується швидкодія процесорів та об’єм оперативної пам’яті сервер, який програми без математично складких алгоритмів не можуть використати повністю та впираються у кількість користувачів, доцільним є використання контейнерів (віртуалізація на рівні операційної системи). Тоді, структурно сервер додатку буде складатися із декількох контейнерів та балансувальника навантаження (рис. 4.5).

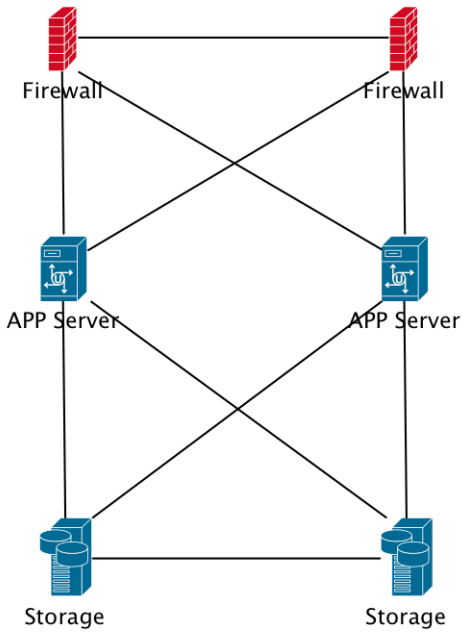


Рис. 4.4. Загальна архітектура одиничного сховища, розроблено самостійно.

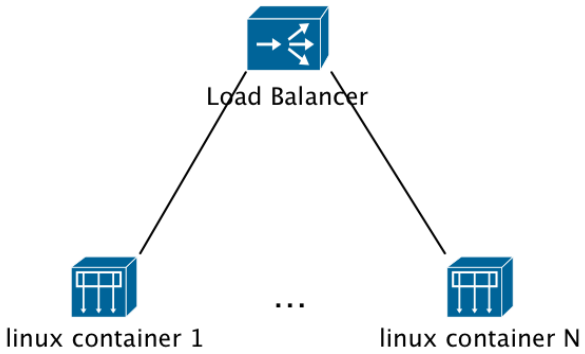


Рис. 4.5. Загальна структура сервера додатку, розроблено самостійно.

Для моніторингу доступності сервісів у контейнерах існує декілька програмних засобів, найпопулярнішим з яких є keepalived. Їхнім недоліком при використанні у нашої архітектурі є відсутність групування сервісів. Оскільки однією з ключових вимог є можливість доступу до даних по FTP протоколу різних користувачів різних компаній (кожна компанія має свій hostname), нам потрібно для кожної компанії мати свій IP-адрес доступу (специфіка використання FTP). FTP-server має можливість підтримувати одночасне використання багатьох IP-адрес. У такому випадку, keepalived на кожен IP-адрес буде пробувати створювати з’єднання та перевіряти доступність, а якщо їх буде достатня кількість, то нагрузки на сервер для перевірки доступності контейнерів будуть збільшуватися в геометричній прогресії.

Для вирішення даної проблеми, спроектовано систему, яка повністю заміняє функції keepalived, та має можливості:

- групувати IP-адреси та сервіси як один сервіс (у випадку недоступності, даний сервіс перестає бути доступний по всім IP-адресам які зазначені. Та навпаки, у випадку повторної доступності сервісу, сервіси по всім IP-адресам стають знову доступні);
- Зберігання статистики за всіма IP-адресами та сервісам з урахуванням:
  - кількості запитів;
  - переданих даних;
  - отриманих даних;
  - втрати працездатності сервісу;
  - відновлення працездатності сервісу.

Архітектурно Сателіт відповідає архітектурі сервера додатку з сховища даних. Йому не обхідно великого і надійного сховища даних, оскільки він виступає виключно «проксі-сервером» між клієнтом та його даними. Також у нас немає необхідності його повністю дублювати, оскільки він не несе у собі критичних даних і у випадку виходу зладу, автоматично запити які йшли на нього будуть йти на ближні з ним інші сателіти.

4.2. Дослідження та аналіз потоків даних в системі.

У роботі сучасного хмаркового сховища даних, яке використовує інформаційні системи, повинні бути забезпечені якомога раціональніша організація інформаційний потоків, так і суттєве підвищення їхньої інтенсивності, тобто прискорення передачі й обробки інформації, що надходить від її джерела до споживача. Для рішення цих завдань при проектуванні інформаційної системи, насамперед, проводиться аналіз інформаційних потоків, до дозволяє:

- розглянути усі ланки системи обробки та збереження даних, починаючи з одержання вихідних відомостей, поступове перетворення та формування кінцевих даних, які направляються керованій системі як

команди у якості звітної й іншої інформації. При цьому визначається роль кожного елемента система у хмарковому сховищі даних, що виконуються системою і зафіксованих у схемі обробки даних, уточнюється їхня структура та функції;

- побудувати схему інформаційних зв'язків всіх елементів системи між собою та зовнішнім середовищем. У схемі можуть залишатися відомості про конкретні форми інформаційних зв'язків і вказуються їх кількісні та часові характеристики;
- виявити первинні (вихідні) для системи дані. Аналіз потоків інформації – найважливіший етап у раціоналізації існуючої системи сховища даних, що повинен забезпечити виконання цільових завдань проектування.

Вивчення потоків інформації надає загальне представлення про функціонування системи і є першим кроком в аналізі ефективного проектування високонавантаженої системи обробки, зберігання та передачі даних. Подальше дослідження інформаційних потоків дозволяє виявити елементи інформаційного відображення об'єктів, відносини між ними, структуру та динаміку потоків інформації.

Рух даних в системі, що супроводжується відповідними інформаційними потоками, є основою для забезпечення роботи хмаркового сховища даних. Між всіма елементами системи, що функціонують в середовищі сховища даних, відбувається неперервний рух інформаційних потоків, які забезпечують надходження інформаційних даних, необхідних для здійснення аланалізу та передачі клієнтських даних.

Основними елементами системи є (рис. 4.6):

- Сховище даних – один з головних селементів системи, на якому візично і довгостроково зберігаються клієнтські дані (рис. 4.7);
- Сателіт доступу до даних – елемент доступу до даних в системі, який знаходиться «як найближче» до користувачів (споживачів даних), та має можливість тимчасового зберігання даних (кешування) (рис. 4.8);

- DNS-сервер – елемент системи, що приймає динамічні оновлення від інших елементів системи, та надає достовірну та актуальну інформацію користувачам даних.

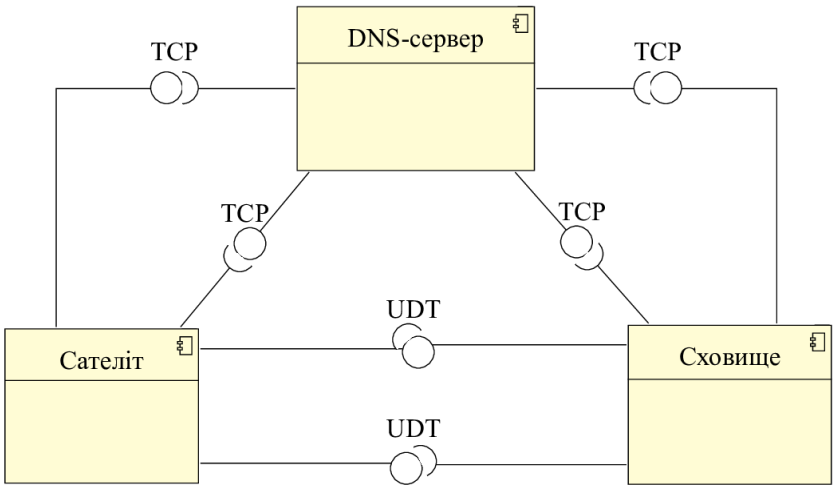


Рис. 4.6. Діаграма взаємодії основних елементів системи, розроблено самостійно.

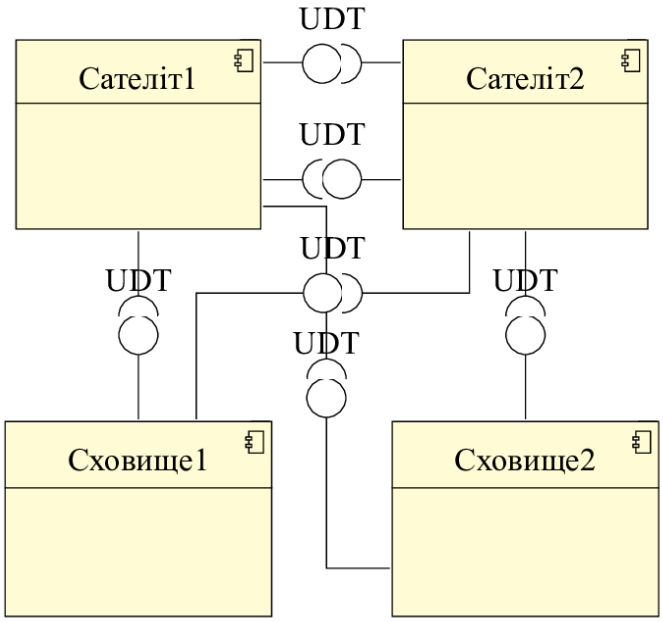


Рис. 4.7. Діаграма розміщення сателіту, розроблено самостійно.

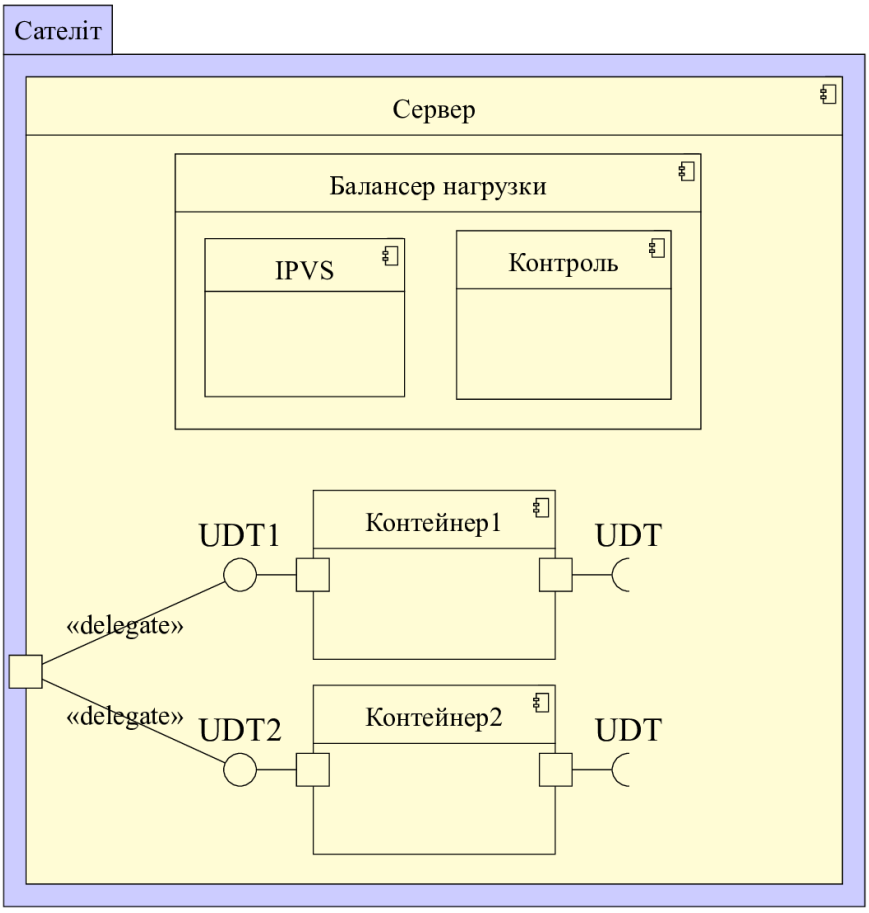


Рис. 4.8. Діаграма розміщення Сховища, розроблено самостійно.

Для актуалізації інформації про місцезнаходження доступних елементів системи та їх доступності, використовується DNS-сервер. Він виступає ключовим елементом інформаційної взаємодії інших елементів системи. Для цього кожен елемент системи з певною періодичністю надсилає інформацію про себе та авторизуються на DNS-сервері використовуючи TCP з'єднання (рис. 4.6).

У свою чергу, інші елементи системи обмінюються інформаційними даними за допомогою протоколу даних, який базуться на UDT (рис. 4.9).

Сховища даних, як елемент системи управління даними активно споживає інформаційні ресурси та, відповідно, є отримувачем і відправником інформаційних потоків.

Сателіт, як елемент споживання даними також активно споживає інформаційні ресурси в двосторонньому напрямку.

Для детальнішого аналізу потоків даних, необхідно розглянути кожний елемент системи в розрізі даних та підсистем даного елементу.

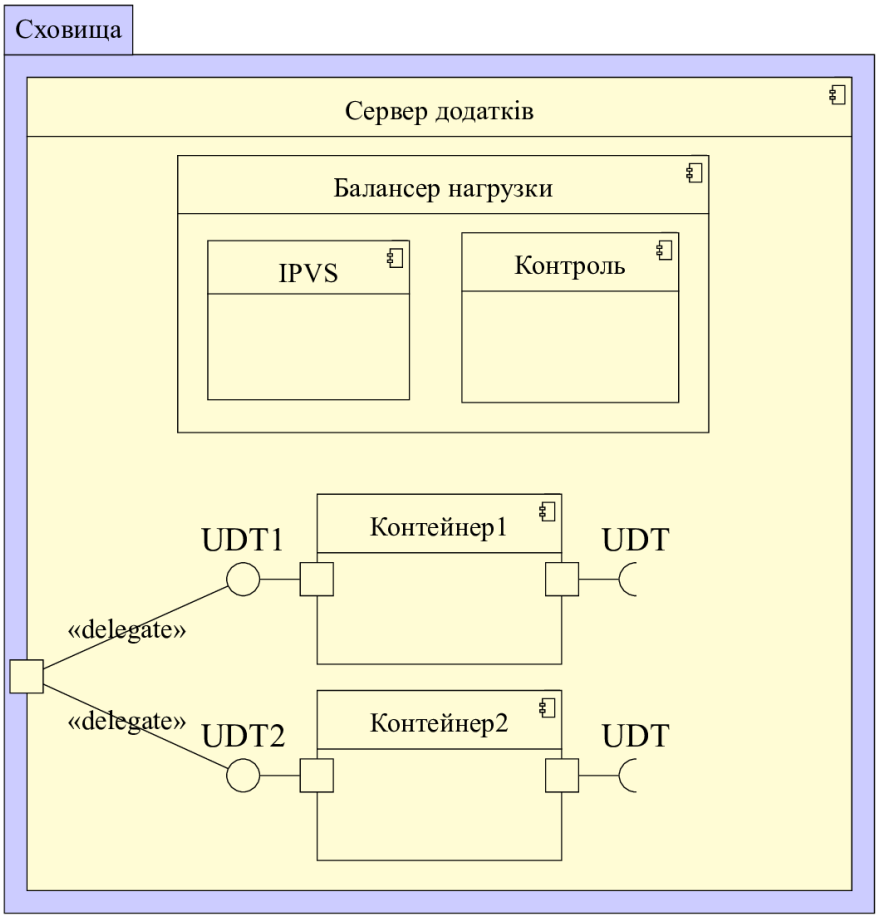


Рис. 4.9. Діаграма взаємодії сховищ та сателітів, розроблено самостійно.

Для максимального використання ресурсів системи, кожний фізичний сервер поділяться на максимально незалежні блоки (контейнери). Контейнери – це система віртуалізації на рівні операційної системи для запуску декількох ізольованих примірників ОС Linux на одному комп'ютері. Вони не використовує віртуальні машини, а створює віртуальне оточення з власним простором процесів і мережевим стеком. Усі примірники контейнерів використовують один примірник ядра ОС, для максимальної ефективності використання їх.

Для забезпечення роботи системи із декількох контейнерів, як однієї системи використано два типи (рис. 4.7, 4.8):

- Контейнер «Балансер навантаження»;
- Контейнер програмного додатку.

Балансер навантаження виконує роль проксі-сервера, який перевіряє доступність контейнерів та переадреадресовує отримані запити до доступного контейнера для обробки їх.

Контейнер програмного додатку – це повноцінна копія програмного додатку який обробляє запит користувача та повертає йому результат роботи.

Контейнер сховища даних (рис. 4.10) отримує та виконує запити виключно по протоколу UDT для чого в контейнері запущені незалежні процеси UDT-сервер та UDT-клієнт. За допомогою даного протоколу контейнер отримує дані з інших сховищ дани або від клієнтів через сателіти, та надсилає дані іншим сховищам даних та клієнтам з використанням сателітів доступу до даних.

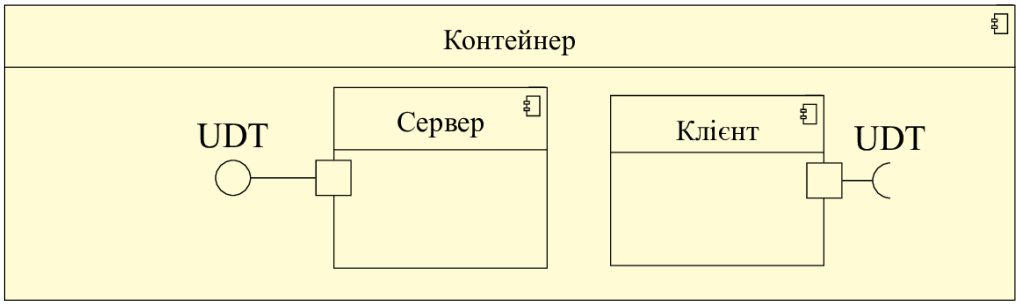


Рис. 4.10. Діаграма розміщення Контейнера сховища, розроблено самостійно.

Контейнер програмного додатку сателіту (рис. 4.11) містить в собі:

- UDT-сервер та UDT-клієнт -- для обміну даними з іншими елементами глобальної системи сховища даних
- Web-сервер для простого доступу до даних для користувачів
- Proftpd-сервер -- призначений для розширення функціоналу доступу до даних за допомогою протоколів: FTP, FTPS, SFTP.

Розглянувши функціонал системи як окремі бізнес-процеси системи, ми отримаємо:

1. Пошук найближчого сателіта доступу до даних
2. Завантаження даних на сховище
3. Доступ до даних
4. Реплікація даних

Пошук найближчого сателіта доступу до даних зображено на рис. 4.12.

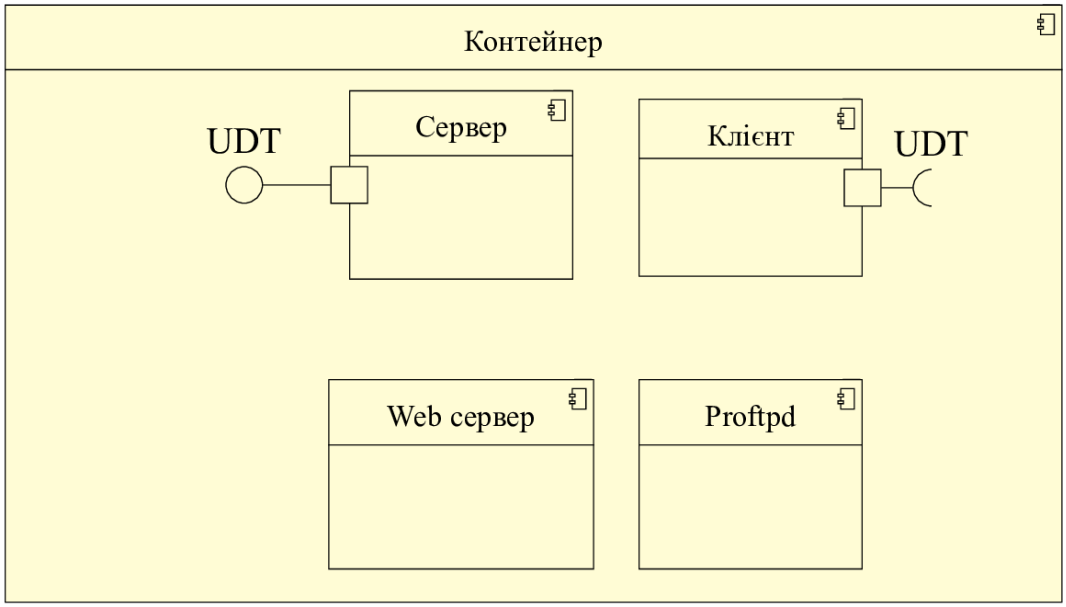


Рис. 4.11. Діаграма розміщення Контейнера сателіту, розроблено самостійно.

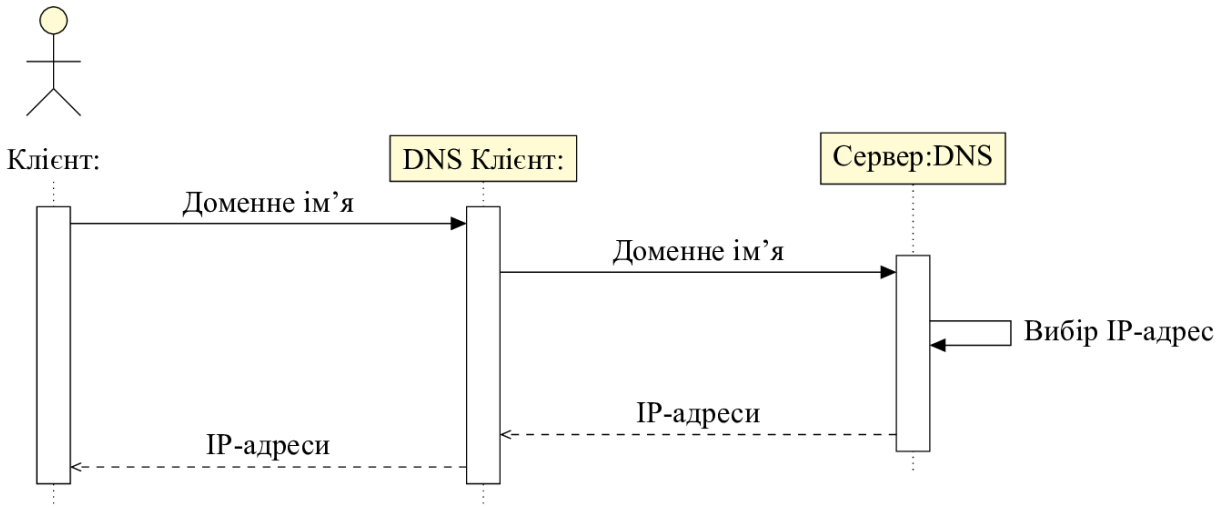


Рис. 4.12. Діаграма послідовності пошуку найближчого сателіта, розроблено самостійно.

У випадку коли клінту необхідно отримати доступ до даних, програмний додаткок звертається до DNS-клієнта операційної системи із запитом отримати IP-адрес системи використовуючи доменне ім'я. У свою чергу DNS-клієнт використовуючи мережу DNS-серверів звертається до NS-сервера системи і він використовуючи IP-адрес клієнта обчислює оптимальний сателіт доступу до даних, який «найближчий» (мінімальна кількість хопів та максимальна швидкість) до нього.

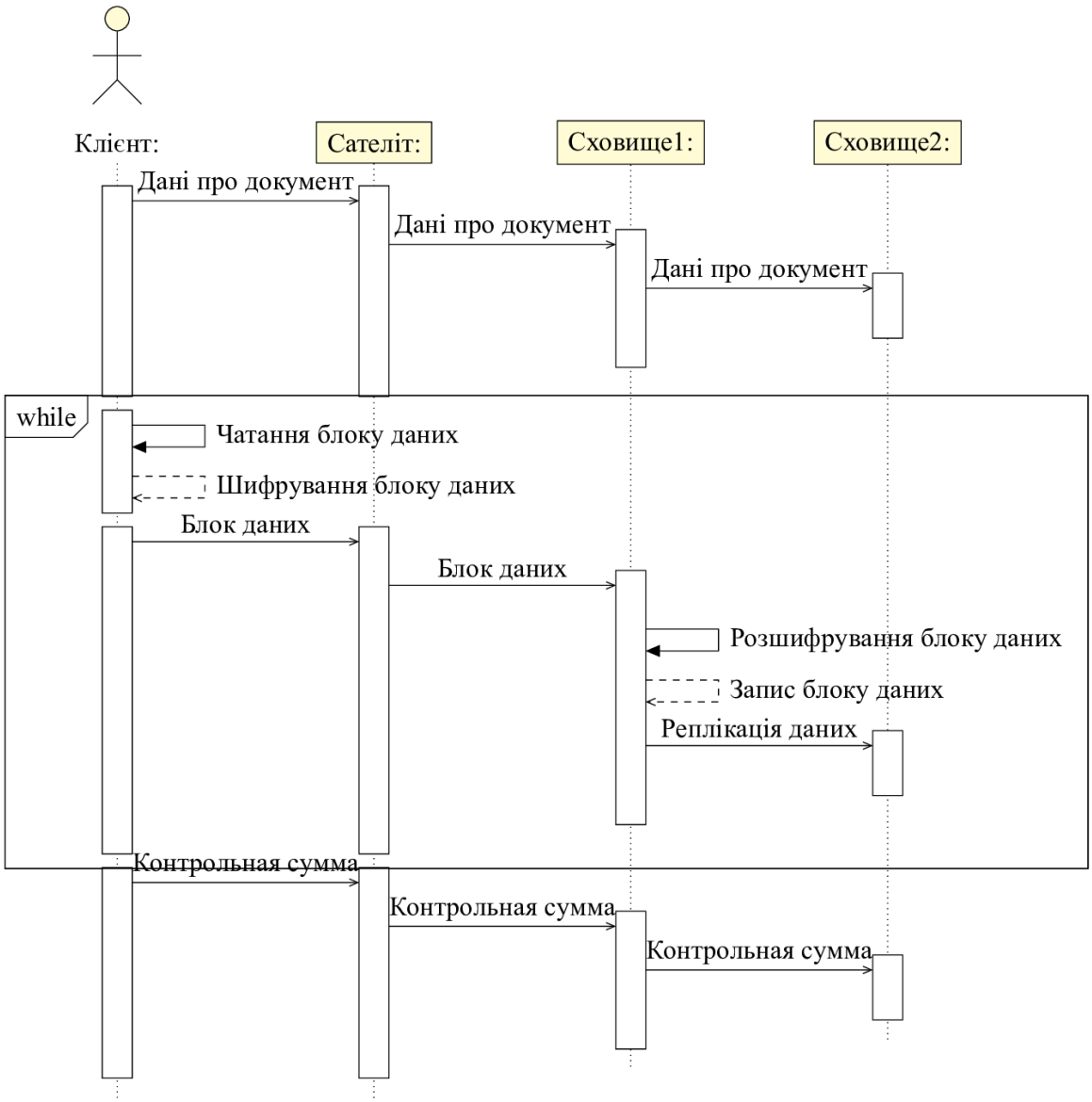


Рис. 4.13. Діаграма послідовності завантаження даних, розроблено самостійно.

Завантаження даних на сховище відбувається згідно рис. 4.13. Отримавши IP адрес найближчого сателіту, клієнта авторизується на сервісі і тільки після того, може отримати доступ до даних. Коли клієнту необхідно завантажити документ, він за допомогою програмного додатку надсилає документ на сателіт з використання протоколу TCP. Сателіт в свою чергу кешує дані та надсилає дані на сховище отримання напряму, або через найближче сховище з використанням протоколу UDT.

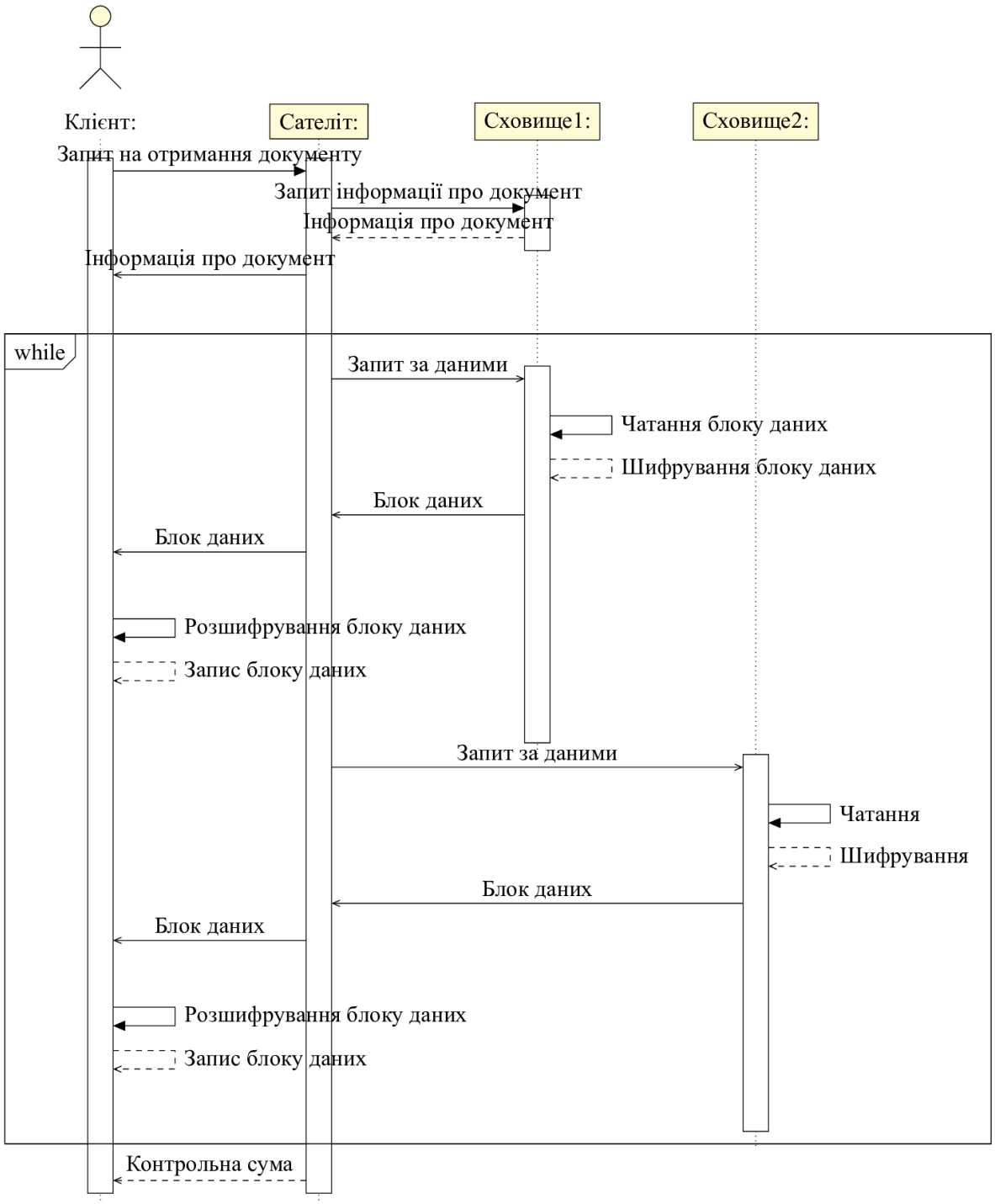


Рис. 4.14. Діаграма послідовності доступу до даних, розроблено самостійно.

Доступ до даних демонструє рис. 4.14. У випадку необхідності отримання документу, клієнт за допомогою програмного додатку надсилає запит на отримання документу до сателіту з використання протоколу TCP. Сателіт в свою чергу отримує документ зі сховища напряму або через інше сховище з використанням протоколу UDT. Документ тимчасово кешується на сателіті на надсилається клієнту (TCP).

4.3. Практична реалізація протоколів обміну даними в розподіленому сховищі.

В концепції глобальної мережі існують два підходи до реалізації обміну даними:

- послідовний передача даних – передача даних послідовно по одному каналу зв'язку;
- паралельна передача даних – передача даних паралельно по декількох каналах зв'язку.

Перевагою паралельної передачі даних є швидкість передачі даних. Даний метод використовується в комп'ютерній периферії для обміну даними в шинах даних. Основним недоліком даного підходу є залежність від якості та провідності провідників, які використовуються в даній передачі. При різних властивостях провідників біти в передачі даних можуть приходити із затримкою, що призводить до значних помилок.

У свою чергу послідовна передача даних є менш залежною від провідників, оскільки передача даних йде виключно по одному каналу зв'язку, але, у свою чергу, є значно повільнішим, нім при паралельній передачі даних. Даний підхід є найбільш розповсюдженим в глобальним мережам.

Методи передачі даних також класифікуються на:

- синхронна передачу даних;
- асинхронна передачу даних.

Асинхронний обмін є найбільш розповсюдженою формою послідовного зв'язку, що передбачає передача пакету данив в якому міститься інформація про початок і кінець передачі даних, інформація для контролю помилок та самі дані.

Оскільки архітектура хмаркового сховища даних передбачає як одночасну передачу даних в різних напрямках, так і прийом даних з різних джерел даних, було вибрано асинхронну послідовну передачу даних. Також для ефективного розподілу ресурсів було вибрано асинхронний підхід до проектування системи.

Модуль системи передачі даних можна представити як систему станів та переходів (рис. 4.15). При запуску процесу створюється канал передачі даних який переходить у статус «Очікування на події». Оскільки даний канал зв'язку передбачає, як передачу даних, так і прийом даних, даний канал зв'язку реарує на наступні події:

- Вхідне з'єднання – під час даної події системі необхідно аутентифікувати клієнта та добавити дане з'єднання до пулу з'єднань.
- Вхідні дані – отримати дані, розшифрувати, та виконати дії передбачені в даному пакеті даних. У випадку відсутності даних в каналу зв'язку для читання, та недоотримання пакета даних, даний пакет поміщається у чергу недоотриманих пакетів даних, та система очікує на можливість отримання даних по даному каналу зв'язку.
- Необхідність у передачі даних – перевірка на існування з'єднання з віддаленим клієнтом, формування пакетів даних, шифрування їх, та передача їх по каналу зв'язку. У випадку неможливості негайно передати дані, пакети даних переміщають у чергу пакетів та очікують на можливість передачі (звільнення вихідного каналу зв'язку).
- Помилка з'єднання – в даному випадку система оцінює дану помилку, та приймає рішення. Якщо система отримала помилковий пакет, або несподіваний для неї, система «просить» клієнта повторно передати дані. Якщо стався більшей збій системи, як збій синхронізації шифрування даних, система розриває з'єднання з даним клієнтом, після чого клієнт перестворює з'єднання та повторно відправляє дані на які не отримав підтвердження від сервера.

Оскільки клієнтські дані зберігають у хмарковому сховищі даних у вигляді файлової структури, передача даних в системі зводиться до передачі файлів. Передачу окремого файлу можна представити, як передачу окремих пакетів даних (рис. 4.16).

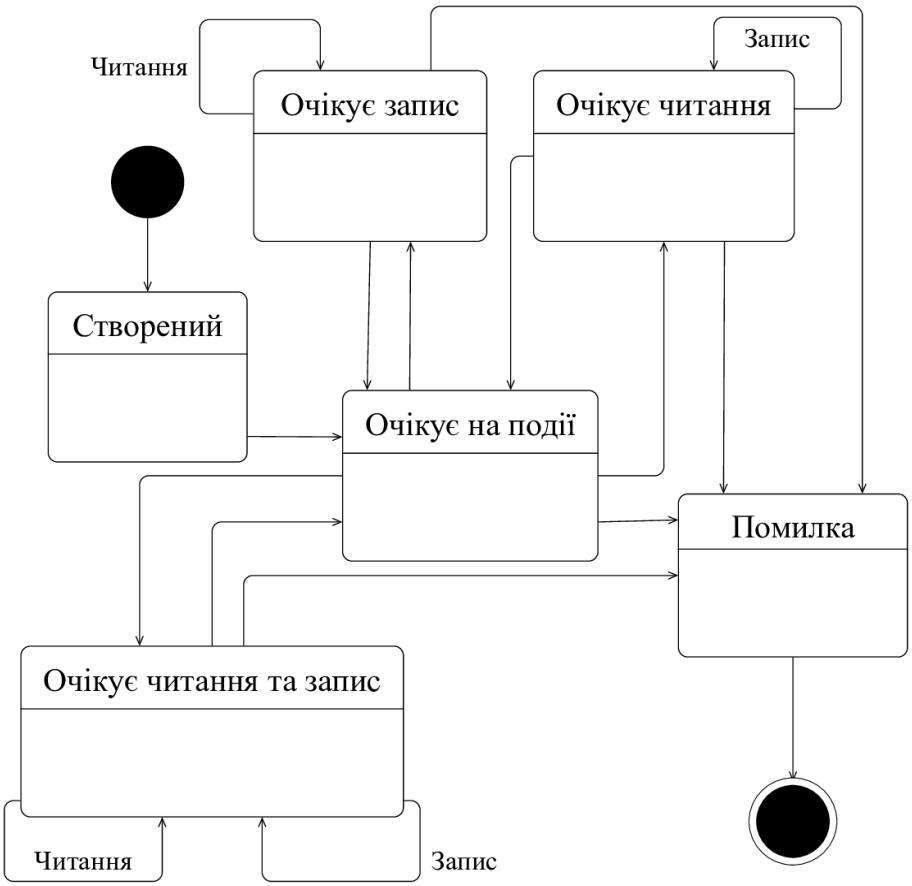


Рис. 4.15. Діаграма станів каналу передачі даних, розроблено самостійно.

Під час передачі будь-якої кількості даних система формує пакети передачі даних та добавляє їх у чергу передачі даних. Після чого система виконує ряд функції:

1. Перевірка на існування з'єднання з пунктом призначення даних.
2. Якщо з'єднання відсутнє:
   - створення з'єднання з пунктом призначення;
   - аутентифікація на пункті призначення;

3. Почергове шифрування па передача пакетів даних (з черги) по даному каналу зв'яку.

4. У випадку заповнення вихідного кешу, система переходить до очікування інших подій.

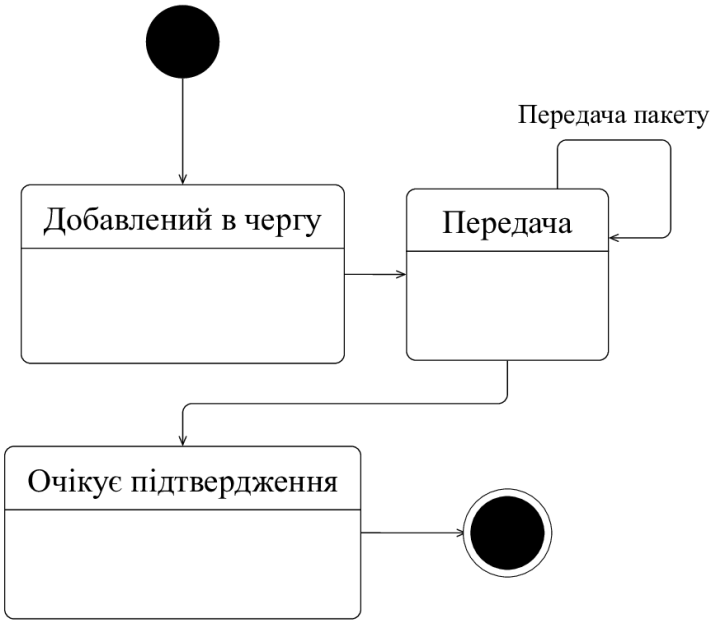


Рис. 4.16. Діаграма станів файлу для передачі, розроблено самостійно.

Після вдалого відправлення пакету даних, система помічає в черзі передачі даних пакет, як «відправлений». Після вдалого отримання клієнтом пакету даних, він відправляє пакет підтвердження. Серверна частина отримавши даний пакет видаляє пакет з черги передачі даних.

У випадку розірвання з'єднання з отримувачем даних, система автоматично змінює статус відправлених пакетів даних, на нові, для повторної передачі даних (рис. 4.17).

Логічну модель системи передачі даних в хмарковому сховищі даних можна представити у вигляді діаграми класів (рис. 4.18).

З діаграми класів очевидно, що концептуальна модель системи передачі даних в Сховищі даних та Сателіті аналогічні. Даний підхід спрощує побудову систем такого характеру. Також з даної структури очевидно, що «З'єднання» є абстрактним класом та може використовуватися протоколами як TCP, або UDT, для траспортування даних. Такий підхід дає суттєві переваги у випадках існування єдиного обмеженого каналу зв'язку, що є доволі актуальним для сучасного стану зв'язку в Україні.

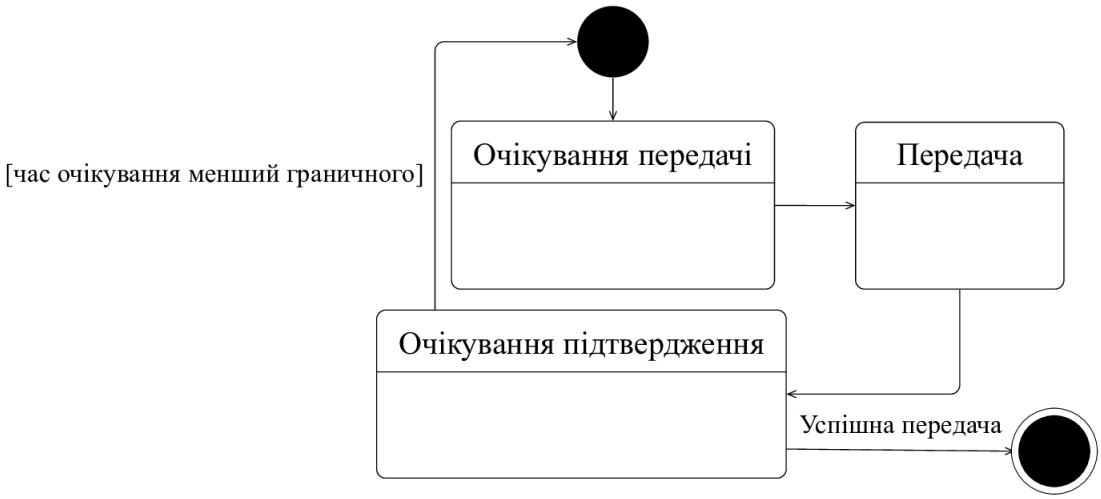


Рис. 4.17. Діаграма станів пакету передачі даних, розроблено самостійно.

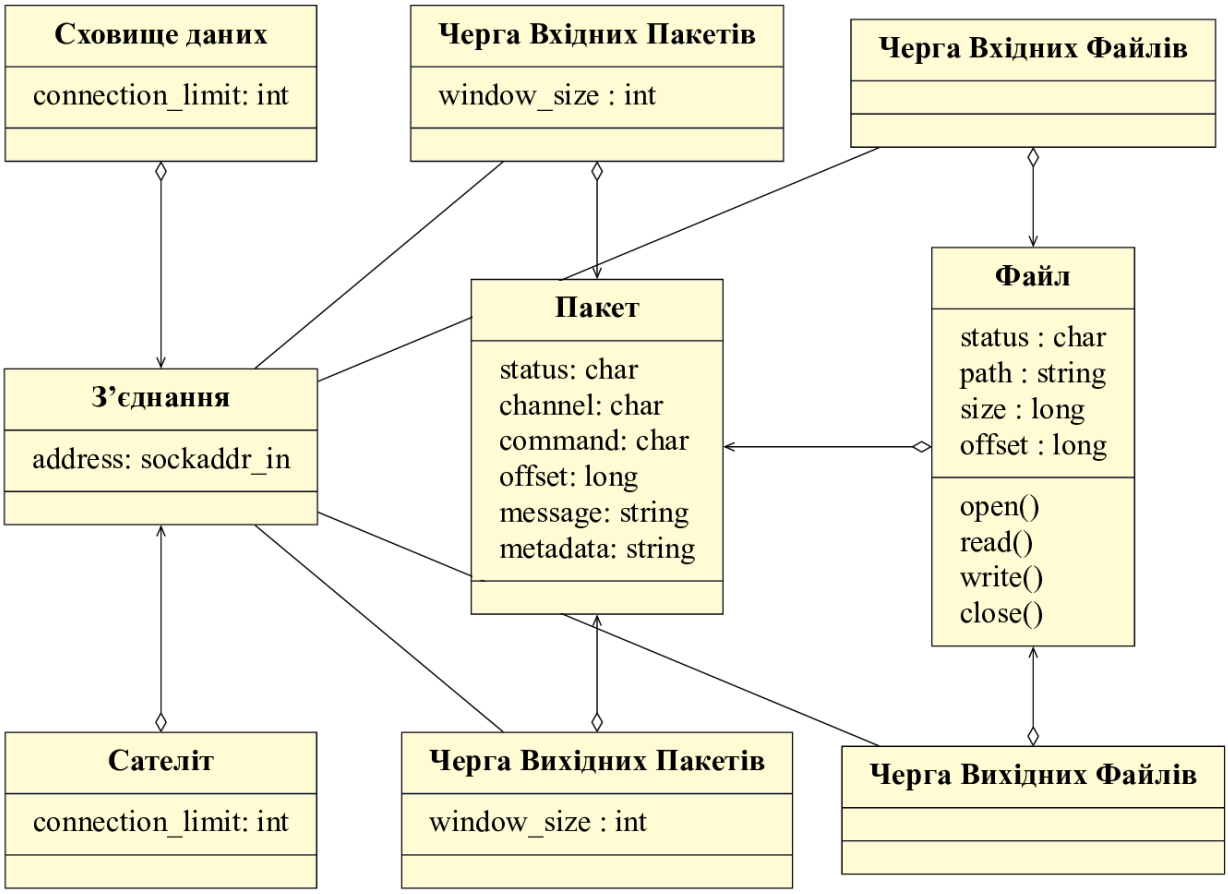


Рис. 4.18. Діаграма класів системи, розроблено самостійно.

При необхідності передачі даних (файлу) система виконує ряд простих дій:

1. Створення екземпляру класу «Файл».
2. Додавання його до «Черги Вихідних Файлів».
3. Отримання посилання на з'єднання (у випадку, якщо відсутнє – створити).

4. Створення екземплярів класу «Пакет» в межах вільного розміру вікна для з'єднання.

5. У випадку готовності з'єднання для переді – отримання «Пакету» з «Черги Вихідних Пакетів», шифрування його та передача по каналу зв'язку.

6. Отримання підтвердження закінчення файлу.

7. Видалення екземпляру класу «Файл».

З іншого боку, отримувач даних (файлу), виконує наступні дії:

1. Отримання пакету даних з даними про дані (файл), та розшифрування його

2. Створення екземпляру «Файл».

3. Додавання його до «Черги Вхідний Файлів».

4. Створення фізичного файлу.

5. Отримання пакетів з даними, розшифрування їх та наповнення ними фізичний файл, надсилання підтвердження отримання пакетів даних.

6. Отримання пакету завершення файлу.

7. Закриття файлу.

8. Видалення екземпляру класу «Файл».

9. Надсилання пітвердження отримання повного файлу.

Процес отримання пакету даних можна розбити на два етапи:

- отримання заголовку пакету;
- отримання основного тіла пакету.

В заголовку повідомлення міститься інформація про:

- Канал зв'язку – для мультиплексної передачі даних декількох файлів в один момент часу;
- Розмір тіла пакету даних;
- Команда – тип даних які передаються в тілі пакету (створення каталогу, створення файлу, дані з файлу, підтвердження отримання пакету...);
- також може міститися додаткова інформація в залежності від команди.

Використання даного підходу дало змогу збільшити продуктивність як елемента передачі даних так і системи в цілому. Використання багатоканального зв'язку дозволило одночасної передачі різних даних по одному каналу зв'язку, а універсалізація архітектури дало змогу використання різних протоколів обміну даних на різних ділянках мережі, в залежності від ефективності.

4.4. Порівняння аналіз ефективності роботи системи.

У зв'язку із швидким розвитоком індустрій послуг базованих на мережі Internet, виникає потреба у швидких протоколах передачі даних. Альтернативним напрямком розвитко таких протоколій є надбудови над протоколом UDP для забезпечення гарантованої доставки даних. Використання UDP-базованих протоколів у порівнянні з TCP є більш ефективним на великих відстанях, тобто при великих затримках [96].

Найпопулярнішими представниками таких надбудов над протоколом UDP є UDT [59] та *µTP* [111]. Проте використання цих протоколів також викликають труднощі при їх налаштуваннях до реального стану мережі. Однією з найбільш актуальних проблем дослідження ефективності протоколів передачі даних є правильно вибрана конфігурація протоколів (вхідних даних) при передачі даних на великі і дуже великі відстані. Метою роботи є розгляд двох найпопулярніших протоколів, дослідження ефективності їх та аналіз найбільш перспективного протоколу та оптимальних його параметрів.

При передачі даних на незначних відстанях з малими часами затримки, в основному, використовувався лише протокол TCP, який показав себе, як недостатньо ефективний для великих відстаней передачі даних.

Для використання модифікованих протоколів передачі даних на великих відстанях потрібно дослідити вплив параметрів цих протоколів на сумарну швидкість передачі. На прикладі реального хмаркового сховища даних протестовано протоколи передачі даних при різних буферах цих протоколів та отримано показники швидкості передачі даних на часових

зрізах, що дає змогу проаналізувати перепади швидкості передачі даних та загальну швидкість. Ці параметри дозволяють визначити ефективний протокол та вхідні параметри цього протоколу для найбільш ефективної передачі даних [17].

Для порівняння ефективності протоколів необхідно дослідити спільні конфігураційні параметри цих протоколів.

UDT протокол має наступні конфігураційні параметри:

- MSS – максимальний розмір пакету UDP;
- SNDSYN – режим синхронізації даних при відправці;
- RCVSYN – режим синхронізації даних при отриманні;
- CC – алгоритм користувача управління перевантаженням;
- FC – максимальний розмір вікна;
- UDT_SNDBUF – розмір буфера UDT-каналу при відправці;
- UDT_RCVBUF – розмір буфера UDT-каналу при отриманні;
- UDP_SNDBUF – розмір буфера UDP-каналу при відправці;
- UDP_RCVBUF – розмір буфера UDP-каналу при отриманні;
- LINGER – час затримки при закритті каналу;
- RENDEZVOUS – включення режиму «рандеву» для обходу брандмауера;
- SNDTIMEO – таймаут відправки повідомлення;
- RCVTIMEO – таймаут отримання повідомлення;
- REUSEADDR – повторне використання адреси;
- MAXBW – максимальна пропускна здатність каналу (швидкість передачі даних).

Натомість протокол μTP має дещо менший список конфігураційних параметрів протоколу:

- SO_SNDBUF – розмір буфера μTP -каналу при відправці;
- SO_RCVBUF – розмір буфера μTP -каналу при отриманні;
- SO_UTPVERSION – версія протоколу (0 або 1), що встановлює визначені розробником розміри буферів при відправці та отриманні в каналі μTP .

Для чистоти дослідження, в роботі змінювалися лише спільні (аналогічні) для двох протоколів параметри, – це величина буферу при відправці та отриманні даних. Для нехтування похибками часу передачі даних було прийняте ріщення використовувати параметри для буферів кратні 512 байт (розміру буферу читання/запису файлової системи на серверах).

Метою дослідження було визначення ефективності протоколів передачі даних при різних розмірах буферів у міжконтинентальній мережі хмаркового сховища даних.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- дослідити швидкість передачі даних при різних величинах буфера;
- проаналізувати зміни швидкості передачі даних під час передачі одного блоку даних;
- встановити залежність швидкості передачі даних від величини буфера;
- зробити висновок про ефективність протоколів, які досліджуються.

Для дослідження було вибрано сервери хмаркового сховища даних, які територіально розміщені в США та Нідерландах. Канал зв'язку між серверами сховища – 1 GB. “Чорною скринькою” в даному дослідженні є зв'язок, а саме його стабільність та всі можливі брандмауери на даному каналі зв'язку. Експерименти проводилися на файлах великих розмірів, адже як раз при їх передачі виникають проблеми швидкості передачі. Для експерименту було вибрано файл розміром 200 MB. Усі заміри проводились для кожного тесту не менше трьох разів і брались для аналізу усереднені показники. Оскільки експерименти проводилися на одних і тих же даних трафік для всіх експериментів з певною імовірністю є подібним. Передача даних проводилась як в одну, так і в іншу сторону, що дозволило ввести симетричність в отримані результати.

У результаті проведених тестових випробувань було отримано залежності часу передачі файлу від розміру буферу (табл. 4.1).

Таблиця 4.1. Залежність часу передачі файлу від розміру буфера.

Розмір буферу	Час передачі μTP (с)	Час передачі UDT (с)
16384	2439,09	2151,4
32768	1114,08	1811,28
65536	527,74	1655,22
131072	269,1	649,27
262144	136,58	337,64
524288	77,58	169,57
1048576	67,15	67,54
2097152	66,77	29,51
4194304	67,12	13,05
8388608	86,92	6,5
16777216	78,6	6,0
33554432	97,22	6,0
67108864	97,24	6,51
134217728	67,43	5,03
268435456	67,21	6,38
536870912	67,44	5,5

Як видно з графіка залежності часу передачі від розміру буфера (рис. 4.19), швидкість для обох протоколів спадає за експоненціальним законом, що й повинно бути з чисто теоретичних міркувань. Так як міжконтинентальна розподілена мережа є мережею з комутації пакетів, математична модель її представляється Марківським процесом. Робота цього процесу моделюється розподілом Пуассона:

$$P(x) = \frac{\lambda^x e^{-\lambda}}{x!}, \tag{4.1}$$

і відповідно функція розподілу для моделювання часу в процесі Пуассона є експоненціальною функцією:

$$P(x) = \zeta e^{-\lambda x}. \quad (4.2)$$

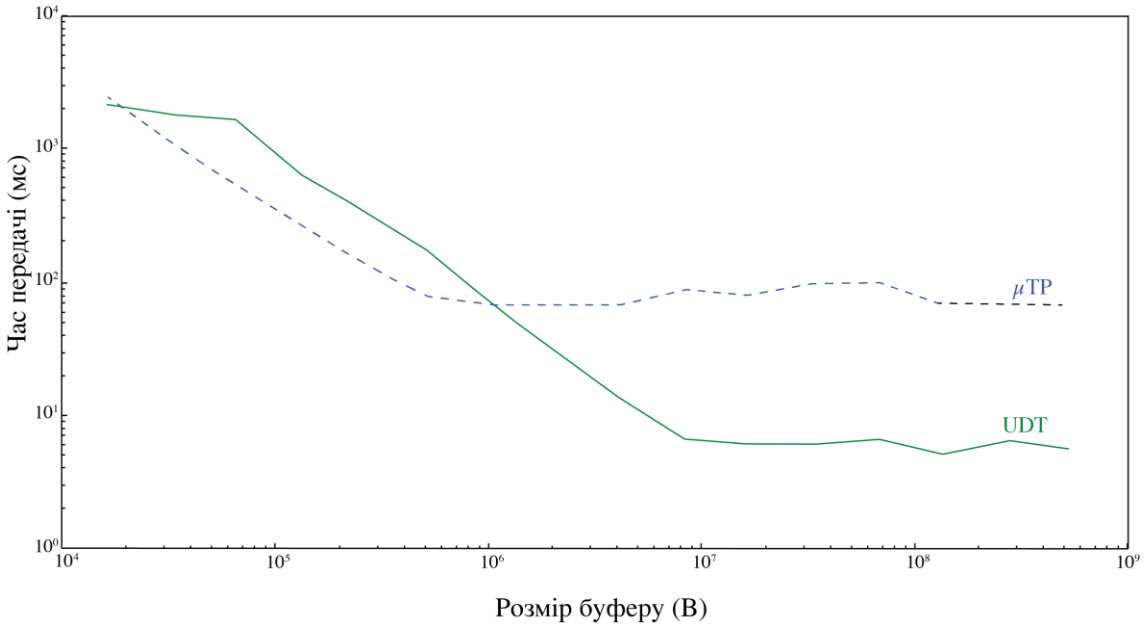


Рис. 4.19. Залежність часу передачі від розміру буфера, розроблено самостійно.

Відповідно до цієї моделі, лінія тренду для протоколу UDT

$$P(x) = 2777e^{-0.47x}, \quad (4.3)$$

при $R^2 = 0{,}89$, а для μTP

$$P(x) = 1778e^{-1.32x}, \quad (4.4)$$

при $R^2 = 0{,}86$.

Хоча при розмірах буфера до 1048576 час передачі за протоколом μTP менший за час передачі за протоколом UDT, після подальшого збільшення розміру буфера протокол UDT далі зберігає тенденцію до зменшення часу передачі, а μTP – час практично не змінюється в межах деякого коридору значень (рис. 4.19). Це може говорити про те, що для протоколу μTP відбувається насичення швидкості, і він не може практично збільшити швидкість передачі. Для протоколу UDT також є така точка, але на рівні розміру буфера – 8388608. При цьому час передачі тестових файлів при оптимальних параметрах обидвох протоколів у 10 разів менший для протоколу UDT, що вже говорить про доцільність його використання у порівнянні з μTP.

Враховуючи, що поточна швидкість змінюється в процесі передачі варто було б і її зміни проаналізувати. Для цього було отримано значення поточної швидкості в залежності від розміру буфера (табл. 4.2). За даними значеннями поточної швидкості було обчислені їх середні значення для різних розмірів буфера та значення середньо-квадратичного відхилення.

Таблиця 4.2. Поточна швидкість передачі файлу.

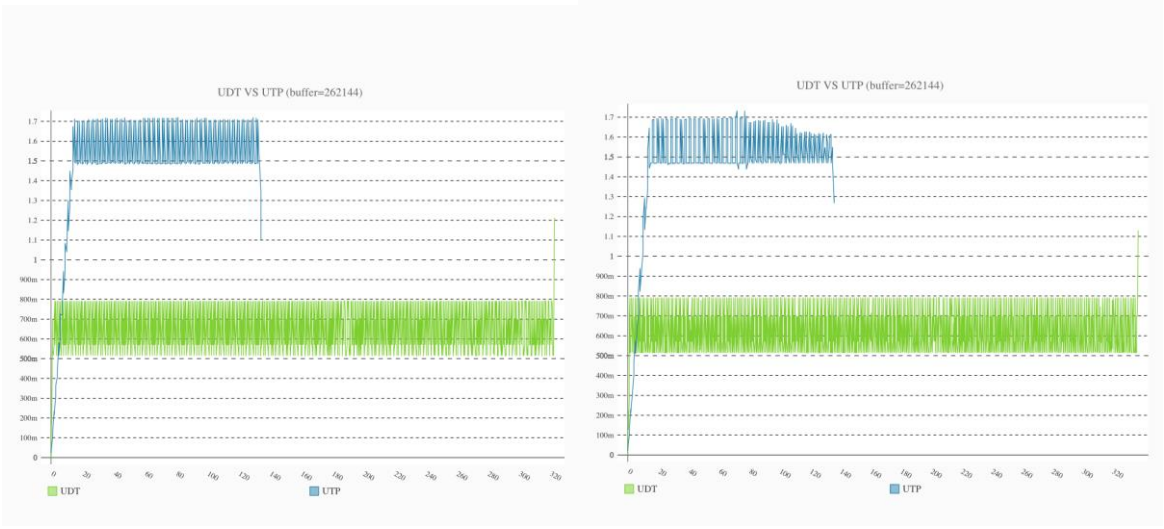
Розмір буферу	μTP		UDT	
	$\hat{v}_{UTP}$	σ_{UTP}	$\hat{v}_{UDT}$	σ_{UDT}
16384	0,08	0,00	0,09	0,22
32768	0,18	0,01	0,11	0,19
65536	0,38	0,03	0,12	0,32
131072	0,74	0,12	0,31	0,44
262144	1,46	0,31	0,59	0,91
524288	2,58	0,97	1,18	1,90
1048576	2,98	1,41	2,96	3,74
2097152	3,00	1,42	6,78	15,78
4194304	2,98	1,40	15,32	52,60
8388608	2,30	1,57	30,75	150,12
16777216	2,54	1,36	33,32	214,96
33554432	2,06	1,63	33,32	253,98
67108864	2,06	1,72	30,74	207,26
134217728	2,97	1,40	39,80	267,50
268435456	2,98	1,41	31,35	249,50
536870912	2,97	1,40	36,35	255,71

Як бачимо з результатів замірів поточних швидкостей передачі при низьких розмірах буфера (до 1048576) середня швидкість μTP і UDT є, практично, співрозмірними, хоча й μTP показує кращі результати.

Для протоколу UDT спостерігається збільшення середньоквадратичного відхилення швидкості, що говорить про зміни швидкості в процесі передачі. Після переломного розміру буфера (8388608) відхилення швидкості стає досить суттєвим. Згідно практичних замірів при даних параметрах мережі середня швидкість протоколу передачі UDT в 10-15 разів вища за середню швидкість протоколом μTP .

Великі значення середньо-квадратичного відхилення вказують на значну зміну швидкості у процесі передачі, що негативно впливає на якість обслуговування клієнтів.

Рисунок (рис. 4.20) демонструє залежність поточної швидкості протоколу μTP на малих буферах стабільну та високу швидкість, як на томість UDT має дещо меншу швидкість і більш не стабільну, з різкими перепадами (про що й говорить більше значення середньоквадратичного відхилення). Аналогічні процеси спостерігаються при збільшенні буферу до 2097152 байт.



а) б)

Рис. 4.20. Швидкість передачі даних при буферах 262144 (а) EU-US (б) US-EU, розроблено самостійно.

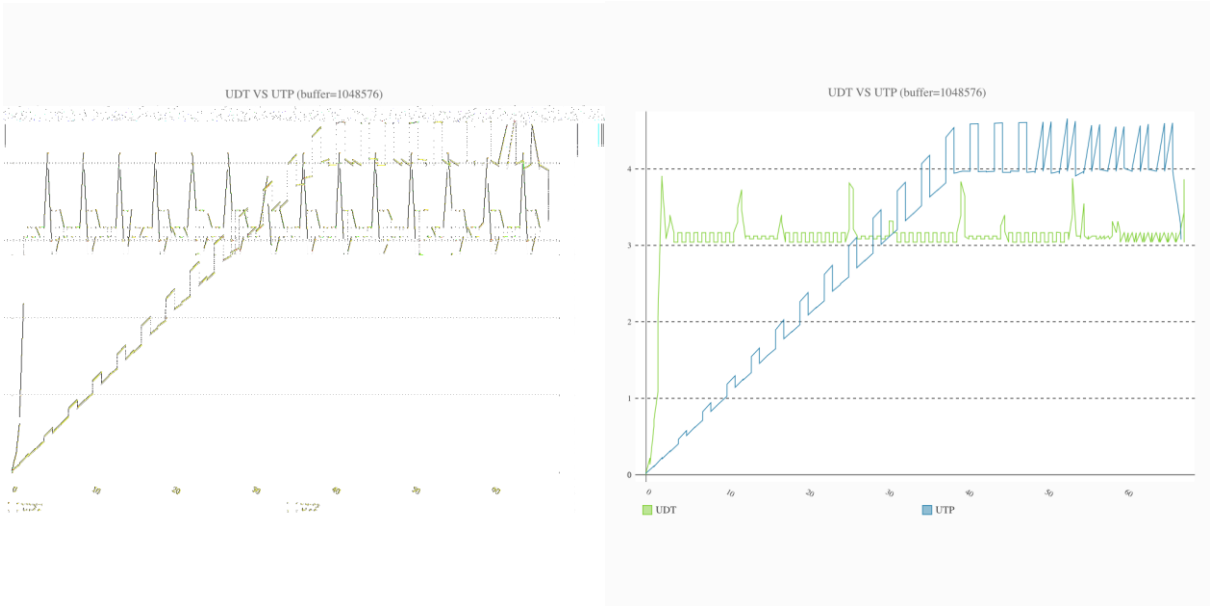
а) б)

Рис. 4.21. Швидкість передачі даних при буферах 524288 (а) EU-US (б) US-EU, розроблено самостійно.

При розмірі буфера 524288 графік швидкості (рис. 4.21) явно показує як протокол *μTP* плавно збільшує швидкість і стабільно її тримає, натомість UDT швидко розганяється і має більшу амплітуду коливань, що вказує на його неможливість стабільно втримувати швидкість.

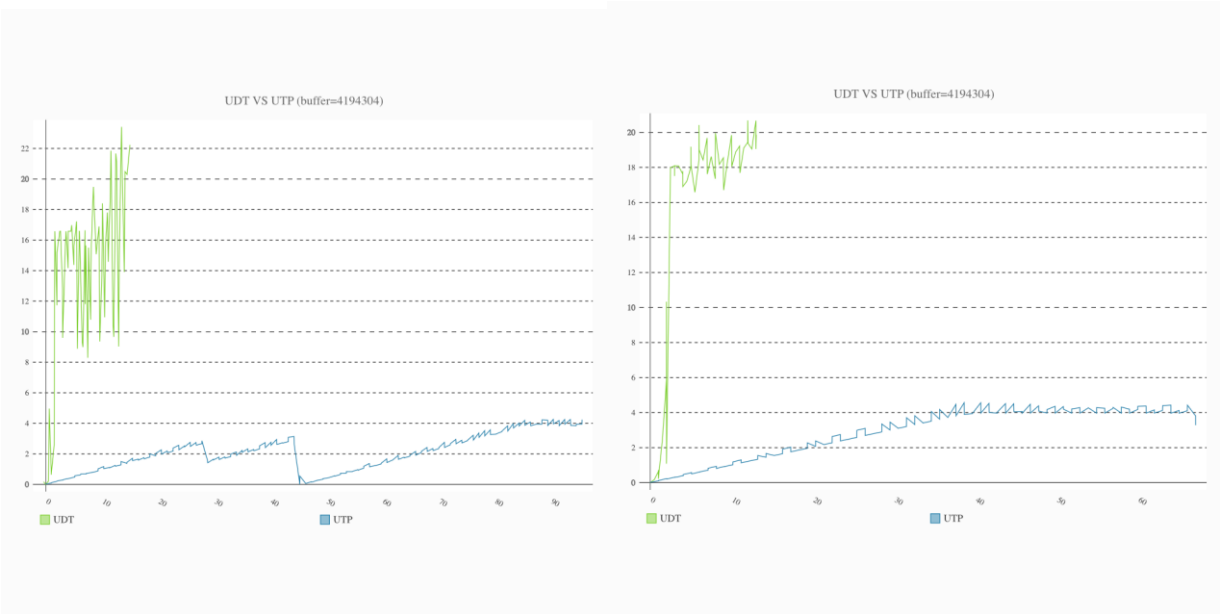
Більш цікавим є результати на рис. 4.22. Графік швидкості чітко демонструє коливання швидкості протоколу UDT, що символізує швидкі спроби збільшити швидкість, але це приводить до швидкого заповнення буферу. При пустому буфері передача зупиняється для отримання даних у буфер, після чого швидкість знову зростає.

Іншим прикладом є рис. 4.23, на якому чітко видно коливання швидкості протоколу UDT, що символізує швидкі спроби збільшити швидкість при відносно малих втратах та швидке падіння швидкості для зменшення втрат. Даний алгоритм управління перевантаженням є стандартний алгоритм UDT, який може бути замінений на користувацький за необхідності.



а) б)

Рис. 4.22. Швидкість передачі даних при буферах 1048576 (а) EU-US (б) US-EU, розроблено самостійно.



а) б)

Рис. 4.23. Швидкість передачі даних при буферах 4194304 (а) EU-US (б) US-EU, розроблено самостійно.

Коли розмір буфера зростає більше 2097152 байт протокол UDT зміг збільшити швидкість передачі даних та заповнити до 78 % каналу передачі даних. Натомість *µTP* , не зміг заповнити більше 7 % каналу передачі даних. Однією з можливих причин може бути вміння сучасних маршрутизаторів та

брандмауерів розрізняти протокол передачі даних торента (μTP) та примусово обмежувати його швидкість.

Дослідження показало, що для протоколів UDT та μTP величина буферу відіграє велику роль у швидкості передачі даних.

При своїй простоті та стабільності протокол μTP не дає бажаних результатів по швидкості.

У свою чергу протокол UDT при своїй нестабільній (велика амплітуда коливань) швидкості передачі даних дає кращі результати за швидкодією. А той фактор, що даний протокол має більш гнучку систему налаштування дає змогу розробникам більш точно сконфігурувати даний протокол, а заданик канал передачі даних, в порівнянні з μTP.

У результаті проведеного дослідження було встановлено, що для ефективної передачі даних через міжконтинентальний канал зв'язку необхідно використовувати протокол передачі даних UDT з достатньо великими буферами обміну (від 2 MB до 8 MB для 1 Gb каналу зв'язку). Буфер обміну для UDT більше 8 MB суттєво не впливає на швидкість передачі, але збільшує середньоквадратичне відхилення швидкості передачі. В свою чергу буфер менше 2 MB для UDT дає гірші показники за μTP. Протокол μTP є більш стабільнішим щодо швидкості та є більш «солідарним» до інших протоколів.

При використанні в хмаркових сховищах даних, серевери якого розміщені на різних континентах і пов'язані між собою високошвидкісною мережею, досить актуальним є висока швидкість передачі даних між серверами. Коливання швидкості не суттєво впливають на роботу системи сховища. Тому отримані в результаті дослідження результати дозволяють зробити висновок про доцільність практичного використання протоколу UDT з достатньо великими буферами передачі для реалізації передачі даних між серверами хмаркового сховища. Практичні експерименти показали, що використання протоколу UDT на високошвидкісних мережах з великим часом затримки достатньо ефективне при збільшенні буферу передачі протоколу.

4.5. Висновки до 4-го розділу.

У результаті роботи у даному розділі:

1. Спроектовано архітектуру системи на основі запропонованих і змодельованих методів організації доступу до хмаркового сховища.

2. Відповідно до спроектованої архітектури проаналізовано та досліджено потоки даних в системі..

3. Проведено практичну реалізацію хмаркового сховища та процесів обміну даними через нього.

4. Проведено порівняльний аналіз запропонованих методів з існуючими та доведено їх ефективність.

Основні результати розділу опубліковані у працях [15, 18, 19].

ВИСНОВКИ

У дисертаційній роботі розв’язано актуальне наукове завдання розроблення моделей та методів підвищення пропускної спроможності розподілених телекомунікаційних систем високодоступних хмарних сховищ даних на основі нових протоколів доступу. Основні результати дисертаційного дослідження викладені у висновках, які зводяться до наступних положень:

1. Проведено аналіз проблем запровадження та функціонування хмаркових сховищ даних. На основі аналізу виділено технологічні та функціональні проблеми впровадження хмаркових сховищ даних. Зосереджено увагу на нерозв’язаних завданнях побудови розподілених систем передавання даних.

2. Удосконалено модель хмаркового сховища даних шляхом подання її як алгебраїчної системи, яка відрізняється від існуючих введенням у архітектуру пов’язаної телекомунікаційної мережі системи методів опрацювання даних на основі протокольних засобів сеансового рівня, що дало змогу більш точно і повно визначити і використовувати її пропускну спроможність відповідно.

3. Здійснено моделювання завантаженості хмаркових сховищ даних для реального сховища, що дало змогу подати завантаженість хмаркового сховища даних як самоподібний процес із довгостроковою залежністю.

4. Розроблено телекомунікаційний протокол сеансового рівня на базі UDP для магістральних розподілених MAN-мереж, який, на відміну від методу вибору протоколу на всій ділянці мережі, характеризується одержаною пропускною спроможністю, вищою від 1.5 до 2 разів, адаптуючись під кожну ділянку мережі окремо.

5. Запропоновано метод агрегації навантаження декількох джерел даних, який, на відміну від методу балансування навантаження, в режимі реального часу визначає завантаженість сервісів хмаркового сховища даних

та каналів телекомунікаційної системи, що дає змогу оптимізувати їх продуктивність.

6. Розроблено метод вибору шлюзу за складністю виконання запиту, який базується на нечітких даних про швидкість обміну з клієнтом і дає змогу вибрати оптимальний за швидкістю маршрут і шлюз передавання даних в реальному часі.

7. Впроваджено створене програмне забезпечення для передавання даних між вузлами хмаркового сховища даних, про що свідчать акти впровадження результатів дисертаційної роботи.

Додаток А. Аналіз архітектур систем зберігання даних.

Зростання обсягів інформації, баз даних, запитів до них, з якими працюють сьогоднішні підприємства, з кожним роком зростають у геометричній прогресії. Тому правильно обрана система зберігання даних – дуже важлива складова успіху в бізнесі. Віртуалізація зберігання і хмаркові сховища можуть допомогти задовольнити сучасні вимоги до зберігання, роблячи їх більш гнучкими і мобільними.

Сервери для зберігання даних, по-перше, необхідні для забезпечення інформаційної безпеки бізнесу. Вона досягається завдяки високому рівню конфіденційності, централізації даних, резервного копіювання даних і т.д.

По-друге, завдяки сучасним системам зберігання даних, збільшується швидкість обробки інформації, а також пошук необхідних даних у величезних інформаційних і дискових масивах.

По-третє, завжди можна бути впевненим, що абсолютно вся нова інформація, скільки величезним б не виявився її обсяг, буде повністю записана і надійно збережена на сервері для зберігання даних (зокрема, проведено резервне копіювання даних на відповідних пристроях).

Зрештою, завдяки надійній системі зберігання даних, можна вирішити проблему втрати важливої інформації, яка була записана на якомусь з дисків.

Найбільш часто при створенні мережі зберігання даних застосовується метод консолідації. Такі системи зберігання файлів і даних представлені в наступних варіантах:

- Система зберігання даних шляхом консолідації на пристроях зберігання (дисковому масиві) з блоковим доступом та підключенням за допомогою мережі зберігання даних SAN (Storage Area Network).

- Система зберігання даних шляхом консолідації на пристроях зберігання з прямим підключенням до мережі передачі даних NAS (Network Attached Storage).

- Система зберігання даних шляхом консолідації на пристроях

зберігання (дискових масивах, стрічкової бібліотеці та інших пристроях резервного копіювання) з прямим підключенням до сервера для зберігання файлів.

Логічне об’єднання фізичних пристроїв мережевого зберігання в один пристрій зберігання даних, який управляється з центрального пульта надає багато зручностей і гнучкості для системних адміністраторів. На сьогоднішній день дане застосування знайшло найбільшу популярність. Управління пристроями зберігання даних може бути виснажливим і трудомістким. На базі мережі варіантів зберігання даних, таких як мережева система зберігання даних NAS, мережі зберігання даних SAN, реалізовано резервне копіювання, архівування та відновлення, маскуючи фактичну складність системи.

Типова архітектура зберігання даних була розроблена 20 років тому, коли були передбачуваними робочі навантаження і структури даних. Але сучасні організації мають справу з безпрецедентною кількістю інформації, в тому числі неструктурованих даних, таких як аудіо і відео, що вимагає значних потужностей. Системи зберігання повинні враховувати безліч різних типів робочих навантажень з різними вимогами до продуктивності [116].

DAS (Direct-attached storage) – система зберігання даних з прямим підключенням – запам’ятовувальний пристрій, який безпосередньо підключений до сервера або робочої станції без допомоги мережі зберігання даних [53]. Цей ретронім в основному використовується для розрізнення не мережевих пристроїв зберігання від SAN і NAS.

Системи типу DAS складаються з накопичувачів (наприклад жорсткого диску), які з’єднані з комп’ютером адаптером контролера шини. Між ними немає мережевого пристрою (концентратора, комутатора чи маршрутизатора), і це є основною ознакою DAS (рис. 1).

Основними протоколами для комунікації в DAS є ATA, SATA, eSATA, SCSI, Serial Attached SCSI і Fibre Channel [53]. Більшість функцій сучасних сховищ не залежать від типу підключення – напряму (DAS) чи через мережі

(SAN і NAS).

Рис. 1. Схематичне зображення DAS, побудовано за даними [116].

Пристрій DAS може використовуватися декількома комп'ютерами, якщо він має декілька інтерфейсів (портів), які дозволяють паралельний прямий доступ. Такий спосіб використовується в кластерах. У реальності більшість пристроїв зберігання SAN чи NAS можуть використовуватися як пристрої DAS – для цього потрібно від'єднати їх порти від мережі даних і приєднати один або більше портів прямо до комп'ютера.

Більш сучасні пристрої DAS, подібні на SAN і NAS, мають підвищену надійність: надійний контролер, система охолодження, засоби резервування при зберіганні типу RAID. Деякі системи DAS мають вбудовані контролери дискових масивів, щоб розвантажити обслуговування RAID адаптером шини сервера. Стандартні ж пристрої DAS не мають таких можливостей.

DAS, як і SAN чи NAS, допускають розширення ємності сховища, зберігаючи швидкість передачі і доступу до даних.

Недоліком DAS є неможливість розділяти дані або ресурси з іншими серверами. Архітектури NAS і SAN роблять спробу зробити це, але при цьому виникають нові проблеми, наприклад, висока початкова вартість, керованість, безпека і конкуренція за ресурси.

Використання мережевої файлової системи дозволяє ширше використовувати переваги цієї технології. Мережева файлова система (NFS - Network File System) – протокол мережного доступу до розподілених файлових систем, розроблений Sun Microsystems у 1984 році, дозволяє користувачеві з клієнтського комп'ютера доступ до файлів в мережі таким же чином, як локальному пристрої пам'яті. NFS є відкритим стандартом, що

дозволяє будь-кому реалізувати цей протокол (рис. 2) [90].

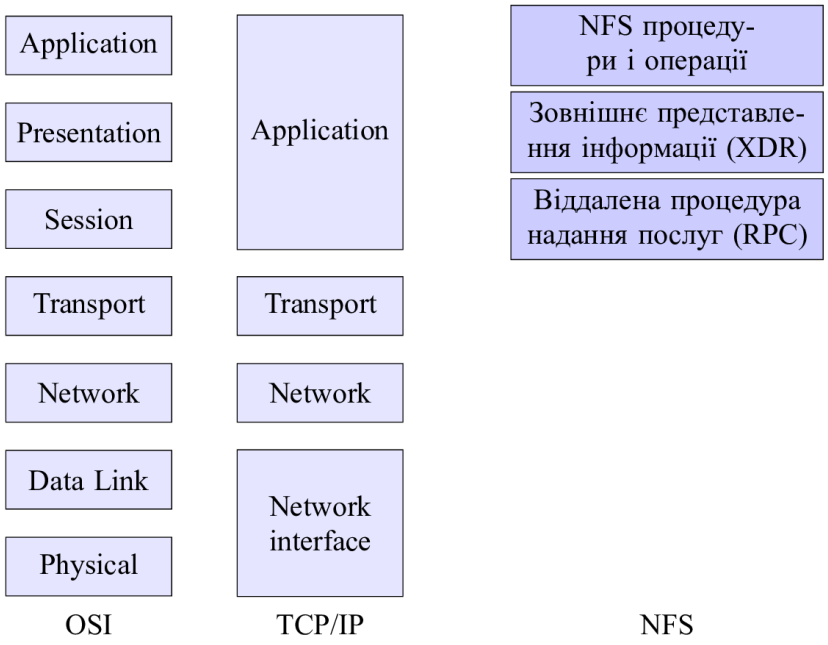


Рис. 2. Блочне представлення NFS як частини OSI і TCP/IP, побудовано за даними [116].

Загальною метою розробки NFS [90] було:

- Незалежність від комп'ютера і операційної системи. Протоколи, які використовуються повинні бути незалежними від операційної системи та доволі простими.

- Відновлення після збоїв. Коли клієнти можуть монтувати віддалені файлові системи з різних серверів і мали можливість легко відновити стан даних після збоїв і проблем в мережі.

- Прозорий доступ. Система повинна дозволяти програмам отримувати доступ до віддалених файлів точно так само як і до локальних даних.

- Підтримка UNIX семантика на клієнтських UNIX. Для того, щоб був прозорий доступ для роботи на машинах UNIX, семантика файлової системи UNIX повинна підтримуватися для віддалених файлів.

- Прийнятна продуктивність. Важливою метою NFS було забезпечити таку ж швидкодію, як і з невеликим локальним диском на інтерфейсі SCSI.

Протокол NFS використовує механізм віддаленого виклику процедур Sun (RPC) [81]. Через ті ж причини, що виклики процедур спрощують

програми, RPC дозволяє спростити визначення, організацію та здійснення дистанційного обслуговування. Протокол NFS визначений у вигляді набору процедур, їх аргументів і результатів роботи. Віддалений виклик процедур є синхронним, тобто блокується клієнтська програма, поки сервер не завершить виклик і поверне результати. Це робить RPC дуже простим у використанні і розумінні – він поводиться, як виклик локальної процедури.

NFS використовується без протоколювання. Параметри кожного виклику процедури містять всю інформацію, яка необхідна для завершення виклику, а сервер не відстежує історію минулих запитів. Це робить відновлення після збою дуже легким, а коли сервер виходить з ладу, клієнт пересилає NFS запити, поки не буде отримана відповідь, і сервер не робить відновлення після збоїв взагалі.

Якщо стан зберігається на сервері, з іншого боку, відновлення набагато складніше. І клієнт, і сервер повинні надійно виявляти збої. Сервер повинен виявляти збої клієнта так, що він може відмовитися від будь-яких станів клієнта, і клієнт повинен виявляти збоїв у роботі сервера, щоб його можна відновити стан сервера. Відсутність протоколювання позбавляє складних відновлень після збоїв.

NAS – мережева система зберігання даних, яка являє собою комп’ютер з певним дисковим масивом, під’єднаний до мережі Ethernet за протоколом TCP/IP [8].

Часто диски в NAS з’єднані у RAID масив. Кілька таких комп’ютерів можуть бути об’єднані в одну систему (рис. 3).

Такі системи забезпечують надійність зберігання даних, зручність доступу для багатьох користувачів, легкість в адмініструванні та масштабованість.

Пристрій NAS є окремим комп’ютером, який може бути побудований як на архітектурі x86, так і на основі ARM, PowerPC, RISC-процесорів. Головним призначенням цього комп’ютера є надання сервісів для зберігання даних іншим пристроям у мережі. Операційна система та програми NAS

забезпечують зберігання даних, файлову систему, доступ до файлів а також контроль над функціями системи. Пристрій не призначено для виконання звичайних обчислювальних завдань, хоча виконання на ньому інших додатків є можливим.

Рис. 3. Схематичне зображення NAS, побудовано за даними [116].

Переважно пристрої NAS не мають монітора й клавіатури, керування і завдання конфігурації відбувається через мережу, зазвичай за допомогою браузера, який під'єднується до пристрою за його мережевою адресою.

Поширені так звані «міні-сервери», у котрих функції NAS поєднано з додатковими сервісами, наприклад, фотоальбом, медіа-центр, BitTorrent-клієнт, e-Mule, Web-сервер тощо. Такі пристрої призначені, в першу чергу, для малих офісів, тому у них рідко встановлюється більше 4 жорстких диски. Головна перевага таких систем – низька вартість в порівнянні з повноцінними серверами, висока швидкість інтеграції.

Дуже часто при зростанні компанії потрібне нарощення дискового простору і менеджери зіштовхуються з вибором між серверами і NAS для забезпечення звичайного доступу до файлів. В цьому випадку NAS мають перевагу не лише за ціною, але й за швидкістю введення у експлуатацію, простотою конфігурування, вартістю експлуатації. Використання електроенергії пристроями NAS на 90 % залежить від кількості і типів встановлених жорстких дисків, а уже пізніше – від вбудованого процесора й пам'яті.

Мережа зберігання даних (SAN) – архітектурне рішення для підключення зовнішніх пристроїв зберігання даних, таких як дискові масиви, стрічкові бібліотеки, оптичні накопичувачі, до серверів таким чином, щоб операційна система розпізнавала підключені ресурси як локальні (рис. 4) [7].

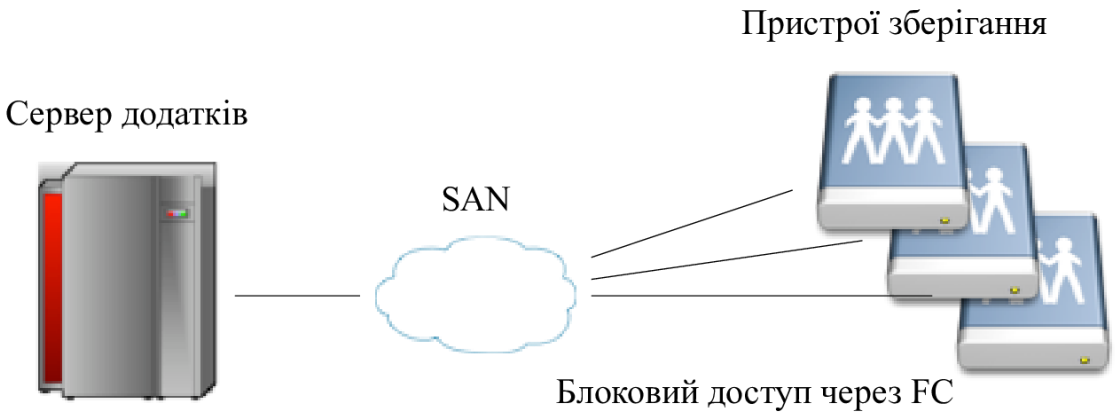


Рис. 4. Схематичне зображення SAN, побудовано за даними [116].

SAN характеризується наданням мережевих блокових засобів, у той час як мережеві сховища даних (NAS) націлені на надання доступу до даних, які зберігаються на їх файловій системі за допомогою мережевої файлової системи (такої як NFS, SMB/CIFS iSCSI або AppleTalk).

Варто також звернути увагу, що категоричний поділ типу «SAN – це тільки мережеві диски, NAS – це тільки мережева файлова система» є штучним: з появою iSCSI (рис. 5) почалося взаємне проникнення технологій з метою підвищення гнучкості і зручності їх застосування. Наприклад, у 2003 році NetApp вже надавали iSCSI на своїх NAS, а EMC і HDS – навпаки, пропонували інтерфейси NAS для своїх SAN-масивів.

Системи SAN забезпечують простий, не типізований, з фіксованим розміром блоку, пам’ять-подібний інтерфейс для управління енергонезалежними магнітними носіями [91]. З точки зору абстракції каналу даних функціональна відмінність між інтерфейсами системи SAN і традиційним підключенням дисків невелика.

Рис. 5. Схематичне зображення iSCSI, побудовано за даними [116].

Навіть якщо мережа SAN дає можливість кільком клієнтам доступу до того ж пристрою зберігання даних безпосередньо, цей тип обміну вимагає координації і не представлений інтерфейсом SAN для забезпечення синхронізації одночасного доступу. Тут згруповані разом пристрої зберігання і мережева підтримка до якої приєднані пристрої зберігання, звертаючись до них під загальною назвою системи SAN. У цьому сенсі, диск Fibre Channel представляє собою пристрій зберігання даних. Більш вузьке тлумачення включає тільки концентратори, комутатори і маршрутизатори.

На перший погляд, рішення SAN і NAS видаються майже ідентичні, в реальному житті вони можуть застосовуватися в будь-якому їх поєднанні. Загалом, рішення NAS і SAN використовують системи RAID, які підключені до мережі. Проте, існують значні відмінності між рішеннями SAN і NAS, які мають величезний вплив на те, як здійснюється доступ і використання фактичних даних.

Технологічно системи NAS і SAN пропонують різні абстракції до використовного програмного забезпечення. У самому простому випадку, різниця між абстракції NAS для зберігання та абстракції SAN полягає в тому,

що системи NAS пропонує функції файлової системи для клієнтів, а SAN такої системи немає.

Більшість мереж зберігання даних використовує протокол SCSI для зв'язку між серверами та пристроями зберігання даних на рівні шинної топології. Оскільки цей протокол не призначений для формування мережевих пакетів, в мережах зберігання даних використовують низькорівневі протоколи.

Віртуалізація систем зберігання даних є очевидним об'єднання даних з декількох пристроїв зберігання, навіть різного типів пристроїв зберігання, в єдиний, що видається як один пристрій, який управляється з центральної консолі.

До переваг віртуалізації зберігання можна віднести [92]:

- Прозорий доступ. Система повинна дозволяти програмам отримувати доступ до віддалених файлів точно так само як і до локальних даних.

- Підтримка UNIX семантика на клієнтських UNIX. Для того, щоб був прозорий доступ для роботи на машинах UNIX, семантика файлової системи UNIX повинна підтримуватися для віддалених файлів.

- Спрощення адміністрування системи.
- Розширення можливостей SAN:
  - Автоматичний розподіл.
  - Автоматична відмовостійкість.
  - Знімок.
  - Висока доступність (синхронне віддзеркалення).
  - Асинхронне дзеркальне відображення IP.
- Значне використання томів.
- Високопродуктивна система кешування.
- Спрощує зберігання розширення.
- Неперервне резервування системи зберігання.

Численні дослідження ефективності зберігання даних на локальних

комп’ютерах показують, що реальний середній коефіцієнт використання дискового простору лише 18 %, що навіть менший, ніж прийнятий для промисловості – 40 %.

Вимогою дня є гнучкий і взаємозамінний пул зберігання. Адміністратор зберігання повинен мати можливість швидко налаштувати зберігання при потребі, а потім так само швидко змінити налаштування для подальшого використання. Зберігання повинне мати можливість швидко розширюватися, так щоб дані і додатки могли легко і надійно мігрувати, а робоче навантаження повинно бути автоматично збалансованим. Критично важливі програми повинні бути в режимі онлайн весь час, тому висока доступність має першорядне значення. Управління всім пулом зберігання, а також координація з віртуальних серверів і мереж повинна бути впорядкована і спрощена.

За даними опитування, проведеного в 2008 і 2012 роках, про методи зберігання показують, що Fibre Channel SAN є кращим вибором для віртуальних машин зберігання, хоча їх чисельність значно впали у порівнянні з 2008 роком (з 52 % до 37 %). Чисельність iSCSI SAN (мережеві пристрої зберігання даних і сховища з прямим підключенням) зростає в цей період часу.

На сьогодні більшість підприємств переходять до зберігання своїх даних за допомогою хмаркових технологій. Досить велика увага цьому напрямку приділяється у практичному розрізі, але не достатньо описано теоретичні засади таких методів зберігання. Тому, перемістимо фокус на теоретичний аналіз можливостей хмаркового зберігання даних.

Додаток Б. Акти впровадження результатів дисертаційної роботи.

"ЗАТВЕРДЖУЮ"
Проректор з наукової роботи
Національного університету
«Львівська політехніка»
_____Чухрай Н.І.
_____ 2016 р.

АКТ
про використання результатів дисертаційної роботи
Струбицького Ростислава Павловича,
представленої на здобуття наукового ступеня кандидата технічних наук,
при виконанні науково-дослідної роботи кафедри інформаційних систем та мереж Національного університету «Львівська політехніка» за темою
«Моделі та алгоритми побудови хмаркових сховищ даних у розподілених інформційних системах»

Начальник науково-дослідної частини к.т.н., доцент Жук Л.В. та члени комісії: завідувач відділу науково-організаційного супроводу наукових досліджень к.т.н. Лазько Г.В та заступник завідувача планово-фінансового відділу Чулой Т.М цим актом підтверджують, що результати дисертаційної роботи аспіранта кафедри інформаційних систем та мереж Струбицького Р.П. використано при виконанні науково-дослідної роботи кафедри інформаційних систем та мереж Національного університету «Львівська політехніка» "Комплекс інтелектуальний інформаційних технологій інтеграції даних для обліку та аналізу підвищення кваліфікації вчителів" № держреєстрації 0113U005273. Зокрема, розроблено метод мультипротокольної передачі даних у хмаркових сховищах даних; удосконалено модель хмаркового сховища даних; розроблено архітектуру хмаркового сховища даних. Отримані автором результати використано для ефективного доступу до даних та для консолідованого, розподіленого зберігання даних.

Голова комісії:

Начальник
науково-дослідної частини
к.т.н., доцент Л.В.Жук

Члени комісії:

завідувач кафедри ІСМ В.В.Литвин

завідувач відділу НОСНД Г.В.Лазько

заступник завідувача ПФВ Т.М. Чулой

YPSILON.NET AG

REFERENCE

über die Umsetzung der Forschungsergebnisse
von Strubytskyi Rostyslav

Dieses Zertifikat bestätigt, dass die Anforderung einer länderübergreifenden Cloud-Datenspeicherung in unserem Unternehmen umgesetzt wurde, die Ergebnisse wurden in der Doktorandenarbeit der Lviv National University "Lviv Polytechnic" Fachbereich ISM von Herrn Strubytskyi Rostyslav erfasst.

1. Die Entwicklung eines Session-Level-Protokoll auf Basis von UDP für die Vernetzung von Unternehmen.
2. Das Verfahren Streaming-Daten zwischen den Satelliten mit verschieden Protokoll zu ermöglichen.
3. Die Methode der Auswahl des Gateways nach komplexität auf den Speicher.

Die Forschungsergebnisse wurden in der Praxis in unserem Unternehmen implementiert um:

- Die Geschwindigkeit der verteilten Daten zu erhöhen
- Die Ausfallsicherheit der Übertragungssysteme zu gewährleisten.

CEO, Hans Joachim Klenz

GLOBALpay
глобальна платіжна мережа

ТОВ «ГЛОБАЛЬНА ПЛАТІЖНА МЕРЕЖА»
04073, м.Київ, Московський проспект, 6
Тел.: 0 44 468 2 772, факс: 0 44 468 3 154

Для пред'явлення
за місцем вимоги

Довідка
про використання результатів наукової діяльності

Видана Струбицькому Ростиславу Павловичу, про те, що в ТОВ «ГЛОБАЛЬНА ПЛАТІЖНА МЕРЕЖА» були використані результати наукових досліджень, а саме:

1. Оптимізована архітектура побудови IPSP;
2. Алгоритми оптимального роутингу між еквайринговими системами;
3. Комбіновані методи оцінки ризиків шахрайств.

Практична реалізація цих досліджень стала основою для побудови системи IPSP для оплат товарів/послуг за домопогою мережі Інтернет, що дало змогу зменшити час очікування проведення транзакції та збільшити безпеку проведення платежів.

Генеральний директор **В.О. Рязанов**

GLOBALpay
глобальна платіжна мережа

ТОВ «ГЛОБАЛЬНА ПЛАТІЖНА МЕРЕЖА»
04073, м.Київ, Московський проспект, 6
Тел.: 0 44 468 2 772, факс: 0 44 468 3 154

Для пред'явлення
за місцем вимоги

Довідка
про використання результатів наукової діяльності

Видана Струбицькому Ростиславу Павловичу, про те, що в ТОВ «ГЛОБАЛЬНА ПЛАТІЖНА МЕРЕЖА» дійсно були використані результати наукових досліджень, а саме:

1. Оптимізована архітектура побудови Платіжної системи;
2. Алгоритми оптимального роутингу між учасниками платіжної системи;
3. Комбіновані методи оцінки ризиків шахрайств.

Практична реалізація цих досліджень стала основою для побудови платіжної системи, на базі якої надаються послуги оплати товарів та послуг, що дало змогу зменшити час очікування проведення транзакції та збільшити безпеку проведення платежів.

Генеральний директор **В.О. Рязанов**

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бессараб В. І. Генератор самоподібного трафіку для моделей інформаційних мереж / В. І. Бессараб, Е. Г. Ігнатенко, В. В. Черівнський. // Наукові праці донецького національного технічного університету. – 2008. – №130. – С. 23–29.

2. Бєльков Д. В. Дослідження мережевого трафіку / Дмитро Вікторович Бєльков. // Наукові праці донецького національного технічного університету. – 2009. – №153. – С. 212–215.

3. Кузьмук В. В. Класифікація мереж Петрі та приклади їх застосування для розв'язання прикладних задач / В. В. Кузьмук, А. М. Парнюк, О. А. Супруненко. // Восточно-Европейский журнал передовых технологий. – 2011. – №50. – С. 40–43.

4. Кучук Г. А. Аналіз та моделі самоподібного трафіка / Г. А. Кучук, О.О. Можаєв, О. В. Воробйов. // Авиационно-космическая техника и технология. – 2006. – №9. – С. 173–180.

5. Пат. US Patent 13/493,859. Content delivery network. /Gagliardi, J.D., Munger T.S., Ploesser D.W. - 2012.

6. Пат. US Patent 5,768,271. Virtual private network. /Lespagnol A., Seid H.A. - 1998.

7. Пат. US Patent 6,564,228. Method of enabling heterogeneous platforms to utilize a universal file system in a storage area network. /O'connor M.A. - 2003.

8. Пат. US Patent D563,994. Network attached storage. /Liu C.Y., Chen H.B. – 2008.

9. Платов В. В. Дослідження самоподібної структури телетрафіку бездротової мережі / В. В. Платов, В. В. Петров. // Електротехнічні та інформаційні комплекси і системи. – 2004. – №3. – С. 38–49.

10.Струбицький Р. П. Аналіз загроз хмарко-вим сховищам даних та методи їх захисту / Р. П. Струбицький, П. Р. Струбицький, Н. Б. Шаховська. // Збірник наукових праць науково-практичної конференції «Стан та удосконалення безпеки інформаційно-телекомунікаційних систем (SITS'2013)». – 2013. – С. 68–72.

11.Струбицький Р. П. Аналіз загроз хмарковим сховищам даних та методів їх захисту / Р. П. Струбицький, П. Р. Струбицький, Н. Б. Шаховська. // Наукові праці. Комп'ютерні технології. – 2013. – №217. – С. 35–38.

12.Струбицький Р. П. Аналіз інфраструктури та моделей організації хмаркових сховищ даних / Р. П. Струбицький, Н. Б. Шаховська. // Вісник Національного університету «Львівська політехніка». – 2014. – №783. – С. 225–233.

13.Струбицький Р. П. Аналіз підходів до моделювання хмаркових сховищ даних / Р. П. Струбицький, Н. Б. Шаховська. // Актуальні проблеми економіки : Науковий економічний журнал. – 2013. – №11. – С. 263–269.

14.Струбицький Р. П. Моделювання транспортних протоколів доступу до хмаркових сховищ даних / Ростислав Павлович Струбицький. // Матеріали міжнародної науково-практичної конференції молодих вчених та студентів «Інформаційні технології, економіка та право: стан та перспективи розвитку». – 2014. – №11. – С. 47–48.

15.Струбицький Р. П. Підходи до мультиплексування передачі документів у розподілених мережах / Ростислав Павлович Струбицький. // Збірник тез доповідей науково-практичної конференції «Стан та удосконалення безпеки інформаційно-телекомунікаційних систем (SITS'2015)». – 2015. – С. 99–100.

16.Струбицький Р. П. Побудова моделей високошвидкісних потоків для хмаркових сховищ / Ростислав Павлович Струбицький. // Матеріали V Всеукраїнської школи-семінару молодих вчених і студентів «Сучасні комп'ютерні інформаційні технології (ACIT'2015)». – 2015. – С. 161–163.

17.Струбицький Р. П. Порівняльний аналіз швидкості роботи протоколів гарантованої передачі даних, базованих на UDP у міжконтинентальній мережі / Ростислав Павлович Струбицький. // Вісник Хмельницького національного університету. – 2015. – №2. – С. 173–177.

18.Струбицький Р. П. Практична реалізація відмовостійкого захищеного сховища даних навчального закладу засобами відкритого програмного забезпечення / Р. П. Струбицький, П. Р. Струбицький. // Збірник наукових праць науково-практичної конференції «Стан та удосконалення безпеки інформаційно-

телекомунікаційних систем (SITS'2014)». – 2014. – С. 32–34.

19.Струбицький Р. П. Практичний метод вибору оптимального шлюзу доступу до хмаркового сховища / Р. П. Струбицький, О.М. Найда. // «Проблеми автоматизації та управління». Збірник наукових праць: випуск 3(51). К.: НАУ. – 2015. – С. 110–115.

20.Струбицький Р. П. Розробка протоколу сеансового рівня для високошвидкісних регіонально-розподілених мереж / Ростислав Павлович Струбицький. // Проблеми інформатизації та управління. – 2015. – №50. – С. 109–117.

21.Струбицький Р. П. Самоподібна модель завантаженості хмаркових сховищ даних / Ростислав Павлович Струбицький. // Вісник Національного університету «Львівська політехніка». – 2015. – №814. – С. 147–156.

22.Таненбаум Э. С. Компьютерные сети / Эндрю С. Таненбаум. – СПб: Издательский дом «Питер», 2012. – 955 с.

23.Хоар Ч. Взаимодействующие последовательные процессы / Ч. Хоар. – М.: Мир, 1989. – 264 с.

24.A cloud storage architecture model for data-intensive applications / Y.Huo, H. Wang, L. Hu, H. Yang. // Computer and Management (CAMAN), 2011 International Conference on. IEEE. – 2011. – С. 1–4.

25.Adas A. Traffic models in broadband networks / Abdelnaser Adas. // Communications Magazine, IEEE. – 1997. – №35. – С. 82–89.

26.Al-Fares M. A scalable, commodity data center network architecture / M. Al-Fares, A. Loukissas, A. Vahdat. // ACM SIGCOMM Computer Communication Review. – 2008. – №38. – С. 63–74.

27.Amazon Web Services [Електронний ресурс] // A.E.C. Retrieved November. – 2011. – Режим доступу до ресурсу: http://dclug.tux.org/200611/AmazonEC2.pdf.

28.An architecture for differentiated services [Електронний ресурс] / [E. Davies, M. Carlson, W. Weiss та ін.] // Informational. – 1998. – Режим доступу до ресурсу: https://tools.ietf.org/pdf/rfc2475.pdf.

29.Arrington M. Gmail Disaster: Reports Of Mass Email Deletions [Електронний

ресурс] / Michael Arrington. – 2006. – Режим доступу до ресурсу: http://techcrunch.com/2006/12/28/gmail-disaster-reports-of-mass-email-deletions/.

30.Avelar V. Guidelines for Specifying Data Center Criticality / Tier Levels [Електронний ресурс] / Victor Avelar // APC. – 2007. – Режим доступу до ресурсу: http://www.lamdahellix.com/assets/contents/files/122_whitepaper.pdf.

31.Braden R. Integrated services in the Internet architecture: an overview [Електронний ресурс] / R. Braden, D. Clark, S. Shenker. – 1994. – Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc1633.pdf.

32.Brownlee N. Understanding Internet traffic streams: dragonflies and tortoises / N. Brownlee, K. C. Claffy. // Communications Magazine, IEEE. – 2002. – №40. – С. 110–117.

33.Callon R. Multiprotocol Label Switching Architecture [Електронний ресурс] / R. Callon, E. C. Rosen, A. Viswanathan // Standards Track. – 2001. – Режим доступу до ресурсу: http://tools.ietf.org/pdf/rfc3031.pdf.

34.Cloud computing and grid computing 360-degree compared / I.Foster, Y. Zhao, I. Raicu, S. Lu. // Grid Computing Environments Workshop. – 2008. – С. 1–10.

35.Cloud computing: distributed internet computing for IT and scientific research / [M. D. Dikaiakos, D. Katsaros, P. Mehra та ін.]. // Internet Computing, IEEE. – 2009. – №13. – С. 10–13.

36.Data Management and Transfer in High Performance Computational Grid Environments / [B. Allcock, J. Bester, J. Bresnahan та ін.]. // Parallel Computing. – 2002. – №29. – С. 749–771.

37.Dekeyser S. Extending google docs to collaborate on research papers / S. Dekeyser, R. Watson. // University of Southern Queensland, Australia. – 2008. – №23. – С. 1–11.

38.Dickens P. M. FOBS: A lightweight communication protocol for grid computing / Phillip M. Dickens. // Euro-Par 2003 Parallel Processing. – 2003. – С. 938–946.

39.Efficient Replica Maintenance for Distributed Storage Systems / [B. G. Chun, F. Dabek, A. Haeberlen та ін.]. // NSDI. – 2006. – №6. – С. 45–58.

40.Equation-based congestion control for unicast applications / S.Floyd, M. Handley, J. Padhye, J. Widmer. // SIGCOMM Computer Communication Review. – 2000. – №30. – С. 43–56.

41.Experimental studies using photonic data services at IGrid 2002 / [R.L. Grossman, Y. Gu, D. Hamelburg та ін.]. // Future Generation Computer Systems. – 2003. – №6. – С. 945–955.

42.Fall K. R. TCP/IP Illustrated, Volume 1: The Protocols / K. R. Fall, R.W. Stevens., 2011. – 850 с.

43.FAST TCP: motivation, architecture, algorithms, performance / D. X.Wei, C. Jin, S. H. Low, S. Hegde. // IEEE/ACM Transactions on Networking (ToN). – 2006. – №6. – С. 1246–1259.

44.Feng W. The failure of TCP in high-performance computational grids / W. Feng, P. Tinnakornsrisuphap. // Supercomputing, ACM/IEEE 2000 Conference. IEEE. – 2000. – С. 37–37.

45.Floyd S. Datagram congestion control protocol (DCCP) [Електронний ресурс] / S. Floyd, M. Handley, E. Kohler // Standards Track. – 2006. – Режим доступу до ресурсу: https://tools.ietf.org/pdf/rfc4340.pdf.

46.Floyd S. HighSpeed TCP for Large Congestion Windows [Електронний ресурс] / Sally Floyd // Experimental Standard. – 2003. – Режим доступу до ресурсу: http://tools.ietf.org/pdf/rfc3649.pdf.

47.Floyd S. Promoting the use of end-to-end congestion control in the Internet / S. Floyd, K. Fall. // IEEE/ACM Transactions on Networking (TON). – 1999. – №4. – С. 458–472.

48.Floyd S. TCP friendly rate control (TFRC): Protocol specification [Електронний ресурс] / S. Floyd, J. Padhye, J. Widmer // Standards Track. – 2008. – Режим доступу до ресурсу: https://tools.ietf.org/pdf/rfc5348.pdf.

49.Foster I. The anatomy of the grid: Enabling scalable virtual organizations / I. Foster, C. Kesselman, S. Tuecke. // International journal of high performance computing applications. – 2001. – №3. – С. 200–222.

50.Frost V. S. Traffic Modeling for Telecommunications Networks / V. S. Frost,

B. Melamed. // Communications Magazine, IEEE. – 1994. – №32. – С. 70–81.

51.Fundamentals of Queueing Theory / D.Gross, J. F. Shortle, J. M. Thompson, J. M. Harris., 2011. – 528 с.

52.Girault C. Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications / C. Girault, R. Valk., 2013. – 607 с.

53.Goda K. Direct Attached Storage / Kazuo Goda. // Encyclopedia of Database Systems. – 2009. – С. 847–847.

54.Gordon J. Pareto process as a model of self-similar packet traffic / James Gordon. // Global Telecommunications Conference, IEEE. – 1995. – №3. – С. 2232–2236.

55.Gu Y. End-to-end congestion control for high performance data transfer / Y. Gu, R. Grossman. // IEEE/ACM Transaction on Networking. – 2003. – №3. – С. 1–12.

56.Gu Y. Experiences in Design and Implementation of a High Performance Transport Protocol / Y. Gu, X. Hong, R. L. Grossman. // Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference. IEEE. – 2004. – С. 22–22.

57.Gu Y. SABUL: A Transport Protocol for Grid Computing / Y. Gu, R. Grossman. // Journal of Grid Computing. – 2003. – №4. – С. 377–386.

58.Gu Y. Supporting configurable congestion control in data transport services / Y. Gu, R. Grossman. // Proceedings of the 2005 ACM/IEEE conference on Supercomputing. – 2005. – С. 31–62.

59.Gu Y. UDT: UDP-based data transfer for high-speed wide area networks / Y. Gu, R. Grossman. // Computer Networks. – 2007. – №7. – С. 1777–1799.

60.Harmantzis F. Heavy Network Traffic Modeling and Simulation using Stable FARIMA Processes / F. Harmantzis, D. Hatzinakos. // IEEE Transactions on Signal Processing. – 2000. – №5. – С. 48–50.

61.Herrick D. R. Google this!: using Google apps for collaboration and productivity / Dan Herrick. // Proceedings of the 37th annual ACM SIGUCCS fall conference. – 2009. – С. 55–64.

62.Hewitt C. ORGs for Scalable, Robust, Privacy-Friendly Client Cloud Computing / Carl Hewitt. // IEEE internet computing. – 2008. – №5. – С. 96–99.

63.Heyman D. P. Stochastic Models in Operations Research, Stochastic Processes and Operating Characteristics, Vol. 1. / D. P. Heyman, M. J. Sobel., 2003. – 547 с.

64.Huebner F. Queueing performance comparison of traffic models for Internet traffic / F. Huebner, D. Liu, J. M. Fernandez. // Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE. – 1998. – №1. – С. 471–476.

65.Hurst H. E. Long term storage capacity of reservoirs / Harold Edwin Hurst. // Transaction of the American society of civil engineer. – 1951. – №116. – С. 770–799.

66.Hurst H. E. Long-Term Storage. An Experimental Study / H. E. Hurst, R.P. Black, Y. M. Simaika. – London: Constable, 1965. – 145 с.

67.Jain R. Packet Trains-Measurements and a New Model for Computer Network Traffic / R. Jain, S. Routhier. // IEEE Journal on Selected Areas in Communications. – 2006. – №6. – С. 986–995.

68.Jensen K. Coloured Petri nets: basic concepts, analysis methods and practical use. Vol. 1. / Kurt Jensen., 2013. – 236 с.

69.Jones M. T. Anatomy of a cloud storage infrastructure. [Електронний ресурс] / Tim Jones // IBM developer works (November 30, 2010). – 2010. – Режим доступу до ресурсу: http://www.ibm.com/developerworks/cloud/library/cl-cloudstorage/cl-cloudstorage-pdf.pdf.

70.JuxtaView - A tool for interactive visualization of large imagery on scalable tiled displays / [N. K. Krishnaprasad, V. Vishwanath, S. Venkataraman та ін.]. // Cluster Computing, 2004 IEEE International Conference. – 2004. – С. 411–420.

71.Katabi D. Congestion control for high bandwidth-delay product networks / D. Katabi, M. Handley, C. Rohrs. // ACM SIGCOMM Computer Communication Review. – 2002. – №4. – С. 89–102.

72.Kelly T. Notes on Effective Bandwidths / Tom Kelly. // Royal Statistical Society Lecture Notes Series. – 1996. – №4. – С. 111–135.

73.Kelly T. Scalable TCP: Improving performance in highspeed wide area networks / Tom Kelly. // ACM SIGCOMM computer communication Review. – 2003. – №2. – С. 83–91.

74.Kleinrock L. Queueing Systems: Theory / Leonard Kleinrock., 1976. – 448 c.

75.Kurose J. F. Computer Networking: A Top-down Approach / J. F. Kurose, K. W. Ross., 2013. – 862 c.

76.Logic Synthesis for Asynchronous Controllers and Interfaces / [J. Cortadella, M. Kishinevsky, A. Kondratyev та ін.]. – Berlin: Springer Berlin Heidelberg, 2012. – 273 c.

77.Meiss M. R. Tsunami: A high-speed rate-controlled protocol for file transfer [Електронний ресурс] / Mark Meiss // Indiana University. – 2004. – Режим доступу до ресурсу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.4551&rep=rep1&type=pdf .

78.Michiel H. Teletraffic engineering in a broad-band era / H. Michiel, K. Laevens. // Proceedings of the IEEE. – 1997. – №12. – С. 2007–2033.

79.Miller L. J. The ISO Reference Model of Open Systems Interconnection: A first tutorial / Leslie Jill Miller. // Proceedings of the ACM '81 conference. – 1981. – С. 283–288.

80.Modeling Internet Backbone Traffic at the Flow Level / [C. Barakat, P. Thiran, G. Iannaccone та ін.]. // Signal Processing, IEEE Transactions. – 2003. – №8. – С. 2111–2124.

81.Muller G. Scaling Up Partial Evaluation for Optimizing the Sun Commercial RPC Protocol / G. Muller, E. Volanschi, R. Marlet. // ACM SIGPLAN Not.. – 1997. – №32. – С. 116–126.

82.O'Reilly T. What is web 2.0 / Tim O'Reilly., 2009. – 12 c.

83.On the self-similar nature of Ethernet traffic (extended version) / W. E.Leland, M. S. Taqqu, W. Willinger, D. V. Wilson. // Networking, IEEE/ACM Transactions. – 1994. – №1. – С. 1–15.

84.Open, Closed, and Mixed Networks of Queues with Different Classes of Customers / F.Baskett, K. M. Chandy, R. R. Muntz, F. G. Palacios. // Journal of the ACM. – 1975. – №22. – С. 248–260.

85.Park K. Self-Similar Network Traffic and Performance Evaluation / K. Park,

W. Willinger. – New Jersey: Wiley, 2000. – 576 с.

86.Reliable blast UDP: Predictable high performance bulk data transfer / E.He, J. Leigh, O. Yu, T. DeFanti. // Cluster Computing, 2002. Proceedings. IEEE International Conference. – 2002. – С. 317–324.

87.Ritakari J. Gbit/s VLBI and eVLBI with off-the-shelf components / J. Ritakari, A. Mujunen. // International VLBI Service for Geodesy and Astrometry. – 2004. – С. 182–185.

88.Roberts J. W. Internet traffic, QoS, and pricing / James Roberts. // Proceedings of the IEEE. – 2004. – №92. – С. 1389–1299.

89.Russell T. Telecommunications Protocols / Travis Russell., 2000. – 427 с.

90.Sahni P. Network File System / P. Sahni, A. Batra. // International Journal of Research. – 2015. – №4. – С. 894–896.

91.Sandberg R. The Sun network file system: Design, implementation and experience / Russel Sandberg. // Distributed Computing Systems: Concepts and Structures. – 1987. – С. 300–316.

92.Sasidhar T. A Generalized Cloud Storage Architecture with Backup Technology for any Cloud Storage Providers / T. Sasidhar, P. K. Illa, S. Kodukula. // International Journal of Computer Application. – 2012. – №2. – С. 2250–1797.

93.Self-similarity Through High-variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level / W.Willinger, M. S. Taqqu, R. Sherman, D. V. Wilson. // Networking, IEEE/ACM Transactions. – 1997. – №1. – С. 71–86.

94.Service Level Agreements for Cloud Computing / P.Wieder, J. M. Butler, W. Theilmann, R. Yahyapour. – New York: Springer, 2011. – 358 с.

95.Shakhovska N. Model of Data Warehouse with Uncertain Consolidated Data / N. Shakhovska, R. Strubytskyi. // Applied computer science. – 2015. – №9. – С. 1753–1762.

96.Shin S. Understanding the performance of TCP and UDP-based Data Transfer Protocols using EMULAB / S. Shin, K. Dhondge, B. Choi. // First GENI Research and Educational Experiment Workshop. – 2012. – №13. – С. 148–153.

97.Stallings W. High-speed Networks and Internets: Performance and Quality of

Service / William Stallings., 2002. – 715 с.

98.Stewart R. Stream Control Transmission Protocol [Електронний ресурс] / Randall Stewart // Standards Track. – 2007. – Режим доступу до ресурсу: https://tools.ietf.org/pdf/rfc4960.pdf.

99.Strubytskyi R. Modeling and forecating of cloud data warehousing load / Rostyslav Strubytskyi. // Applied Computer Science. – 2014. – №10. – С. 30–43.

100.Strubytsky R. Organization of Cloud Storage Data in Distributed Systems // Proceeding of the XIIIth International Conference "Modern problems of radio engineering, telecommunications and computer science (tcset'2016)" Lviv–Slavske, Ukraine, February 23–26, 2016. С. 463–467

101.Tanenbaum A. S. Computer Networks / A. S. Tanenbaum, D. Wetherall. – Pearson: Prentice Hall, 2011. – 933 с.

102.Thajchayapong S. Mobility patterns in microcellular wireless networks / S. Thajchayapong, J. M. Peha. // IEEE Wireless Communications and Networking Conference. Proceedings. – 2003. – №3. – С. 52–63.

103.The Pseudo-self-similar Traffic Model: Application and Validation / R. E.Khayari, R. Sadre, B. Haverkort, A. Ost. // Performance Evaluation. – 2004. – №56. – С. 3–22.

104.The SDSS SkyServer - Public Access to the Sloan Digital Sky Server Data / [A. S. Szalay, J. Gray, A. R. Thakar та ін.]. // Proceedings of the 2002 ACM SIGMOD international conference on Management of data. – 2002. – С. 570–581.

105.Theories and models for internet quality of service / V.Firoiu, J. Le Boudec, D. Towsley, Z. Zhang. // Proceedings of the IEEE. – 2002. – №9. – С. 1565—1591.

106.Thompson K. Wide-area Internet traffic patterns and characteristics / K. Thompson, G. J. Miller, R. Wilder. // IEEE Network. – 1997. – №11. – С. 10–23.

107.TransLight: a global-scale LambdaGrid for e-science / [T. DeFanti, C. De Laat, J. Mambretti та ін.]. // Communications of the ACM. – 2003. – №11. – С. 34–41.

108.Transport protocols for high performance: Whither TCP / [A. Chien, T. Faber, A. Falk та ін.]. // Communications of the ACM. – 2003. – №46. – С. 42–49.

109.Vacche A. D. Mastering Zabbix / A. D. Vacche, S. K. Lee., 2013. – 358 с.

110.Vascellaro J. E. Google gears down for tougher times / J. E. Vascellaro, S. Morrison. // Wall Street Journal. – 2008. – С. A1.

111.Welzl M. A survey of lower-than-best-effort transport protocols [Електронний ресурс] / M. Welzl, D. Ros // Informational. – 2011. – Режим доступу до ресурсу: https://tools.ietf.org/pdf/rfc6297.pdf.

112.Willinger W. Where Mathematics meets the Internet / W. Willinger, V. Paxson. // Notices of the American Mathematical Society. – 1998. – №45. – С. 961–970.

113.Wu C. H. A New Analytic Framework for Dynamic Mobility Management of PCS Networks / C. H. Wu, H. P. Lin, L. S. Lan. // IEEE Transactions on Mobile Computing. – 2002. – №3. – С. 208–220.

114.Wu X. R. Evaluation of rate-based transport protocols for lambda-grids / X.R. Wu, A. Chien. // High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium. – 2004. – С. 87–96.

115.Xu L. Binary increase congestion control (BIC) for fast long-distance networks / L. Xu, K. Harfoush, I. Rhee. // INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies.. – 2004. – №4. – С. 2514–2524.

116.Yu J. A taxonomy of scientific workflow systems for grid computing / J. Yu, R. Buyya. // ACM Sigmod Record. – 2005. – №34. – С. 44–49.