**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«На правах рукопису»
УДК_____

«До захисту допущено»
Завідувач кафедри
_____ Наталія АУШЕВА
“___”_____2025 р.

# Магістерська дисертація

на здобуття ступеня другого (магістерського) рівня вищої освіти
за освітньою програмою “Цифрові технології в енергетиці”
зі спеціальності 122 “Комп’ютерні науки”

на тему «Розподілена система зберігання файлів із адресацією вмісту»

Виконав: студент _2_ курсу, групи ТР-43мп
Гуменюк Андрій Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник професор каф. ЦТЕ, д.т.н., професор, Сергій ОТРОХ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент професор каф. ІПЗЕ, д.т.н, професор Олег БАРАБАШ _____
(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль асистент Володимир РУДИК _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ — 2025

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти другий (магістерський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-наукова програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

«___»_____2025 р.

**ЗАВДАННЯ**
**на магістерську дисертацію студенту**

*Гуменюку Андрію Олександровичу*
(прізвище, ім’я, по батькові)

1. Тема дисертації: *«Розподілена система зберігання файлів із адресацією вмісту»*
керівник роботи *Отрох Сергій Іванович, д.т.н., професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «03» листопада 2025 р. № 4779-с

2. Термін подання студентом дисертації: 09.12.2025

3. Об’єкт дослідження: процес розподіленого зберігання та доступу до файлів із адресацією вмісту у піринговых мережах.

4. Вихідні дані: мова програмування Golang, сховище ключів-значень LevelDB, оркестратор процесів Systemd, фреймворк для розробки настільних застосунків Tauri.

5. Перелік завдань: 1) провести аналіз існуючих розподілених систем зберігання файлів; 2) дослідити алгоритми маршрутизації та пошуку даних у піринговых мережах; 3) спроєктувати та реалізувати розподілену систему зберігання файлів з адресацією вмісту; 4) впровадити та протестувати розроблену систему; 5) виконати експерементальний аналіз розробленої програмної системи.

6. Орієнтований перелік ілюстративного матеріалу: діаграми, що демонструють принцип роботи використаних алгоритмів, архітектуру системи, функціональні можливості, взаємодію користувача з системою.

7. Орієнтовний перелік публікацій: Гуменюк А.О., Отрох С.І. Проактивна

реплікація даних у піринговых системах з адресацією вмісту. Тези 3-тя Міжнародна науково-практична конференція «Innovations in Science: From Theoretical Foundations to Practical Impact». Антверпен, Бельгія.

8. Дата видачі завдання «16» вересня 2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів підготовки магістерської дисертації | Термін виконання | Примітка |
|---|---|---|---|
| 1 | Вивчення та аналіз задачі | 01.09.25 — 10.09.25 | виконано |
| 2 | Розробка архітектури та загальної структури системи | 11.09.25 — 16.09.25 | виконано |
| 3 | Розробка структур окремих підсистем | 17.09.25 — 30.09.25 | виконано |
| 4 | Програмна реалізація системи | 01.10.25 — 19.10.25 | виконано |
| 5 | Оформлення магістерської дисертації | 22.10.25 — 23.11.25 | виконано |
| 6 | Захист створеного програмного забезпечення | 21.10.2025 | виконано |
| 7 | Передзахист | 25.11.2025 | виконано |
| 8 | Захист | 23.12.2025 | виконано |

Студент _____ **Андрій ГУМЕНЮК**
(підпис) (ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ **Сергій ОТРОХ**
(підпис) (ім’я, ПРІЗВИЩЕ)

# РЕФЕРАТ

Дисертація присвячена розробці та впровадженню розподіленої системи зберігання файлів з адресацією вмісту з поєднанням піринговых мереж для досягнення масштабованого та відмовостійкого рішення для зберігання даних.

Актуальність теми визначена зростанням потреби до масштабування систем для зберігання файлів, яку неможливо повністю досягнути за допомогою традиційних централізованих архітектур. Використання піринговых мереж та адресації файлів за вмістом дозволяє покращити доступність та забезпечити цілісність даних і загальну відмовостійкість масштабованої системи.

Метою роботи є визначення та обґрунтування архітектурних принципів і механізмів для побудови ефективної розподіленої системи зберігання файлів, що забезпечує високу доступність та швидкий доступ до даних за ідентифікаторами вмісту.

Для досягнення мети були визначені наступні задачі:

— провести аналіз існуючих розподілених систем зберігання файлів;

— дослідити алгоритми маршрутизації та пошуку даних у піринговых мережах;

— спроєктувати та реалізувати розподілену систему зберігання файлів з адресацією вмісту;

— виконати експериментальний аналіз розробленої програмної системи.

Об’єктом дослідження є процес розподіленого зберігання та доступу до файлів із адресацією вмісту у піринговых мережах.

Предметом дослідження є методи та алгоритми, що забезпечують адресацію, маршрутизацію, реплікацію та відмовостійкість розподіленої системи зберігання файлів із адресацією вмісту.

Методи дослідження включають аналіз існуючих розподілених файлових систем і алгоритмів маршрутизації в піринговых мережах, моделювання та проєктування програмного забезпечення для побудови архітектури системи, а також експериментальні дослідження для оцінки її ефективності.

Основні положення роботи були апробовані в рамках III Міжнародної науково-практичної конференції «Innovations in Science: From Theoretical Foundations to Practical Impact», 2025, та у статті «Підвищення доступності даних у піринговых системах з адресацією вмісту за рахунок проактивної реплікації», прийнятій до друку в журналі «Systems and Technologies» (перше видання, 2026).

Обсяг дисертації становить 94 сторінки та містить 15 рисунків, 22 таблиці, 24 використаних джерел та 1 додаток. Основний вміст викладено на 88 сторінках.

Ключові слова: РОЗПОДІЛЕНІ СИСТЕМИ, P2P МЕРЕЖІ, DHT, АДРЕСАЦІЯ ВМІСТУ, ДЕЦЕНТРАЛІЗАЦІЯ.

# ABSTRACT

The dissertation is devoted to the development and implementation of a distributed file storage system with content addressing combined with peer-to-peer networks to achieve a scalable and fault-tolerant data storage solution.

The relevance of the topic is determined by the growing need to scale file storage systems, which cannot be fully achieved using traditional centralized architectures. The use of peer-to-peer networks and content-based file addressing improves accessibility and ensures data integrity and overall fault tolerance of a scalable system.

The goal of this work is to define and justify architectural principles and mechanisms for building an effective distributed file storage system that provides high availability and fast access to data by content identifiers.

To achieve this goal, the following tasks were defined:

— analyze existing distributed file storage systems;

— investigate routing and data search algorithms in peer-to-peer networks;

— design and implement a distributed file storage system with content addressing;

— perform experimental analysis of the developed software system.

The object of the study is the process of distributed storage and access to files with content addressing in peer-to-peer networks

The subject of the study is methods and algorithms that provide addressing, routing, replication, and fault tolerance of a distributed file storage system with content addressing.

The research methods include analysis of existing distributed file systems and routing algorithms in peer-to-peer networks, modeling and software design for constructing the system architecture, as well as experimental evaluation of its effectiveness.

The main findings of the work were presented at the 3rd International Scientific and Practical Conference «Innovations in Science: From Theoretical Foundations to Practical Impact» (2025), as well as in the article «Improving Data Availability in

Content-Addressable Peer-to-Peer Systems through Proactive Replication», accepted for publication in the «Systems and Technologies» journal (first edition, 2026).

The thesis consists of 94 pages and contains 15 figures, 22 tables, 24 references, and 1 appendix. The main content is laid out on 88 pages.

Keywords: DISTRIBUTED SYSTEMS, P2P NETWORKS, DHT, CONTENT ADDRESSING, DECENTRALIZATION.

# ЗМІСТ

# СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ACL — список контролю доступу (Access Control List)

API — програмний інтерфейс прикладного програмування (Application Programming Interface)

CADFS — розподілена файлова система з адресацією вмісту (Content-Addressable Distributed File System)

CAS — сховище з адресацією вмісту (Content-Addressable Storage)

CDN — мережа доставки контенту (Content Delivery Network)

CID — ідентифікатор вмісту (Content Identifier)

DAG — спрямований ациклічний граф (Directed Acyclic Graph)

DHT — розподілена хеш-таблиця (Distributed Hash Table)

DFS — розподілена файлова система (Distributed File System)

GC — збирання сміття, механізм автоматичного очищення невикористовуваних даних (Garbage Collection)

LSM-tree — журнально-структуроване дерево зі злиттям (Log-Structured Merge Tree)

NAT — трансляція мережевих адрес (Network Address Translation)

NFT — невзаємозамінний токен (Non-Fungible Token)

P2P — пірингова модель (Peer-to-Peer)

SDK — набір засобів розробника програмного забезпечення (Software Development Kit)

TTL — час життя пакета або повідомлення в мережі (Time To Live)

Web3 — концепція децентралізованого Інтернету на базі блокчейн-технологій

# ВСТУП

Потреба у децентралізації ресурсів існує відколи комп’ютери почали об’єднувати в локальні та глобальні мережі. Розподілений підхід до побудови архітектури дозволяє координувати роботу окремих обчислюваних машин об’єднаних в одну мережу, як єдиною логічною системою. Це забезпечує масштабованість, відмовостійкість і доступність яку неможливо досягнути покладаючись лише на ресурси одного хоста. Такого роду системи забезпечують роботу широкого спектра застосунків — від хмарних і аналітичних платформ і до блокчейн-мереж.

Розподілені системи зберігання файлів(DFS) є одним із найважливіших компонентів сучасного програмного забезпечення. Вони надають змогу користувачам та іншим застосункам об’єднати дані у єдиному просторі, при цьому фізично розподіленому між багатьма вузлами. Розподілення даних та робочого навантаження між декількома або багатьма вузлами дозволяє значно покращити продуктивність та знизити точкове навантаження, проте основною перевагою даних систем є здатність до відмовостійкості. Механізми реплікації та резервації даних дозволяють їм залишатись доступними навіть при виході з ладу окремих вузлів чи навіть цілих підмереж. Класичні реалізації DFS, такі як Google File System та Hadoop Distributed File System, продемонстрували що дані системи займають ключову роль у забезпеченні надійної обробки великих обсягів даних. Хоча й по своїй суті, їх внутрішня реалізація є розподіленою системою, проте на користувацькому рівні це все ще централізовані клієнт-серверні застосунки.

Пірингові мережі, безпосередньо, обмінюються та маршрутизують дані не покладаючись на централізований сервер. Інтеграція такої архітектури з розподіленими файловими системами дає можливість для повної децентралізації та самоорганізації. Орієнтованість такого роду систем на адресацію за вмістом, а не шляхом до файлу, як у традиційних файлових системах, зумовлена бажанням забезпечення легкого способу перевірки даних на цілісність та їх дедублікації.

Система як IPFS(InterPlanetary File System), хоч і є у першу чергу рішенням для маршрутизації, а не зберігання даних, проте вона є гарним прикладом поєднання надійності DFS та повної децентралізації піринговых мереж.

Метою роботи є визначення та обґрунтування архітектурних принципів і механізмів для побудови ефективної розподіленої системи зберігання файлів, що забезпечує високу доступність та швидкий доступ до даних за ідентифікаторами вмісту. Для досягнення зазначеної мети було здійснено розгляд низки як теоретичних, так і практичних питань. По-перше, алгоритми маршрутизації, способи їх оптимізації та проблеми при розгортанні піринговых мереж. По-друге, методи адресації вмісту, та що саме відбувається з файлом коли він потрапляє до CAS. По-третє, буде детально описана реалізація розподіленої системи зберігання файлів із адресацією вмісту, всіх її компонентів, засобів розробки та відображено результати роботи програмного застосунку та обчислювальних експериментів.

Об’єктом дослідження є процес розподіленого зберігання та доступу до файлів із адресацією вмісту у піринговых мережах.

Предметом дослідження є методи та алгоритми, що забезпечують адресацію, маршрутизацію, реплікацію та відмовостійкість розподіленої системи зберігання файлів із адресацією вмісту.

Методи дослідження включають аналіз існуючих розподілених файлових систем, алгоритмів маршрутизації та адресації вмісту. Для побудови архітектури застосовано методи моделювання та проєктування програмного забезпечення. Для надання оцінки розробленого прототипу застосовано метод експериментальних досліджень.

Практичне значення роботи полягає у розробці прототипу розподіленої системи зберігання файлів з адресацією вмісту, що може слугувати основою для подальших досліджень та створенню реальних платформ. Запропоновані архітектурні рішення можуть бути інтегровані в хмарну інфраструктури та для покращення доступності даних, спрощення перевірки цілісності та забезпечення ефективної дедублікації даних.

У першому розділі сформульовано постановку задачі розробки розподіленої системи зберігання файлів з адресацією вмісту, описано основні сфери

застосування, розглянуто існуючі розподілені системи зберігання файлів, а також узагальнено вимоги до масштабованості, доступності, цілісності даних та розглянуто використані засоби розробки.

Другий розділ присвячено аналізу проблеми побудови розподіленої системи зберігання файлів із адресацією вмісту: проаналізовано модель піринговых мереж та розподіленої хеш-таблиці, формалізовано алгоритм Кадемлія та механізмів адресації за вмістом.

У третьому розділі подано опис програмної реалізації системи: представлено архітектуру програмного забезпечення, структуру основних модулів, механізми взаємодії між ними, особливості реалізації DHT, модулів зберігання та обробки даних, а також внутрішні протоколи обміну даними.

Четвертий розділ присвячено роботі користувача з програмною системою: наведено системні вимоги, описано процес розгортання вузлів, використання програм командного рядка та настільного застосунку, послідовність виконання основних операцій із файлами та особливості адміністрування розподіленого сховища.

У п’ятому розділі розроблено стартап-проєкт на основі створеної системи: сформовано бізнес-ідею та ціннісну пропозицію продукту, проведено аналіз цільового ринку та конкурентного середовища, обґрунтовано цінову політику, канали збуту та маркетингові комунікації, а також подано узагальнену оцінку перспектив комерціалізації рішення.

Обсяг дисертації становить 94 сторінки та містить 15 рисунків, 22 таблиці, 24 використаних джерел та 1 додаток. Основний вміст викладено на 88 сторінках.

# 1 ЗАДАЧА РОЗРОБКИ РОЗПОДІЛЕНОЇ СИСТЕМИ ЗБЕРІГАННЯ ФАЙЛІВ З АДРЕСАЦІЄЮ ВМІСТУ

У цьому розділі буде сформульовано постановку задачі, основні вимоги до системи та описано засоби розробки. Також буде надано аналіз існуючих рішень та ключових проблем предметної області.

## 1.1 Постановка прикладної задачі

Розподілена система зберігання файлів з адресацією вмісту являє собою реалізацію вузла пірингової мережі, що використовує розподілену хеш-таблицю із використанням алгоритму Кадемлія як шар для маршрутизації, безпосередньо, вузлів та даних.

Дана система має забезпечувати ефективний та відмовостійкий засіб для пошуку та вилучення контенту без використання централізованих серверів. Окрім того, адресація вузлів та даних має відбуватись за допомогою криптографічних ключів, а не змінюваних локацій. Розв’язання таких питань як, єдина точки відмови та масштабування при приєднанні чи від’єднанні вузла повинні мати вбудовану підтримку.

Потенційними користувачами даної системи є розробники програмного забезпечення, що бажають впровадити дану систему, як спосіб забезпечення даних із незмінною локацією. Системні адміністратори можуть мати на меті об’єднати приватну мережу, у єдину систему для локації даних, у випадках коли їх централізація є недоцільною. А також звичайні користувачі, що прагнуть зберегти місце на своєму жорсткому диску, завдяки можливості розподілення даних між вузлами у мережі.

Кожна реалізація вузла повинна містити TCP сервер, який використовує власний протокол із використанням JSON-формату повідомлень. Даний канал використовується виключно для координації між вузлами та маршрутизації у DHT. Поряд із TCP сервером працює HTTP клієнт для впровадження

уніфікованого інтерфейсу, за допомогою якого відбувається взаємодія між вузлом та різноманітними клієнтськими застосунками. Також впровадження незалежного компонента для фрагментації файлів і їх ефективного збереження як всередині пам’яті процесу, так і на диску.

Застосунок постачається у вигляді сервісу із використанням оркестратора процесів та системних служб systemd, що використовується у більшості дистрибутивів Linux. Разом із сервісом було розроблено програму командного рядка та користувацький інтерфейс, що дозволяють взаємодіяти з вузлом і використовувати його як розподілене сховище файлів, так і сховище ключів-значень.

Для розробки даної системи було виконано наступні завдання:

— розроблено алгоритми ідентифікацію та маршрутизації вузлів і даних у мережі;

— спроєктовано JSON протокол на основі TCP для комунікації вузлів, що реалізує такі види повідомлень, як: PING, STORE, FIND_NODE, FIND_VALUE, FETCH_BLOCK, PUT_BLOCK;

— розроблено системи фрагментації та зберігання файлів, що взаємодіють із мережею вузлів для оголошення постачальників контенту;

— розроблено клієнтські застосунки із використанням як користувацького інтерфейсу, так і командного рядка;

— розроблено сервер ретранслятор для обходу NAT за допомогою ретрансляції запитів та відповідей та визначення публічної IP-адреси вузла;

— проведено тестування для перевірки коректності та надійності роботи системи.

## 1.2 Проблематика предметної області

Розподілені системи зберігання файлів з адресацією вмісту(CADFS), що базуються на піринговых мережах, поєднують ідентифікацію даних, використовуючи криптографічні хеші, та децентралізоване сховище для маршрутизації даних поміж автономними вузлами. Такий підхід представляє

цілий набір технічних та дослідницьких питань стосовно масштабованості, динамічності мережі, доступності даних та безпеки.

### 1.2.1 Доступність даних

Фундаментальною проблемою є підтримка працездатності системи у випадках, коли вузли часто під’єднуються та від’єднуються з мережі. Ця проблема зазвичай вирішується за допомогою використання структурованих топологій піринговых мереж, таких як розподілена хеш-таблиця. Структурованість мережі дозволяє вузлам підтримувати детермінований набір сусідніх вузлів. Типовими стратегіями для підтримки доступності мережі при високому навантаженні є:

— миттєвий пошук сусіднього вузла на заміну при відмові поточного;

— розрахунок затримок запитів під час операцій пошуку;

— надання переваги ближчим вузлам над дальніми[1].

Крім того, надсилання запиту одразу до декількох вузлів паралельно часто також сприяє збереженню високої якості маршрутизації та запобігання помилкам пошуку під час навантажень у мережі.

Розміщення та реплікація даних є ще однією важливою проблемою. Оскільки ідентифікатори вмісту в CADFS походять від криптографічних хеш-функцій, місця зберігання визначаються на рівні маршрутизації, а не за допомогою ієрархій, як, наприклад у файловій системі операційної системи. Це створює ризик нерівномірного розподілу сховищ, виникнення гарячих точок навколо популярного вмісту. Типовим рішенням є децентралізація копій — зберігання реплік на декількох вузлах, ідентифікатори яких знаходяться поблизу хеш-функції вмісту в різних частинах мережі. Системи часто реалізують адаптивну реплікацію, збільшуючи кількість реплік на основі частоти запитів, та кешування вздовж маршруту, що дозволяє тимчасово зберігати часто використовувані дані на проміжних вузлах для зменшення концентрації навантаження та збереження доступності у разі виходу з ладу відповідальних вузлів.

### 1.2.2 Узгодженість даних

Однією з головних переваг CADFS є автоматична дедублікація даних. Той самий вміст зберігається лише один раз оскільки той завжди відповідає одному й тому ж самому хешу. Проте адресація за вмістом робить неможливим операції видалення, оскільки це може призвести до ситуацій коли один фрагмент посилається на вже відсутні дані[2]. На заміну, використовуються збирачі сміття, що рахують посилання фрагментів у різних файлах та періодично видаляють непотрібні дані. Нарешті, хоча й ризики колізій хешів теоретично існують, проте на практиці є близькими до неможливих, при умові використання складних хеш-функцій.

CADFS зазвичай розглядають дані як незмінні(immutable) — після збереження вміст об’єкта ніколи не змінюється. Будь-яке оновлення файлу призводить до створення нової адреси вмісту, а не до його модифікації.

Такий підхід розв’язує деякі проблеми узгодженості даних, наприклад, кеш ніколи не вказує на застарілі дані. Якщо файл змінюється, то отримує новий хеш, а репліка старої версії залишається чинною. незмінність усуває необхідність в інвалідації кешу та складних протоколах узгодженості[3].

Через особливість такого рішення, забезпечення прозорої послідовності даних стає проблемою вищого рівня. CADFS мають впроваджувати версіювання або підміну посилання на попередній об’єкт для підтримки такого функціоналу. Така підміна часто призводить до лише кінцевої узгодженості даних — публікація нового хешу для оновленого вмісту повинна поширитись через мережу, під час чого, інші вузли будуть все ще бачити стару версію. У піринговых мережах така координація може бути доволі повільною або вимагати додаткових механізмів узгодження.

### 1.2.3 Конфіденційність даних

За замовчуванням, CADFS ніяк не приховують дані. Якщо вузол може отримати файл за його ідентифікатором — він може прочитати дані. На практиці, файл можна зашифрувати, а потім використовувати хеш шифрованого тексту для її адресації. Недоліком є те, що шифрування може перешкоджати дедублікації,

оскільки зашифровані ідентичні дані будуть мати однаковий ідентифікатор лише у випадку використання одного й того ключа, що багатокористувацьких сценаріях є неможливим з міркувань безпеки.

Деякі системи розв’язують цю проблему за допомогою конвергентного шифрування, тобто шифрування даних за допомогою ключа, отриманого з самих даних. Завдяки чому ідентичний відкритий текст дає ідентичний зашифрований , що дозволяє дедублікацію, проте має відомі проблеми з безпекою, наприклад, атака з відомим відкритим текстом(known plain-text attack).

Найчастіше такі системи залишають шифрування на рівні додатку. Наприклад, IPFS за замовчуванням не шифрує вміст і не реалізує жодного контролю доступу, але дозволяє користувачам шифрувати дані перед додаванням і навіть рекомендує це робити для конфіденційної інформації. Іншим запропонованим рішенням є використання гібридних приватних IPFS мереж, що фільтрують запити до вузла за списком контролю доступу (ACL) [4].

## 1.3 Сфери застосування

Попри проблеми визначені у пункті 1.2, розподілене зберігання даних з адресацією за змістом успішно застосовується в багатьох сферах.

Міжпланетна файлова система (IPFS) є яскравим прикладом CADFS, що використовується для децентралізованого вилучення вебконтенту, файлів та медіа. Вона використовується для хостингу децентралізованих вебсайтів та медіа, наприклад, багато NFT платформ зберігають зображення на IPFS. Дані залишається доступним, допоки хоча б один вузол їх закріплює, що дозволяє користувачам не покладатися на єдиний сервер.

Менеджери пакунків та реєстри контейнерів використовують адресацію за вмістом для підвищення ефективності та надійності систем. Наприклад, Docker, розраховує хеш для кожного шару образу для ідентифікації спільного. Це дозволяє уникнути повторного завантаження. Іншим прикладом є менеджер пакунків для операційної системи Лінукс Nix, що використовує адресацію за вмістом для дедублікації поширених файлів.

Яскравим прикладом є Git. Кожен файл, дерево каталогів (directory tree), коміт — захешовані та проіндексовані за цим хешем. Це гарантує, що будь-яка зміна вмісту створює новий об’єкт та один і той самий вміст, доданий до двох різних сховищ, матиме однаковий хеш, що дозволяє заощадити простір на хмарних платформах як Github.

Хоча традиційні CDN адресують дані за місцем розташування, можна уявити CDN, що використовує хеші вмісту для забезпечення контенту. Насправді деякі налаштування нових веб-API дозволяють кешування за вмістом, щоб уникнути подвійного завантаження одних і тих самих даних. Такі проєкти, як WebTorrent, використовують BitTorrent протокол для доставлення медіа в браузери, розвантажуючи сервери. Бази даних, як DynamoDB, внутрішньо використовують ідеї розподіленого зберігання даних для забезпечення високої доступності.

## 1.4 Аналіз існуючих рішень розподілених файлових систем

Задача розробки розподіленої системи зберігання файлів потребує попереднього аналізу існуючих рішень. Такий аналіз дозволяє визначити ключові відмінності та архітектурні рішення, які були застосовані на практиці так само як і визначити недоліки та обмеження існуючих підходів.

Одним із найперших та найбільш поширених підходів є мережеві файлові системи на основі клієнт-серверної моделі, зокрема Sun Network File System (NFS). NFS представляє механізм доступу до віддалених файлів для робочих станцій UNIX шляхом інтеграції протоколу віддаленого доступу до файлів в ядро операційної системи та використання віртуальної файлової системи для відображення віддалених каталогів у локальний простір імен[5]. Такий підхід суттєво спрощує спільний доступ до даних у локальних і корпоративних мережах, проте спирається на центральні файлові сервери. Унаслідок чого, масштабування потребує додаткових механізмів реплікації, балансування навантаження та резервування, які не є частиною базової моделі NFS.

Подальший розвиток клієнт-серверної моделі проілюстровано на прикладі файлової системи Andrew (AFS). AFS забезпечує прозорий для розташування глобальний простір імен і використовує агресивне кешування цілих файлів на клієнтській частині у поєднанні з протоколом узгодженості на основі колбеків[6]. Така модель краще масштабується порівняно з NFS, проте все ще залежить від набору надійних централізованих серверів і централізованого адміністрування; це не повністю однорангова система і не призначена для середовищ, де велика кількість вузлів може автономно приєднуватися до мережі та залишати її.

Google File System (GFS) була розроблена як масштабована DFS для додатків, що інтенсивно використовують дані(data-intensive). GFS використовує архітектуру master–chunkserver, де один головний вузол зберігає метадані та координує розміщення великих фрагментів файлів на декількох chunkservers, які, своєю чергою, реплікують фрагменти для забезпечення відмовостійкості[7]. Схожий підхід переймає Hadoop File System(HFS). HFS призначена для надійного зберігання великих наборів даних та використовує NameNode для підтримки простору імен файлової системи та DataNodes для зберігання реплікованих блоків на багатьох серверах[8]. Ці системи забезпечують високу масштабованість і відмовостійкість, але так само передбачають централізоване управління і не мають властивостей повністю децентралізованої мережі.

На відміну від класичних DFS, сучасні системи розподіленого зберігання даних поєднують пірингову мережу разом із використання розподіленої хеш-таблиці та адресацію за вмістом[9].

Яскравим прикладом є Interplanetary File System (IPFS), що реалізує модель сховища блоків із адресацією вмісту. Система поєднує розподілену хеш-таблицю та протокол обміну блоками.

Водночас IPFS задумано як систему загального призначення з власними припущеннями щодо моделі безпеки чи довгоживучості даних, тому вона не надає моделі життєвого циклу даних, перекладаючи відповідальність безпосередньо на користувачів.

## 1.5 Огляд засобів розробки

Для розробки запропонованої системи було обрано набір інструментів та технологій розробки з акцентом на підтримку паралельного виконання, можливості мережевого програмування, криптографії та кросплатформного розгортання. Вибір цих інструментів безпосередньо впливає на продуктивність, надійність та зручність обслуговування системи, тому є важливою частиною загального процесу проєктування.

### 1.5.1 Мова програмування Golang

Для розробки вузла розподіленої системи зберігання файлів з адресацією вмісту було обрано мову програмування Golang. Загалом, Go добре підходить для створення розподілених і мережевих додатків завдяки простій синтаксичній структурі, потужній системі типів та орієнтації на простоту і надійність. Мова була розроблена з урахуванням паралельності та масштабованості, що відповідає вимогам CADFS, де вузли повинні обробляти багато одночасних підключень, фонових завдань та операцій вводу та виводу.

Однією з ключових переваг Go є вбудована модель паралельності, заснована на співпрограмах(Goroutines) і каналах. Goroutines — це невибагливі до ресурсів потоки, що управляються середовищем виконання Go та дозволяють системі запускати тисячі паралельних завдань з мінімальними витратами. Це спрощує розподіл обов’язків, таких як обробка запитів клієнтів, підтримка таблиць маршрутизації, виконання фонової реплікації та обробка вхідних блоків даних, без необхідності використання системних потоків чи зовнішніх фреймворків. Канали забезпечують зручну абстракцію для безпечного обміну даними між паралельними компонентами, що спрощує реалізацію архітектур, керованих подіями, та зменшує ймовірність конфліктів даних, коли декілька співпрограм працюють паралельно.

Стандартна бібліотека охоплює усі необхідні елементи, у тому числі мережеві бібліотеки для взаємодії з протоколами TCP, UDP та HTTP, криптографію, утиліти для серіалізації/десеріалізації даних та інтерфейсів для

роботи із файловою системою. Як результат, переважна частина систем можуть бути розроблені за допомогою вбудованих пакетів. Golang наголошує на чіткості, уникненні невизначеної поведінки і явної обробки помилок. Ці властивості полегшують розуміння та підтримку кодової бази.

Підтримка кросплатформної компіляції та розгортання ще більше обґрунтовують вибір Go. Мова підтримує статичну компіляцію в єдиний бінарний файл для кожної цільової платформи, що спрощує розгортання та розподіл вузлів між різними операційними системами та архітектурами. Ця особливість є доволі важливою, оскільки вузли можуть бути розгорнуті на серверах, настільних комп’ютерах чи хмарній інфраструктурі з мінімальними зусиллями.

Варто зазначити, що Golang використовує збирач сміття, на відміну від більш низькорівневих мов програмування як C/C++, що може спричинити затримки виконання та використанню обсягу пам’яті, недопустимі у системах реального часу або середовищах із дуже обмеженими ресурсами. Проте, в цілому, Golang забезпечує збалансоване поєднання продуктивності та простоти у розробці, що робить його гарною основою для реалізації прототипу запропонованої системи.

### 1.5.2 Сховище LevelDB

У запропонованій системі — LevelDB використовується як механізм зберігання ключів-значень для індексації закріплень і фрагментів файлів. LevelDB зберігає впорядковані записи ключ-значення, що робить його зручним для зберігання метаданих. У контексті розподіленої системи зберігання файлів з адресацією вмісту — це дозволяє швидко перевірити, чи є певні блоки локально, які з них закріплені і як вони пов’язані з об’єктами вищого рівня.

Внутрішньо, LevelDB використовує журнально-структуроване дерево зі злиттям(LSM-tree). Всі записи спочатку додаються до журналу попереднього запису(write-ahead log), а лише потім періодично записуються в сортовані таблиці на жорсткий диск[10].

Використання LevelDB як вбудованої бібліотеки дозволяє уникнути складності обслуговування сервера бази даних. Зберігання даних є повністю

локальним для кожного вузла, що спрощує розгортання та зменшує специфічні до платформи залежності. З погляду реалізації, бібліотека для Go надає відносно простий API у вигляді операцій читання, запису, видалення та групових операцій(bulk operations). Хоча й запис даних всередині LevelDB виконуються послідовно, проте читання може бути паралельним завдяки знімкам(snapshots) та ітераторам.

### 1.5.3 Оркестратор сервісів Systemd

Systemd — це стандартна система ініціалізації та менеджменту сервісів та служб у більшості сучасних дистрибутивів Linux. Вона координує ініціалізацію системи та керує довготривалими процесами.

У запропонованій системі Systemd використовується для управління життєвим циклом вузла, забезпечуючи роботу у вигляді фонової служби. Завдяки інтеграції із systemd вузол може запускатися автоматично під час завантаження, перезапускатися в разі збою та керуватися без необхідності ручного втручання.

Systemd надає можливість налаштування сервісних модулів, які визначають, як запускається служба, за яких умов вона повинна перезавантажуватись та які змінні середовища вона використовує. Завдяки вбудованим механізмам автоматичного перезапуску, обмеження ресурсів та сторожових таймерів(watchdog timers) — systemd гарантує, що тимчасові збої або несподівані переривання не вплинуть на роботу довготривалих фонових служб.

Ще однією перевагою є інтегроване ведення журналу за допомогою journalctl, що спрощує збір діагностичних даних, відстеження помилок і моніторинг поведінки системи без використання додаткових інструментів. Можливість Systemd керувати службами на рівні користувача та на рівні системи дозволяє працювати з відповідними правами доступу.

### 1.5.4 Фреймворк Tauri

Фреймворк Tauri є кросплатформним засобом розробки настільних застосунків, що поєднує вебінтерфейс із нативним бекендом, реалізованим переважно мовою Rust. Графічний інтерфейс відображається у вбудованому

WebView операційної системи, а нативна частина відповідає за керування вікнами, доступ до файлової системи, мережеві операції, криптографічні функції та інші системні ресурси. На відміну від фреймворків, які постачають повноцінний браузерний рушій разом із застосунком, використання вбудованого WebView дає змогу істотно зменшити розмір виконуваних файлів.

Архітектура Tauri орієнтована на мінімізацію прав та чітке розмежування відповідальності між інтерфейсом та бекендом. Взаємодія між ними здійснюється через систему явно визначених команд (commands), що викликаються з інтерфейсу через канал міжпроцесної комунікації(inter-process communication). Доступ до файлової системи, мережі, системних діалогових вікон та інших критичних ресурсів визначається у маніфесті застосунку, що дозволяє обмежити кількість потенційних вразливостей.

## Висновки до розділу 1

У даному розділі було сформовано проблематику та визначено контекст, у межах якого розглядається тема роботи. Описано, чому розподілені файлові системи з адресацією вмісту на базі піринговых мереж є актуальним напрямом, а також окреслено основні вимоги до таких систем: масштабованість, відмовостійкість, цілісність даних, ефективна маршрутизація та самоорганізація в умовах динамічної мережі. Проаналізовано ключові проблеми предметної області та проведено аналіз існуючих рішень.

Окремо розглянуто інструменти та технології, обрані для реалізації системи та обґрунтовано їх доцільність з огляду на визначені вимоги.

# 2 АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ РОЗПОДІЛЕНОЇ СИСТЕМИ ЗБЕРІГАННЯ ФАЙЛІВ З АДРЕСАЦІЄЮ ВМІСТУ

У даному розділі буде розглянуто розподілені файлові системи з адресацією вмісту ширшому контексті розподілених систем та піринговых мереж. Окрім того, будуть визначені такі поняття як контентна адресація, розподілені хеш-таблиці, спрямований ациклічний граф Меркла(Merkle DAG) та проілюстровано, як ці компоненти взаємодіють, щоб забезпечити децентралізоване зберігання, ідентифікацію, цілісність та маршрутизацію.

## 2.1 Пірингові мережі

Пірингові мережі — це підвид розподілених систем, що позбуваються централізованої маршрутизації за допомогою впровадження групи вузлів, що виконують роль як клієнтів, так і серверів та працюють спільно для впровадження координації, перевірки та пошуку даних.

Власне, всі пірингові мережі являються розподіленими, проте не всі розподілені системи мають за основу пірингову архітектуру. До прикладу, база даних може використовувати розподілений дизайн внутрішньо, але мати зовнішніх користувачів які є лише клієнтами.

Характеристики піринговых мереж можна поділити по топологіях, мірі децентралізації та механізмам маршрутизації.

### 2.1.1 Структуровані та неструктуровані пірингові мережі

Неструктуровані топології можуть характеризуватись графом довільної форми. При доставленні повідомлень, вузли покладаються лише на вибірку прилеглих чи сусідніх.

Структура графа, сформованого неструктурованою топологією, може бути порівняна зі структурою випадкових графів, що демонструють феномен маленького світу та інших соціальних мереж[11].

Повідомлення поширюються по графу із застосуванням таких механізмів як TTL(time to live), випадкових блукань та випадкового обміну інформацією(gossip). Такого роду пірингові мережі віддають перевагу простоті та стійкості до відмов. Вузли можуть приєднуватись та від’єднуватись з мережі, при цьому мінімально координуючи свої дії з іншими, оскільки не існує набору «відповідальних» вузлів без яких мережа б обвалилась.

Водночас фактор випадковості означає, що вартість пошуку може бути доволі великою. Це проблема зазвичай вирішується кешуванням влучань впродовж шляху.

У структурованих мережах зв’язки між вузлами відповідають чіткій топології — кільце, сітка, дерево тощо. Як можна побачити на рисунку 2.1, у топології сітка кожен вузол має 4 сусідніх вузли, що забезпечує багато альтернативних шляхів, проте її стає важче підтримувати з ростом. Своєю чергою, кільце з’єднує кожен вузол рівно з одним сусідом у кожному з напрямків.

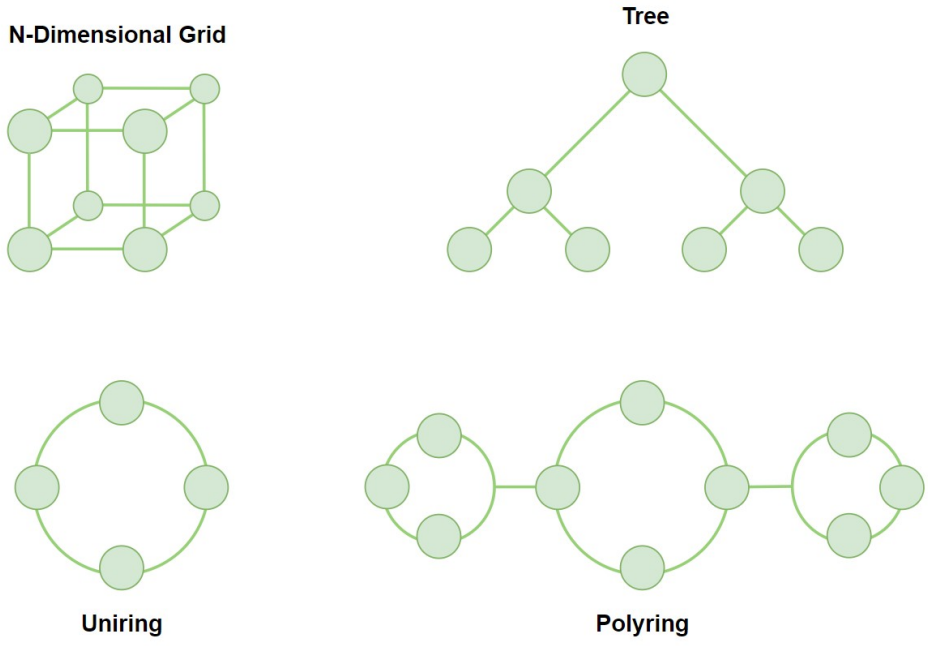


Рисунок 2.1 — Топології структурованих піринговых мереж[12]

Визначена структура організована відповідно до математичних правил чи алгоритмів, що забезпечують детерміновану маршрутизацію та пошук даних, навіть при під’єднанні та від’єднанні вузлів.

Більшість структурованих топологій користуються маршрутизацією на основі ключів які пов’язані з адресами в адресному просторі. Таким чином, що найближчі до адреси вузли зберігають значення відповідних ключів, а алгоритм маршрутизації обробляє ключі як адреси[11].

### 2.1.2 Повністю децентралізовані та гібридні пірингові мережі

В повністю децентралізованих піринговых мережах, всі її учасники мають однакові можливості. Кожен вузол може ініціювати запит, передавати дані чи повідомлення та допомагати іншим вузлам у маршрутизації.

Не існує залежності від координаторів, трекерів чи ретрансляторів. Такого роду мережа надає високу відмовостійкість, оскільки немає єдиної точки відмови, а відтік вузлів з мережі впливає лише на її локалізовану частину. В ту ж чергу повільні чи непостійні вузли можуть погіршити продуктивність системи, збільшуючи затримку у ланцюгу запитів. Без явної політики вилучення проблемних вузлів, частина мережі може залишитись повністю відірваною від інших.

На практиці повністю децентралізованих пірингових систем майже не існує, оскільки більшість вузлів знаходяться за NAT або фаєрволом. Для розв’язування цієї проблеми застосовуються техніки по типу TCP/UDP hole punching, що дозволяють прокласти пряме з’єднання без переадресації портів і ручного налаштування шлюзу.

Поміж цього, навіть повністю децентралізовані мережі майже завжди починають як гібридні, доки не досягнуть певної точки ентропії. Так, поширеною практикою є створення низки надійних публічних вузлів для підтримки працездатності мережі. На рисунку 2.2 показано, як гібридна модель пропонує двошарову структуру, що складається із супервузлів, що керують шаром звичайних, для підтримки рівня доступності у мережі [13].

Як було зазначено раніше, не всі розподілені системи є децентралізованими. Часто вони є повністю децентралізованими лише у своїй внутрішній реалізації та представляються зовнішньому користувачеві як типові клієнт-серверні застосунки.

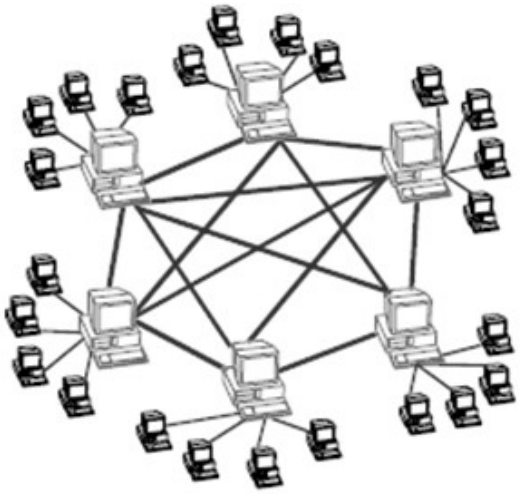

Рисунок 2.2 — Гібридна модель піринговоі мережі[13]

Це робиться для уникнення жорсткої залежності від надійності невідомої сторони та зменшення порогу входження при їх користуванні. До прикладу для IPFS існують хмарні сервіси шлюзи(Gateway), що умовно дозволяють брати участь у піринговій мережі, при цьому, безпосередньо, не розгортаючи власноруч вузли.

### 2.1.3 Проблеми мережевої взаємодії у пірингових мережах

Практичне застосування пірингових мереж у загальнодоступному інтернеті ускладнюється частою неможливістю прямої комунікації двох хостів. Основними причинами цього є Network Address Translation (NAT) та фаєрволи. NAT приховує декілька приватних машин за одним публічним IP-адресою і, як правило, дозволяє вхідний трафік лише як відповідь на вихідний. У результаті значна частина вузлів не має публічної адреси й не може приймати вхідні з’єднання напряму, що

зменшує зв’язність мережі, погіршує маршрутизацію та створює залежність від вузлів із відкритими портами.

Щоб подолати NAT, P2P системи зазвичай намагаються встановити пряме з’єднання, а потім переходять до методів делегації. Найпоширенішим підходом є метод пробивання отворів(hole punching).

Основна ідея полягає у тому, щоб обидва вузли одночасно надіслали пакети даних на публічні IP адреси один одного, що провокує NAT зробити тимчасовий запис. Оскільки кожен NAT приймає вхідні пакети з адрес, що відповідають адресі іншої сторони, це дає змогу встановити прямий канал. На рисунку 2.3 зображено кроки пробивання отвору у NAT для встановлення прямого каналу:

1) хост A надсилає вихідний пакет на публічний хост S. NAT A створює тимчасовий запис і призначає A публічну адресу 134.106.1.1:4000;

2) хост B повторює дію та отримує публічну адресу 134.106.1.1:5000;

3) дізнавшись публічну адресу A, B надсилає пакет на 134.106.1.1:4000. NAT A розглядає це як непроханий пакет, тому відкидає;

4) хост S відповідає B, вказавши публічну адресу A (134.106.1.1:4000), і сигналізує B про початок спроб;

5) хост S повторює дію для A;

6) хост A надсилає пакети на 134.106.2.2:5000.

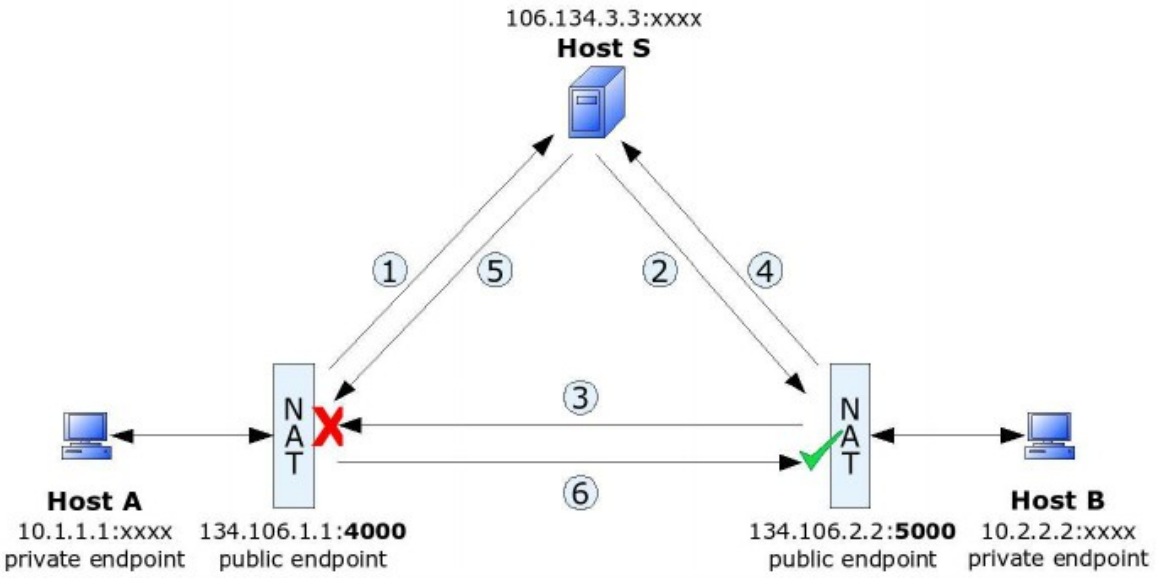


Рисунок 2.3 — Встановлення прямого каналу за допомогою hole punching[14]

Вихідний трафік відкриває NAT A у напрямку B. Як тільки B отримує пакет A та відповідає тим самим шляхом, то NAT A приймає його, оскільки вихідний запит створив необхідний запис, що дозволяє встановити прямий канал між двома хостами.

Ще однією практичною проблемою є початкове розгортання вузла у мережу. Новий вузол не має жодних знань про наявні маршрути, тому більшість систем використовують вузли завантаження(bootstrap) або відомі списки вузлів для отримання початкових сусідів. Після приєднання, вузли розширюють своє поле бачення шляхом обміну інформації та постійного оновлення набору сусідів. Початкове завантаження не централізує маршрутизацію повністю, проте додає стартову залежність від централізованої інфраструктури.

Нарешті, пірингові мережі змушені пристосовуватись до неоднорідних та нестабільних ланок. Кожен вузол у мережі відрізняються за пропускною здатністю, безперервним часом роботи та географічним розташуванням. Щоб зменшити вплив на якість маршрутизації, вибір вузлів часто враховує затримку, віддаючи перевагу швидшим сусідам, а маршрутна інформація утримує надлишкові шляхи для роботи навіть під час тимчасових збоїв.

## 2.2 Розподілена хеш-таблиця та алгоритм Кадемлія

Найпоширенішим способом організування піринговых мереж є розподілена хеш-таблиця. Подібно до хеш-таблиці DHT надає послугу пошуку за парами ключ-значення, де кожен учасник мережі може дістати значення за унікальним ключем. Проте, це може призвести до більших накладних витрат, ніж неструктуровані пірингові мережі[15]

Як ключам, так і вузлам призначаються унікальні ідентифікатори. Кожен вузол володіє власною частиною простору ключів таким чином, що при його пошуку можна легко визначити хто саме ймовірно його зберігає. Зазвичай для уніфікації ідентифікаторів вузлів та ключів використовуються різноманітні хеш-функції.

### 2.2.1 Формальна модель розподіленої хеш-таблиці

Розподілена хеш-таблиця — це структурована пірингова мережа, яка пропонує детермінований пошук значень за ключами. Кожен елемент даних представлений ключем, і мережа маршрутизує запит на цей ключ до вузлів, відповідальних за зберігання відповідного значення. У наведеній нижче формальній моделі викладено спільні припущення та цілі, на яких базуються алгоритми маршрутизації DHT.

DHT припускає спільний, суто логічний простір ідентифікаторів, в якому відображаються як вузли, так і ключі даних. Простір визначається як набір бітових рядків фіксованої довжини:

$$I = \{0,1\}^m,$$

де I — це ідентифікатор у просторі ключів,

m — це довжина ідентифікатора ключа у бітах.

Адреса вузлів та даних у просторі розраховується за допомогою хеш-функції, що можна представити як:

$$id(v) = H(S_v),$$

де v — вузол або об’єкт даних,

Sv — випадково обране значення для вузла або вміст об’єкта даних,

H — хеш-функція.

Модель базується на припущенні про рівномірне хешування, тобто вихідні дані функції H рівномірно розподілені по I, тому ідентифікатори вузлів і ключі об’єктів даних поводяться як випадкові точки в єдиному просторі. Це припущення є основою очікування, що області відповідальності мають однаковий розмір, що дозволяє розподілити навантаження у мережі.

Відповідальність за ключ у просторі визначається за допомогою метрики відстані між ідентифікаторами. Багато реалізацій побудовані на ідеї циклічного впорядкування ідентифікатора I, де ключ зберігається у його наступника в кільці, що забезпечує стабільність та доступність мережі в динамічних умовах, оскільки при приєднанні або виході учасників переміщуються лише сусідні ключі.

Маршрутизація в DHT виконується через віртуальну мережу, побудовану поверх фізичної. Кожен вузол підтримує лише невелику таблицю маршрутизації,

що містить вибраних сусідів, а не повну інформацію про всіх учасників. Конкретні правила вибору сусідів відрізняють конкретні реалізації DHT, але формальні вимоги однакові: локальний стан повинен залишатися обмеженим, при цьому забезпечувати глобальну доступність.

Нарешті, основною властивістю DHT є припущення, що при масштабуванні мережі як очікувана кількість запитів для операцій пошуку, так і розмір таблиці маршрутизації зростає логарифмічно до розміру мережі:

$$L(N)=O(logN), N(v)=O(logN),$$

де L(N) — очікувана кількість стрибків між вузлами для операції пошуку,

N(v) — розмір таблиці маршрутизації,

N — кількість вузлів у мережі.

### 2.2.2 Алгоритм Кадемлія

Кадемлія організовує вузли та ключі у великі ідентифікатори, часто це 160-бітне ціле число. Використання великих ідентифікаторів зумовлене бажанням уникнення колізій без попереднього узгодження з іншими вузлами у мережі.

Фактично, даний алгоритм розглядає вузли як листя в бінарному дереві, де кожна позиція вузла розглядається як унікальний префікс ідентифікатора. На рисунку 2.2 зображено позицію вузла з префіксом 0011[16].

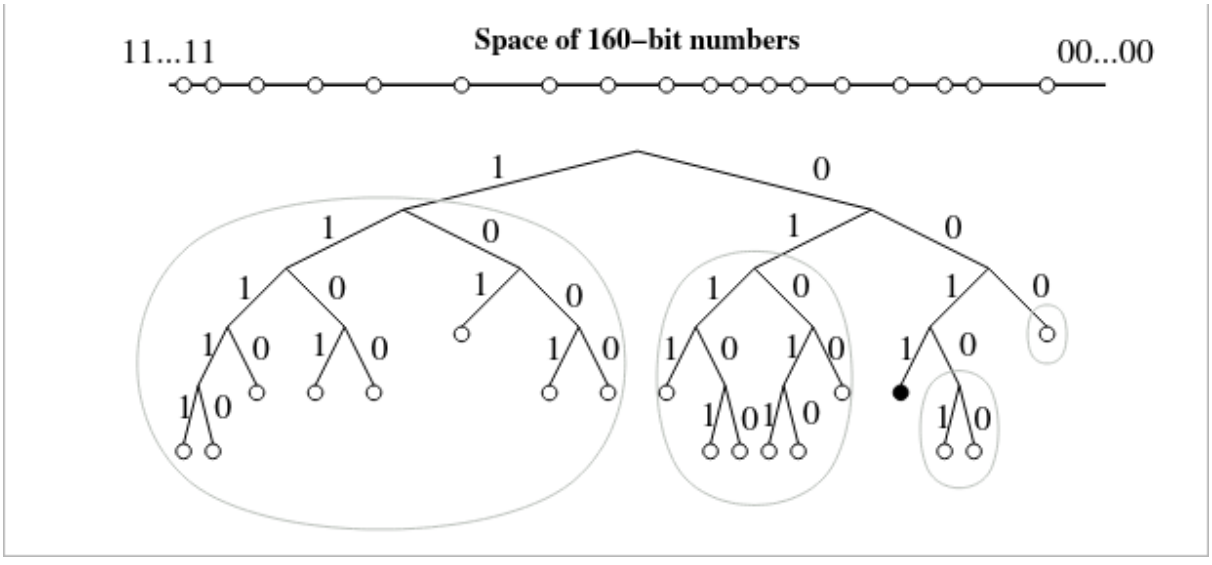


Рисунок 2.2 — Бінарне дерево Кадемлія

Для кожного вузла, бінарне дерево поділяється на серію послідовно нижчих піддерев, які не містять цього вузла. Найвище піддерево складається з половини бінарного дерева, що не містить вузол. Наступне — з половини решти дерева. У даному прикладі піддерева, що не містять вузол 0011 обведені колами та складаються з вузлів із префіксами 1, 01, 000, 0010 відповідно[16].

Параметр відстані обчислюється побітовою операцією XOR. XOR є доволі влучною та дуже дешевою в обчисленні операціє.. Як можна помітити з таблиці 2.1, операція XOR має всі необхідні якості: відстань до самого себе рівна нулю та відстань від A до B та B до A є симетричною.

Таблиця 2.1 — Таблиця істинності XOR

| A | B | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Хоча таке визначення відстані ігнорує фізичну затримку, проте типові реалізації віддають перевагу швидшим сусіднім вузлам, коли вибір є рівноцінним.

Кожен вузол зберігає компактну таблицю маршрутизації, що складається з так званих K-сегментів — списків вузлів згрупованих по відстані, тобто кількості бітів, що збіглися в ідентифікаторах двох вузлів. Важливо зазначити, що маршрутна таблиця являє собою локалізовану інтерпретацію кожного вузла і відстані, відповідно, розраховані не відносно нульового ідентифікатора, а власного вузла.

Таблиця маршрутизації являє собою бінарне дерево, листя якого — це K-сегменти. Кожен K-сегмент включає вузли з певним спільним префіксом. Префіксом є позиція у бінарному дереві. Отже, кожен сегмент покриває певний інтервал площини ідентифікаторів, а K-сегментів покривають всю площину[16].

Коли K-Сегментів заповнюються контактами, сегменти що покривають ідентифікатори ближчі до власного вузла поділяють на менші, щоб дати вузлу більше деталей ближче до себе.

На рисунку 2.3 зображено маршрутну таблицю вузла у 160-бітному просторі ключів. Як можна помітити, ліва частина показує широкі сегменти для далеких вузлів. Праворуч — сегменти вузлів, що ближче до власного ідентифікатора стають вужчими, і дерево розгалужується частіше.

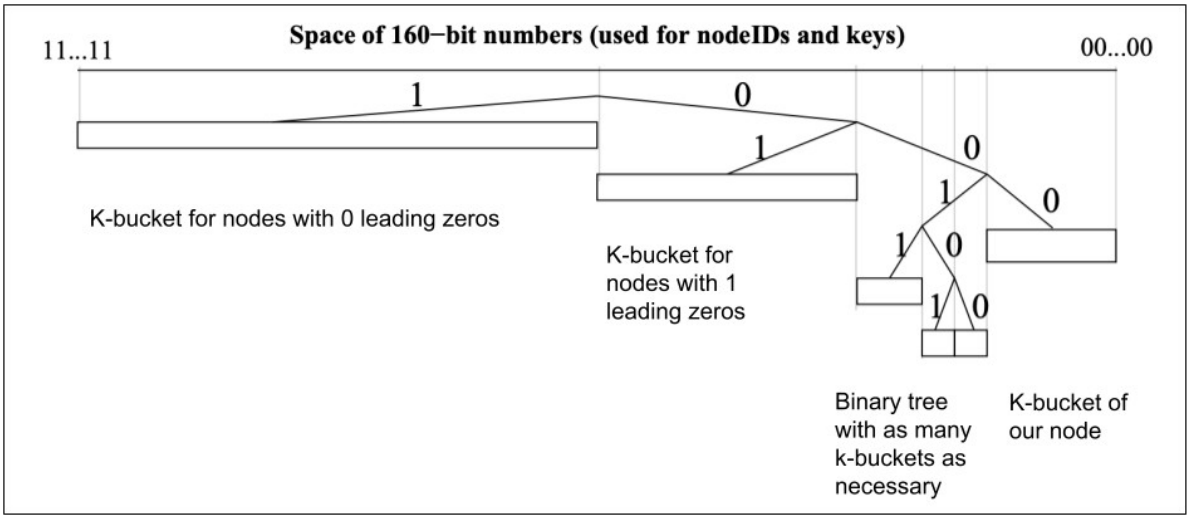


Рисунок 2.3 — Маршрутна таблиця вузла[16]

Тобто коли сегмент, що охоплює область біля власного вузла, переповнюється, він поділяється на два менші за наступним бітом префікса — умовно «0» та «1». Контакти з ідентифікатором, що падають у першу половину, йдуть у лівий сегмент, інші — у правий. Якщо новий локалізований сегмент знову переповнюється, він поділяється ще раз. Так утворюється бінарне дерево сегментів із дедалі тоншим поділом поблизу власного вузла.

Стратегії пошуку в Кадемлії прагнуть досягнути вузлів, найближчих до цільового ключа.

Цільовим підходом є ітеративний пошук. Вузол, що робить запит — контролює процес, починаючи з найближчих до цілі відомих контактів у таблиці маршрутизації. Частина цих контактів паралельно опитується, де кожна відповідь надає або шукане значення, або список вузлів ще ближчих до шуканого ключа.

Далі нові контакти, що отримані з попередніх запитів об’єднуються, сортуються за відстанню та дедублікуються з початковим списком. З них береться K найкращих, де K — число сегментів у таблиці маршрутизації і далі процес знову повторюється.

Рекурсивний підхід, своєю чергою передає контроль процесу до наступного вузла, доки не буде досягнуто необхідного регіону у просторі ключів. Рекурсивне посилання може значно зменшити кількість надісланих ініціатором запиту повідомлень, проте ускладнює обробку помилок та можливості ефективно оновлювати контактну інформацію у таблиці маршрутизації.

Припинення пошуку вузла відбувається коли ініціатор зробив запити до K найближчих контактів і жоден не повернув у відповіді список наступних. Для пошуку значення термінація відбувається як тільки буде отримано достатню кількість дійсних запитів, хоча й у класичній реалізації Кадемлії пошук зупиняється при першому ж знайденому записі. Вздовж ланцюга запитів, вузли кешують дані та найближчі контакти у наступних. Локальний кеш також є допустимим.

### 2.2.3 Інші реалізації розподіленої хеш-таблиці

Хоча Кадемлія є найпоширенішим алгоритмом, існують й інші реалізації DHT, що розв’язувати одну й ту саму проблему — пошук вузлів відповідних цільовому ключу, з використанням різних визначень таблиці маршрутизації та відстані. З переліку можна визначити наступні: Корд, Пестрі та Тапестрі

Корд відображає ідентифікатори вузлів і ключів у кільці. Кожен ключ стає відповідальністю вузла з ідентифікатором рівним, або наступним до нього. Кожен вузол зберігає вказівники на ключі наступного та попереднього для збереження цілісності кільця. Вузол підтримує таблицю стрибків, кожен з яких знаходиться на $2^i$ відстані. Це дозволяє здійснювати пошук щонайбільше за log N стрибків. На відміну від Кадемлії, таблиця маршрутизації Корд не сегментується, що робить важчим провадження механізмів вибору сусідів з урахуванням фізичної затримки.

Пестрі відображає ідентифікатори у base-b(наприклад base-16) та виконує маршрутизацію за допомогою збільшення спільного префіксу з цільовим ключем

на кожному стрибку. Кожен вузол підтримує таблицю маршрутизації організовану в ряди по спільному префіксу. Стрибки здійснюються по принципу спільного префіксу з ключем. Хоча префіксні ряди й забезпечують явність «сусідства» вузлів, розмір таблиці росте зі збільшенням обраного параметра b, що встановлює складність алгоритму пошуку як $O(\log_b N)$.

Тапестрі поділяє префіксний підхід Пестрі, проте робить акцент на розташуванні ключів: під час пошуку в мережі, вузли кешують зворотні вказники, щоб подальші запити займали коротший час, рухаючись по більш інформованому шляху.

## 2.3 Сховище з адресацією вмісту

Сховище з адресацією вмісту ідентифікує шлях до даних за їх вмістом, а не за місцем їхнього розташування. CADFS(Content-addresable distributed file system), складається з 4 основних компонентів: ідентифікатори вмісту(CID), блоки, Меркл граф, що пов’язує ці блоки та записи провайдерів у DHT, що вказують на вузли які мають можливість віддавати відповідні блоки. Разом вони забезпечують цілісність, дедублікацію та розподіленість розташування.

### 2.3.1 Ідентифікатори контенту та блоки

В системах адресації вмісту, кожен збережений фрагмент даних ідентифікується тим, чим він є, а не його фактичним розташуванням. Фактично CID є результатом хеш-функції над даними на які він вказує, тобто він є похідною від даних, що дозволяє гарантувати цілісність файлів без додаткових дій, адже зміна навіть одного біту також змінює й ідентифікатор.

Власне, файли зберігаються у вигляді блоків чи фрагментів, для кожного з яких розраховується CID. Розбиття файлу на блоки надає наступні переваги:

— дедублікація, тобто ідентичні фрагменти в різних файлах будуть мати однакові ідентифікатори;

— вибіркове вилучення, тобто клієнти завантажують лише відсутні блоки замість всього файлу;

— паралелізм, тобто блоки можуть завантажуватись паралельно з декількох вузлів, тим самим покращуючи загальну продуктивність системи.

### 2.3.2 Меркл граф

Нарешті сформовані блоки організовуються у направлений ациклічний граф, де листя графу — блоки даних, вузли — штучні блоки, що зберігають список ідентифікаторів блоків даних та опціональні метадані, корінь — блок із метаданими безпосереднього файлу.

Меркл граф є дуже схожим до Меркл дерево — структури даних, що широко використовується у блокчейн системах. Єдиною різницею є відсутність вимог до збалансованості, тобто можливості вузла мати декілька батьків[17].

Криптографічним елементом є те, що контент кожного вузла це хеш його дочірніх вузлів, тобто A = Hash(B) -> B = Hash(C) -> C. Власне, одностороння властивість хеш-функцій дозволяє забезпечити відсутність циклів у графі[17].

На рисунку 2.4 зображено Меркл граф, можна помітити що він складається з 4 блоків даних.

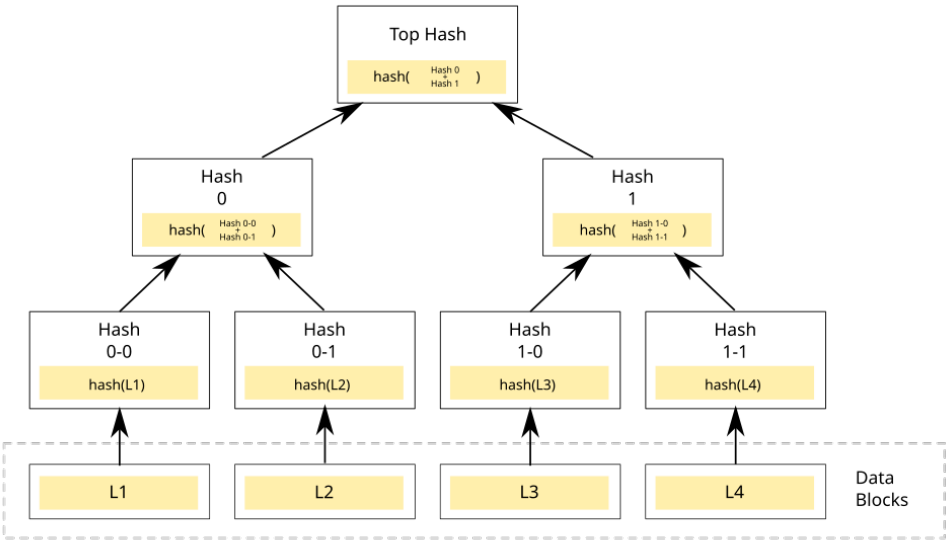


Рисунок 2.4 — Структура фрагментованих файлів у Меркл графі[18]

Кожен блок дає унікальні хеші, що комбінуються у вузли, що комбінуються у кореневий хеш. Власне кореневий хеш і є ідентифікатором вмісту файлу.

### 2.3.3 Записи провайдерів у DHT

Записи провайдерів — це записи, що зберігаються у DHT як пара ключ-значення, де ключем — є CID блока, а значення — ідентифікатор та адреса провайдера(вузла).

Коли вузол зберігає до локального сховища блок та оголошує для сусідніх вузлів у мережі, що він за даним CID зберігається безпосередньо у нього. На перший погляд, може здатись, що такий підхід створить проблему із повторенням ключів у DHT, проте згадуючи властивість локальності таблиць маршрутизації, ближчі вузли будуть звертатись лише до найближчих провайдерів блоків, ігноруючи тих хто знаходиться в іншому кінці мережі. Окрім того, це дозволяє отримувати декілька вузлів для завантаження блоку, як альтернативний ресурс при можливих відмовах у мережі. Підсумовуючи, потік завантаження файлу з CADFS виглядає наступним чином:

1) пошук провайдерів, тобто запит до DHT за CID;

2. завантаження блоків, тобто запит до вузла на отримання необхідних блоків, за CID та перевірка на присутність у локальному сховищі;

3) складання файлу, тобто реконструкція файлу шляхом проходження графу від кореня із припустимим дозавантаженням відсутніх локально блоків, виконуючи пункти 1 та 2;

4) кешування блоків локально для прискорення майбутніх запитів.

## Висновки до розділу 2

У розділі 2 було сформовано теоретичний фундамент, на якому ґрунтується подальше проєктування та програмна реалізація розподіленої файлової системи з адресацією вмісту. Насамперед було проаналізовано основні класи піринговых мереж, з особливим акцентом на структуровані топології, їхні властивості маршрутизації та чутливість до динаміки і неоднорідності мережі. На основі чого було введено формальну модель розподіленої хеш-таблиці як механізму відображення простору ключів на множину вузлів у мережі та також детально розглянуто алгоритм Кадемлія. Окрему увагу було приділено метриці відстані

XOR як основі для впорядкування простору ідентифікаторів, структурі маршрутної таблиці на основі k-buckets та ітеративному пошуку вузла, що дозволяє досягати логарифмічної складності.

Спираючись на модель DHT, було розглянуто сховище з адресацією вмісту як альтернативу традиційним підходам до ідентифікації даних за місцем розташування. Описано поняття ідентифікаторів вмісту як криптографічних хешів, структурування даних у вигляді блоків та організацію цих блоків у граф. Було показано, як файл розбивається на окремі блоки, які зв’язуються у вузли , формуючи цілісну структуру.

Окремо було розглянуто роль записів провайдерів у DHT як механізму, що пов’язує розподілену хеш-таблицю та сховище блоків. Сукупність розглянутих понять і моделей обґрунтовує вибір пірингової мережі на основі Кадемлія в поєднанні з адресацією вмісту, організованою у формі графа, як ядра архітектури розробленої програмної системи.

# 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Цей розділ подає опис практичної реалізації розподіленої системи зберігання файлів з адресацією вмісту. Надалі буде описано структуру вузла, його основні компоненти та їх взаємодія. Окрім того, буде надано модель взаємодії користувача з системою.

## 3.1 Архітектура програмної системи

Розроблена система організована як модульна багаторівнева архітектура навколо реалізації автономного вузла піринговой мережі. Кожен вузол поєднує три основні функції: інтеграція у пірингову мережу, управління локальними даними та надання API для клієнтських застосунків.

На рисунку 3.1 зображено діаграму компонентів та як можна помітити архітектура розроблена таким чином, що визначені функції розділені по окремих підсистемах з чітко визначеними обов’язками. При чому, під час процесу завантаження та вивантаження файлів усі модулі залучені та співпрацюють один з одним.

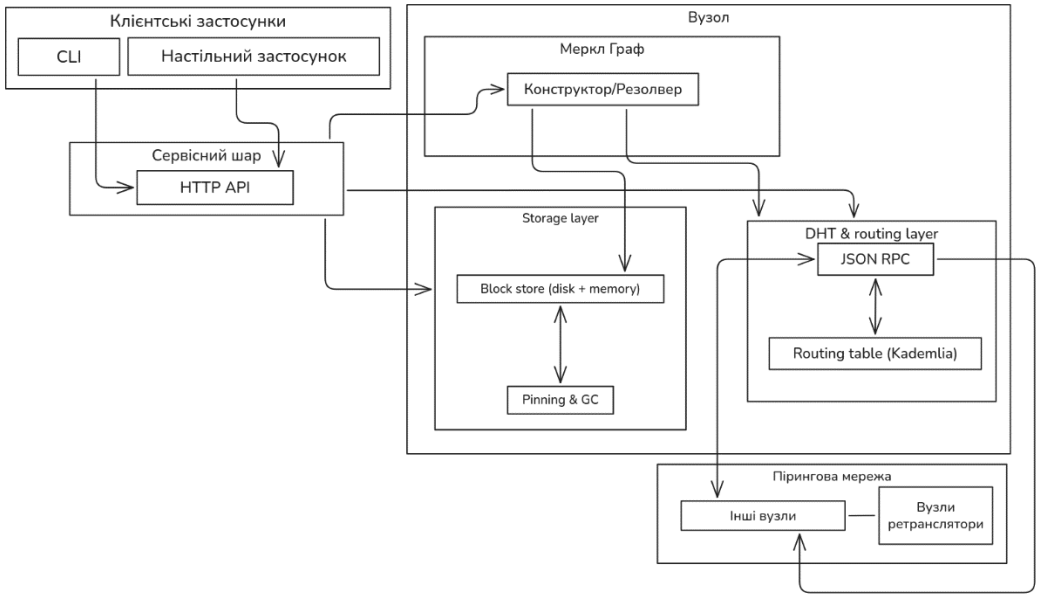


Рисунок 3.1 — Діаграма компонентів розробленої системи

Після старту вузла операційна система запускає фонову програму peerdrived.service за допомогою systemd, яка ініціалізує зовнішній HTTP інтерфейс для комунікації з клієнтськими застосунками та основні компоненти вузла, а саме: заповнює маршрутну таблицю початковими вузлами, переконується що всі конфігураційні файли на місці, реєструє фонові задачі для обслуговування вузла, починає TCP сервер та, за потреби, під’єднується до сервера ретранслятора.

HTTP API виступає зовнішнім інтерфейсом для комунікації із вузлом. HTTP сервер обробляє запит, перетворює його на внутрішню команду і делегує виконання сервісному шару.

## 3.2 Реалізація DHT

Модуль DHT забезпечує логічну основу, для маршрутизації та відповідальний за інтеграцію кожного вузла у пірингову мережу та забезпечення примітивів для комунікації від яких залежить функціональність усіх інших компонентів. До його відповідальностей відносяться наступні задачі:

— підтримка та оновлення маршрутної таблиці;

— пошук вузлів та пар ключів-значень;

— управління життєвим циклом записів;

— фонове обслуговування маршрутної таблиці.

На додаток до маршрутизації, даний модуль визначає RPC протокол для оголошення записів провайдерів, зберігання та завантаження блоків.

З погляду компонентів відповідальних за сховище даних — даний модуль надає загальний інтерфейс для пошуку провайдерів по заданому ключу та вивантаження блоків з інших вузлів, при цьому приховуючи деталі мережевої комунікації та маршрутизації.

Кожному вузлу у мережі відповідає 256-бітний ідентифікатор, який генерується випадково за допомогою криптографічної функції для забезпечення рівномірного розподілу усіх ключів.

Ідентифікатори для пар ключ-значення створюються за допомогою застосування хеш-функції SHA-256 над вхідними байтами. Таким чином усі

ідентифікатори у мережі мають однаковий формат та надалі можуть порівнюватись за відстанню.

Відстань між двома ідентифікаторами визначена за допомогою побітової операції XOR та інтерпретована як ціле число у порядку старшого байта(Big-endian). Ця операція використовується кожен раз коли необхідно вирішити який із двох контактів ближче до заданого цільового ключа.

На кожному вузлі підтримується маршрутна таблиця, що складається з набору впорядкованих за довжиною префікса сегментів. Кількість сегментів рівна кількості бітів в ідентифікаторі та кожен сегмент бере на себе відповідальність за ідентифікатори, що знаходяться у певному діапазоні відстані від поточного вузла. Всі операції модифікації таблиці розроблені, щоб бути потокобезпечними, оскільки оновлення інформації про маршрутизації та обробка запитів можуть відбуватися одночасно.

Віддалені вузли представлені як контакти кожен з яких зберігає наступну інформацію:

— ідентифікатор вузла;

— мережева адреса;

— інформація про необхідність використання сервер ретранслятор.

Контакти згруповані у сегменти залежно від їх відстані до поточного вузла та в межах одного сегменту організовані як LRU список із визначеною максимальною довжиною. Контакт, звернення до якого було найпізніше розміщується у кінці списку, а той, який був переглянутий раніше — на початку.

Індекс сегмента до якого має потрапити ідентифікатор визначається за допомогою підрахунку кількості нульових бітів на початку числового значення XOR відстані. Особливим випадком є нульова відстань, що відповідає поточному вузлу та ігнорується при додаванні до таблиці.

Після визначення відповідного індексу сегмента контакт або оновлюється або додається до таблиці. Якщо контакт з таким самим ідентифікатором вже присутній, його збережена інформація оновлюється, а сам контакт переміщується в кінець списку. Якщо контакт не існує, а сегмент має вільне місце — він

додається до списку. Якщо сегмент вже заповнений — контакт на початку списку видаляється і замінюється новим.

Для різних операцій маршрутизації таблиця надає спосіб отримання набору найближчих контактів до цільового ідентифікатора. На рисунку 3.2 зображено блок схему даного алгоритму.

Процес відбору починається з пошуку сегмента, до якого потрапляє ціль відносно локального вузла. Починаючи з цього сегмента, алгоритм симетрично розширюється на сусідні сегменти збільшуючи радіус, поки не буде досягнуто запитуваної кількості контактів або не будуть перевірені всі сегменти.

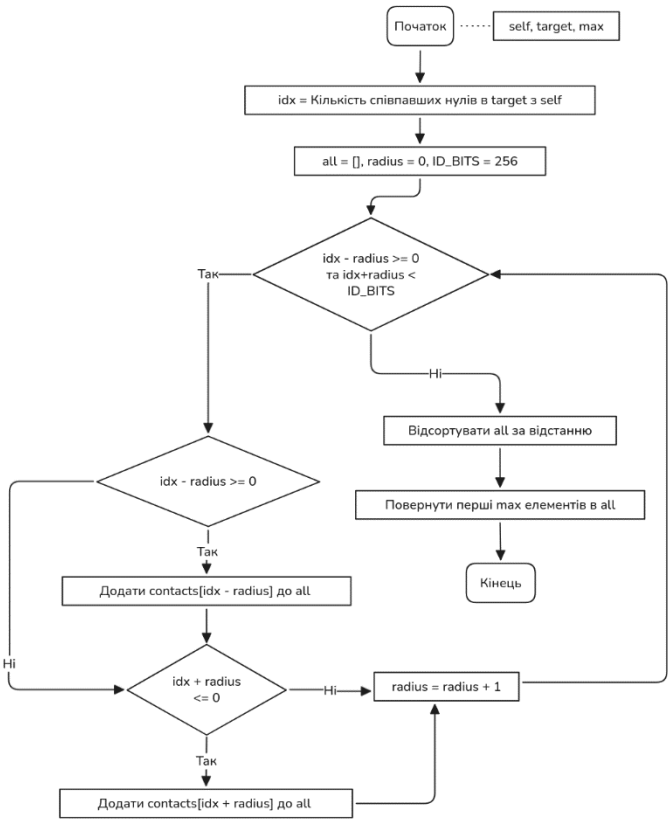


Рисунок 3.2 — Блок схема алгоритму пошуку найближчих вузлів

Об’єднаний список контактів сортується за XOR відстанню до цілі. Отриманий набір контактів виконує роль початкового списку кандидатів для методу ітеративного пошуку вузлів.

Процес ітеративного пошуку вузла повністю контролюється локально ініціатором запиту. Як зображено на рисунку 3.3, алгоритм починається з пошуку

у маршрутній таблиці найближчих відомих контактів до цілі. На кожній ітерації до кожного контакту у списку робиться запит на надання власних найближчих кандидатів. Запити виконуються паралельно та після їх виконання кожна успішна відповідь містить список контактів, які віддалений вузол вважає найближчими до цілі. Ці контакти об’єднуються в список, який очищається від дублікатів і сортується за відстанню.

Алгоритм порівнює найкращу (найближчу) відстань в оновленому списку з попередньою найкращою та якщо нова найкраща відстань є ближчою до цілі, то ітерація вважається успішною. В іншому випадку жоден новий контакт не є ближчим за ті, що вже знаходяться у списку.

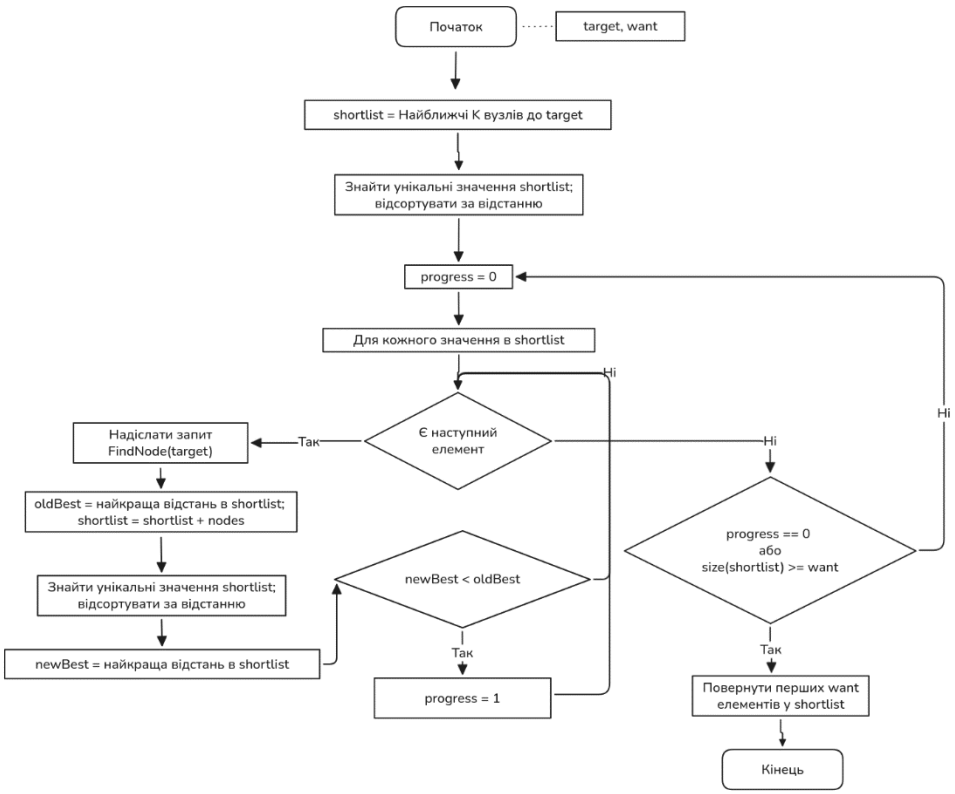


Рисунок 3.3 — Блок схема алгоритму ітеративного пошуку

Пошук завершується якщо після ітерації жоден новий ближчий до цілі контакт не був виявлений або кількість елементів зібраному списку не менша за запитувану.

Для підтримки стабільність DHT з часом, вузол виконує фонові задачі обслуговування. Цикл збирання сміття(GC Loop) періодично сканує локальне сховище ключів-значень на наявність записів термін дії яких закінчився і видаляє їх. Це запобігає нескінченному обслуговуванню застарілих значень і обмежує обсяг використаної пам’яті.

Цикл оновлення маршрутної таблиці(Bucket refresh Loop) періодично обирає випадковий ідентифікатор та виконує для нього ітеративний пошук. Метою є виявлення нових вузлів та оновлення сегменти у маршрутній таблиці.

Нарешті, цикл повторної перевірки(Revalidation Loop) зосереджується на стані відомих контактів. Для цього періодично обирається невеликий набір із таблиці маршрутизації яким надсилаються Ping запити. Контакти, які неодноразово дають збій, записуються та коли кількість відмов для певного вузла перевищує встановлений поріг — він видаляється з таблиці маршрутизації та навпаки, будь-яка успішна взаємодія очищає лічильник.

## 3.3 Протокол мережевої комунікації

Мережева комунікація в системі побудована на основі легкого RPC протоколу через TCP з повідомленнями, закодованими у форматі JSON. Кожен вузол виконує роль як клієнту, так і серверу.

Коли вузлу потрібно зв’язатися з віддаленим вузлом, він використовує контактну інформацію цього вузла з маршрутної таблиці. Якщо контакт містить пряму мережеву адресу, вузол намагається встановити пряме TCP з’єднання, а потім надсилає RPC повідомлення у форматі JSON.

Сторона клієнту розглядає кожен RPC як одноразову взаємодію, тобто з’єднання відкривається, відбувається один обмін запитом-відповіддю і закривається відразу після обробки відповіді, що дозволяє уникнути збереження великої кількості неактивних TCP-з’єднань. На стороні сервера кожен вузол прослуховує налаштовану адресу TCP на вхідні з’єднання. Нескінченний цикл приймає з’єднання та створює окремий обробник для кожного прийнятого.

Усі RPC мають єдиний формат повідомлень та складаються з наступних значень:

— тип RPC запиту;

— контакт відправника;

— необов’язкове значення ключа шуканої або доданої пари ключ-значення;

— необов’язковий вміст значення доданої пари ключ-значення;

— необов’язковий список контактів;

— булеве значення, що вказує чи знайдено запитувану пару ключ-значення.

Тип повідомлення RPC запиту визначає який метод буде викликано віддаленим вузлом. Система визначає наступні методи:

— PING, що тестує досяжність вузла;

— STORE, що пов’язує ключ зі значенням у вузлі;

— FIND_NODE, що запитує вузол надати список найближчих контактів до заданого ключа;

— FIND_VALUE, що поводиться як FIND_NODE, проте додатково повертає пару ключ-значення у разі успішного пошуку;

— FETCH_BLOCK, що запитує вміст блоку у байтовому форматі за визначеним ідентифікатором вмісту;

— PUT_BLOCK, що дозволяє вузлу надіслати блок до іншого вузла.

Щоб захистити таблицю маршрутизації від спуфінгу даних, сервер не довіряє адресі, що міститься в повідомленні. Натомість він перевіряє адресу точки з’єднання та поєднує IP адресу з портом, заявленим відправником. Якщо така реконструкція вдається, адреса контакту переписується з використанням фактичної віддаленої IP-адреси та заявленого порту.

У багатьох сценаріях вузли не мають змоги приймати вхідні TCP-з’єднання напряму, наприклад, коли вони знаходяться за NAT. Для підтримки системою приватних вузлів використовується сервер ретранслятор, який перенаправляє RPC запити на основі ідентифікаторів вузлів.

Протокол ретрансляції побудований на основі повідомлень, які містять тип запиту, ідентифікатор запиту, ідентифікатор цільового вузла, вміст RPC повідомлення та опціональне повідомлення про помилку. Всього визначено три

основні методи для взаємодії: реєстрація вузла, ретрансляція клієнтських запитів та допоміжний запит, що повертає публічну адресу цільового вузла.

Для реєстрації вузол відкриває TCP-з’єднання з ретранслятором і надсилає повідомлення, що містить його власний ідентифікатор. Ретранслятор записує це з’єднання в індексовану за ідентифікаторами вузлів таблицю. Якщо такий запис вже існує — попереднє з’єднання закривається та замінюється новим. Після закріплення вузол залишає це з’єднання відкритим і очікує на повідомлення, що містять RPC-запити, призначені для нього.

Для ретрансляції повідомлення до приватного вузла, відкривається окреме TCP-з’єднання з ретранслятором та надсилається повідомлення, що містить ідентифікатор запиту, ідентифікатор цільового вузла та RPC повідомлення, яке потрібно доставити. Якщо вузол зареєстровано, то ретранслятор зберігає підключення клієнта за ідентифікатором запиту і пересилає запит на доставлення з тим самим ідентифікатором запиту до адреси призначення.

Цільовий вузол отримує повідомлення, виокремлює вбудоване RPC повідомлення, оновлює свою маршрутну таблицю на основі інформації про відправника та надсилає повідомлення назад. Отримавши відповідь, ретранслятор знаходить запит клієнта за ідентифікатором запиту, видаляє його та повертає повідомлення, що містить відповідь до RPC запиту на очікуване з’єднання клієнта, а потім закриває це з’єднання.

Ретранслятор також підтримує запит, який запрошує повернути публічну адресу цільового вузла. У цьому випадку він перевіряє хост визначений у з’єднанні та відповідає спостережуваною адресою. Цей метод використовується вузлами для виявлення їх зовнішньої IP-адреси при налаштуванні контактної інформації.

## 3.4 Сховище блоків та модель закріплення

Завданням сховища блоків є збереження блоків, адресованих за змістом на диску, відстеження їхнього розташування та визначення тих які необхідно зберегти, а які можна видалити.

Фізично, сховище блоків поєднує два компоненти:

— структуровану директорію, що зберігає байти для кожного блоку;

— пари ключів-значень у LevelDB, що зберігають шляхи до файлів та закріплення індексовані за ідентифікаторами вмісту(CID).

Конструктор Меркл графу та вивантажувач(Block Fetcher) розглядають сховище блоків як абстрактний інтерфейс для додавання та видалення блоків за їх ідентифікаторами.

На диску блоки зберігаються в піддиректорії blocks/. Для уникання каталогів з великою кількістю записів, сховище блоків застосовує просту схему шардінгу — CID розділяється на короткі префікси, які утворюють вкладені піддиректорії, а цілий ідентифікатор використовується як ім’я файлу. Наприклад, CID може бути розміщене в blocks/ab/cd/<cid>.

LevelDB визначає декілька видів записів в одному індексі за допомогою додавання байтового префіксу:

— ключі що починаються із «b» зберігають значення CID до шляху файлу;

— ключі що починаються із «p» «s» зберігають інформацію про закріплення.

У системі файли представлені у виглядів блоків, що об’єднані у Меркл граф. Загалом, визначаються наступні види блоків:

— блоки з даними(листя графу), що зберігають фрагменти файлу;

— блоки вузли, що зберігають інформацію про дочірні блоки та як їх зібрати;

— кореневий блок(корінь графу), що зберігає метадату файлу.

Конструктор графу будує таку структуру поверх використання інтерфейсу сховища блоків. Отримавши потік байтів файлу, він проходить наступні етапи:

1) вхідний потік читається послідовно по фрагментах фіксованого розміру; для кожного фрагмента створюється блок даних, який вставляється в сховище;

2) конструктор будує граф знизу догори та блоки з даними об’єднуються у групи фіксованого розміру, що повторюється рівень за рівнем, поки не залишиться лише корінь графу;

3) створюється маніфест файлу, що містить загальний розмір файлу, розмір фрагмента, кореневий CID та опціональні метадані, такі як ім’я файлу та MIME type.

Як зображено на рисунку 3.4 вивантаження блоку відбувається у два етапи: локальний пошук у сховищі та мережевий запит.

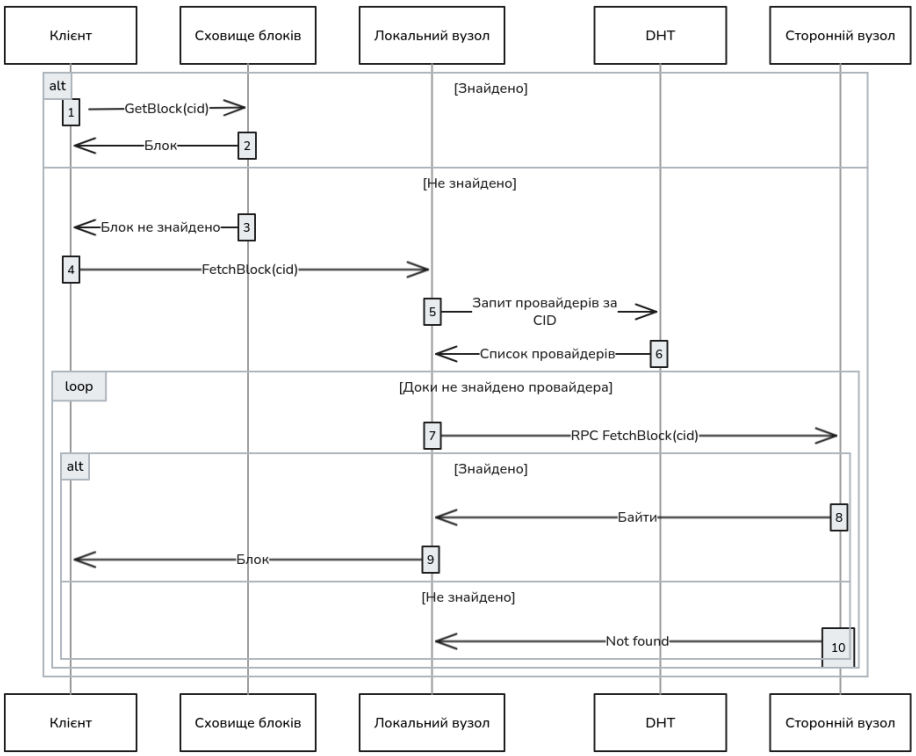


Рисунок 3.4 — Діаграма послідовності вивантаження блоку

Для локального пошуку клієнт звертається до індексу LevelDB під префіксом «b», щоб отримати відносний шлях для цього CID. Якщо запис присутній, він зчитує відповідний файл з диска, декодує його як блок і перевіряє, чи відповідає CID блоку запитуваному.

Якщо пошук у локальному сховищі неуспішний або файл не може бути прочитаний, сховище блоків вивантажує блок з мережі. У разі успіху отримані байти декодуються та перевіряються на відповідність очікуваному CID. Якщо вони збігаються, блок-сховище зберігає блок локально, використовуючи стандартну процедуру вставки, щоб наступні запити могли обслуговуватися з диска.

Отримання повного файлу починається з отримання маніфесту файлу за його CID та рекурсивно обходу графу при наступних кроках. Для зменшення затримки вивантаження передбачено варіант із використанням паралелізму. Він має ту саму логічну структуру, але дозволяє одночасно завантажувати кілька піддерев графу.

Збереження файлів контролюється за допомогою механізму закріплення блоків. Загалом, система розрізняє наступні види закріплень:

— жорсткі закріплення, що зберігають блок при будь-яких умовах;

— м’які закріплення, що мають час життя;

— прямі закріплення, що захищають лише єдиний блок;

— рекурсивні закріплення, що захищають повний граф або підграф.

Збирач сміття використовує ці записи для визначення набору активних блоків та видалення усіх інших, що не потрапляють до нього. Робота збирача сміття відбувається у три етапи:

1) сканування жорстких закріплень, щоб зібрати рекурсивні корені та прямі корені, окрім того, до набору прямих коренів додаються м’які закріплення у яких не закінчився термін життя;

2) для кожного рекурсивного закріплення виконується обхід графу, де кожен відвіданий CID позначається як активний;

3) скануються всі записи з індексу блоків та відбувається перевірка на наявність у наборі активних та при відсутності збирач сміття видаляє відповідний файл з диска та LevelDB.

## 3.5 Постановка та результати обчислюваних експериментів

Метою проведення обчислюваних експериментів є надання оцінки ефективності роботи запропонованої системи під навантаженням з фокусом на наступні цілі. По-перше, надання оцінки надійності та доступності запропонованої розподіленої файлової системи з адресацією вмісту. Оскільки система призначена для роботи в динамічному середовищі — важливо кількісно оцінити, як часто дані залишається доступним. По-друге, експерименти оцінюють

продуктивність завантаження та вивантаження файлів з погляду затримки та пропускної здатності.

### 3.5.1 Постановка експериментів

Усі експерименти проводяться в локальному кластері вузлів на одній фізичній машині. Кожен вузол прослуховує окремий TCP-порт та має власну маршрутну таблицю та сховище блоків. Така конфігурація забезпечує контрольоване середовище, в якому затримка запитів та збої визначаються ефективністю системи, а не нюансами реальних мереж.

В усіх експериментах використовуються однакова конфігурація розподіленої хеш-таблиці. Вузол ініціалізується з коефіцієнтом реплікації k=20 та фактором паралелізму ітеративного пошуку α=5.

Для імітації навантаженого середовища, періодично обирається підмножина вузлів, які примусово вилучаються з мережі шляхом припинення їх роботи на час в діапазоні від 200 мс до 1,5 с. Після закінчення часу відключення вузли відновлюють нормальну роботу і продовжують брати участь у маршрутизації та зберіганні даних.

Експериментальна установка складається з операцій завантаження та вилучення файлів. Для кожної конфігурації виконується велика кількість незалежних випробувань завантаження-вивантаження. Кожна спроба складається з завантаження файлу в систему і подальшої спроби отримати файл з іншого вузла.

В ході експериментів збирається набір кількісних показників, призначених для оцінки як правильності, так і швидкості виконання. Головною метрикою є доступність даних. Для кожного експерименту доступність даних визначається за наступною формулою:

$$A = \left( \frac{N_y}{N} \right) \cdot 100\,\%,$$

де $N_y$ — кількість успішних запитів вивантаження,

N — загальна кількість запитів вивантаження.

Завантаження вважається успішним, якщо отримано повну послідовність байтів та отриманий ідентифікатор відповідає очікуваному значенню. Неуспішні завантаження включають як явні помилки, наприклад, перевищення часу очікування, так і неявні помилки, наприклад, неправильний або неповний вміст файлу.

Для характеристики швидкості виконання збираються показники як для операцій завантаження, так і для операцій вивантаження. Для кожного випробування система реєструє час, що минув між початком і завершенням операції. Для завантажень це включає розбиття файлу на блоки, побудову графу, запис блоків до сховища та поширення їх мережею. Для завантажень це включає вивантаження маніфесту файлу по його ідентифікатору, обхід графу та реконструкцію файлу. Розподіл затримок підсумовується за допомогою процентилів: медіана, P95, P99. Пропускна здатність визначається як співвідношення розміру файлу у байтах до виміряної повної затримки відповідної операції:

$$T = \frac{S}{\Delta t},$$

де S — розмір файлу у байтах,

Δt — затримка повного запиту у секундах.

### 3.5.2 Залежність доступності даних від розміру мережі

Даний експеримент досліджує, як змінюється доступність даних залежно від кількості вузлів, що беруть участь в мережі. Для цього обираються наступні значення: S=5 МБ, C=256 КБ, N=5-50, де S та C — розміри файлу та блоку відповідно та N — розмір кластера. Отримані результати наведено на рисунку 3.5, що зображає графік залежності доступності від кількості вузлів у мережі.

Вимірювання показують чітку позитивну кореляцію між розміром кластера та доступністю. Для невеликого кластера з N=5 вузлами доступність становить приблизно A≈59% При цих значеннях кількість реплік близька до загальної кількості вузлів, тому коли деякі з них виходять із ладу, цілком ймовірно, що всі копії певного блоку одночасно стають недоступними. У міру зростання кластера

до N=10 та N=25 доступність збільшується приблизно до 79% та 91% відповідно. А при N=50 доступність досягає 98%.

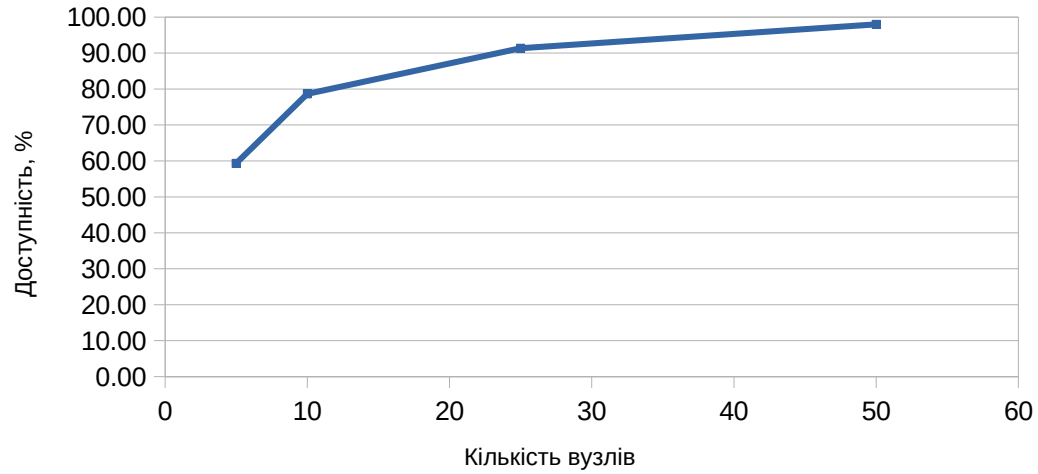


Рисунок 3.5 — Графік залежності доступності даних від розміру мережі

Така поведінка є передбачуваною, тому що у міру збільшення кількості вузлів — репліки кожного блоку розподіляються між більшою та більш різноманітною групою вузлів. Ймовірність одночасної недоступності всіх реплік певного блоку швидко зменшується зі збільшенням кількості вузлів, що, своєю чергою, збільшує ймовірність того, що кожен блок у файлі залишатиметься доступним. Отже, результати експерименту підтверджують, що запропонована система добре масштабується за розміром кластера з погляду доступності.

### 3.5.3 Затримка та пропускна здатність завантаження та вивантаження файлу

Даний набір експериментів оцінює, як кінцева затримка та пропускна здатність залежать від розміру файлу та фрагмента для фіксованого розміру кластера N=25. Розміри файлів $S \in \{2,5,20\}$ МБ поєднуються із розмірами фрагментів $C \in \{256,512,1024\}$.

Результати затримки завантаження на рисунку 3.6 показують чітке зростання зі збільшенням розміру файлу для всіх розмірів фрагментів. З фрагментами розміром 256 КБ медіанна затримка завантаження зростає з

приблизно 141 мс для файлів розміром 2 МБ до приблизно 341 мс для файлів розміром 5 МБ і до приблизно 1,36 с для файлів розміром 20 МБ.

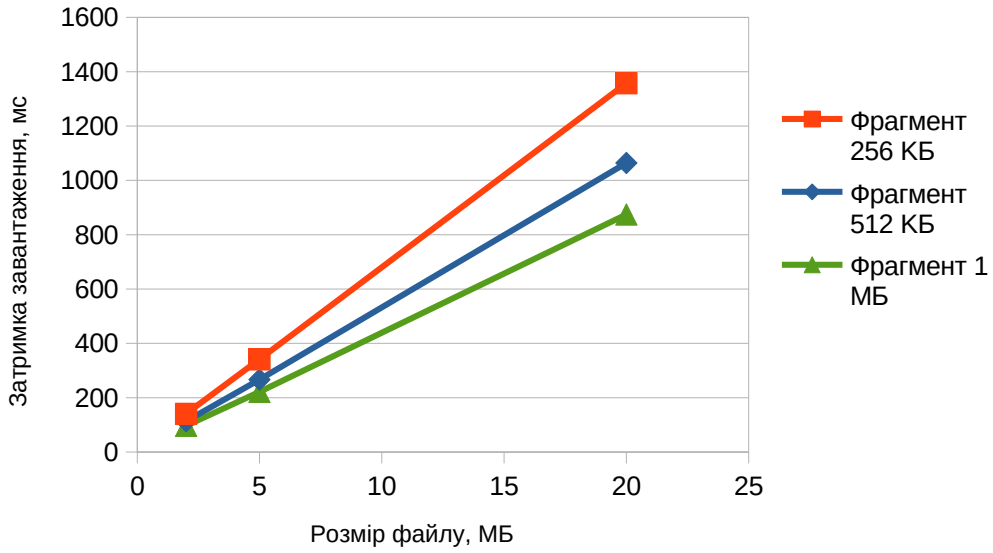


Рисунок 3.6 — Графік залежності затримки завантаження від розміру файлу

Проте, як можна помітити, час завантаження є меншим для файлів однакового розміру з більшими фрагментами. Різниця є більш явною для файлу розміром 20 МБ, що є очікуваним оскільки більші файли розділяються на більшу кількість блоків, які необхідно обробляти, записувати на диск та оголошувати в DHT. Зі збільшенням розміру блоку зменшується їх кількість у файлі, тому вузол виконує менше запитів та записів на диск. В результаті затримка завантаження змінюється з діапазону менше ніж 100 мс для невеликих файлів з великими блоками до приблизно 1 с для найбільших файлів з найменшими блоками. Це вказує на те, що в поточній системі витрати на обробку блоку мають більший вплив ніж пропускна здатність вводу-виводу.

Результати пропускної здатності вивантаження в таблиці 3.7 підкреслюють комплементарний аспект системи. Тут домінантним фактором є не розмір блоку, а розмір файлу. У міру збільшення розміру файлу система може видавати паралельно більше блоків для завантаження, оскільки її можливості застосовуються більш повно.

При цьому розмір блоку все ще має помітний ефект на пропускну здатність. Більші блоки, як правило, забезпечують дещо кращі показники для великих

файлів коштом зменшення накладних витрат на блок, а менші навпаки — показують кращі результати для малих файлів.

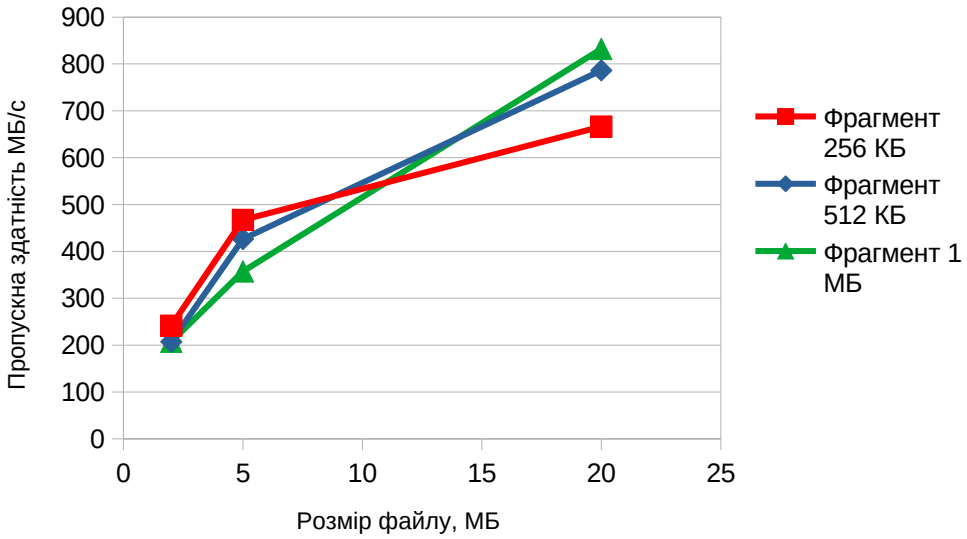


Рисунок 3.7 — Залежність пропускної здатності вивантаження від розміру файлу

Додатково було визначено вплив параметра міри паралелізму на пропускну здатність вивантаження. Цей параметр обмежує кількість одночасних вивантажень блоків в одному графі. Для цього експерименту розмір кластера був обраний як N=50, розмір файлу S=5 МБ та розмір фрагмента C=256 КБ, а міра паралелізму P={1,5,10,20}.

Результати на рисунку 3.8 показують, що збільшення має сильний позитивний вплив на продуктивність вивантаження файлів до певної міри.

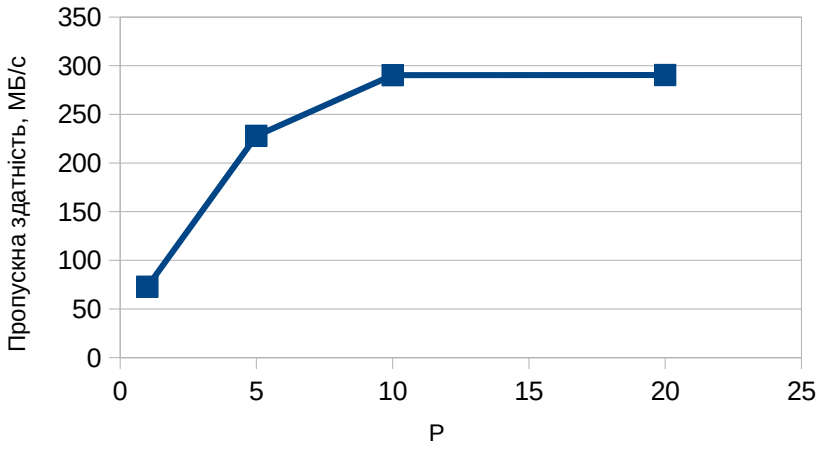


Рисунок 3.8 — Залежність пропускної здатності вивантаження від P

При послідовному вивантаженні P=1 пропускна здатність є доволі обмеженою, оскільки клієнт чекає на кожен блок, перш ніж запитувати наступний. Навіть невелика зміна параметра P=5 вже збільшує пропускну здатність більш ніж утричі, що вказує на те, що кілька незалежних передач блоків можуть відбуватися одночасно без перевантаження системи.

Поза цією межею переваги подальшого збільшення P стають помірнішими. Перехід від 5 до 10, а потім до 20 все ще трохи збільшує пропускну здатність, але виграш є явно сублінійним і крива починає вирівнюватись. Це свідчить про те, що в оцінюваному середовищі реалізація починає наближатися до власних обмежень.

Додаткові паралельні запити здебільшого збільшують накладні витрати на координацію на стороні клієнта і приносить лише незначне поліпшення швидкості передачі даних. Важливо, що доступність залишається на тому ж високому рівні для всіх протестованих значень P, що свідчить про те, що налаштування цього параметра в першу чергу є компромісом між затримкою і пропускною здатністю на стороні клієнта та складністю реалізації та використаних ресурсів, без істотного впливу на ймовірність успішного отримання файлу.

## Висновки до розділу 3

Даний розділ представив архітектуру та реалізацію розподіленої системи зберігання файлів із адресацією вмісту. Було визначено чітку структуру підсистем: DHT для маршрутизації вузлів, сховище блоків для збереження файлів на диску, керування механізмами очищення та закріплення, Меркл граф для фрагментації файлів та сервісний шар для створення шлюзу між клієнтськими застосунками і внутрішніми компонентами системи.

Експерименти показали, що доступність файлів помітно зростає зі збільшенням розміру. Збільшення розміру блоку зменшує затримку завантаження завдяки зменшенню накладних витрат на обробку, тоді як більша фрагментація дає змогу досягати ефективної пропускної здатності під час вивантаження завдяки паралельній обробці графу.

# 4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У даному розділі розглядаються аспекти взаємодії користувача із розробленою системою. Будуть описані системні вимоги та процедуру встановлення програмного забезпечення, включаючи необхідні програмні та апаратні засоби та кроки необхідні для розгортання. Окрім того, буде продемонстровано користувацький інтерфейс системи розробленого настільного застосунку, що ілюструє типові робочі процеси, такі як додавання файлу, вивантаження та перегляд даних з мережі, закріплення та перегляд закріплених файлів.

## 4.1 Системні вимоги для роботи із системою

Розроблене програмне забезпечення розгортається за допомогою програми командного рядку(CLI), що виконує роль як клієнтського застосунку, так і команди для ініціалізації вузла, що надає доступ до HTTP клієнту. Окрім того, створено користувацький інтерфейс у вигляді настільного застосунку. Процес інсталяції складається з налаштування трьох компонентів: фонового сервісу, програми командного рядку та настільного застосунку.

Програма командного рядка постачається як окремий виконуваний файл, який спілкується з вузлом через HTTP API та надає команди для додавання файлів, отримання даних за ідентифікатором вмісту, закріплення та відкріплення, а також використання DHT як сховища ключів-значень.

Після завантаження CLI, на цільовій машині встановлюється фоновий сервіс вузла. У типовій конфігурації вузол інтегрується з systemd шляхом використання конфігураційного файлу, який постачається разом із програмним забезпеченням та запускає та ініціалізує вузол із відповідними аргументами. Після встановлення та увімкнення сервісу, життєвий цикл вузла керується за допомогою стандартних команд systemd.

Настільний застосунок опціонально встановлюється для користувачів, яким потрібен графічний інтерфейс. Додаток на базі Tauri запакований як стандартний виконуваний файл для настільних комп’ютерів. У поточній реалізації передбачається, що вузол працює як фонова служба та не керує життєвим циклом вузла, а покладається на systemd для запуску, зупинки, перевірки статусу та перегляду логів сервісу.

Система призначена для операційної системи Linux на базі 64-бітної архітектури процесора та не потребує вимогливого апаратного забезпечення. До необхідного програмного забезпечення входять наступні компоненти:

— операційна система сімейства Linux із підтримкою systemd;

— підтримка мережевого стеку TCP/IP;

— компілятор мови програмування Golang при потребі збирання виконуваного файлу з вихідного коду;

— графічне середовище X11 або Wayland і стандартні залежності для виконання настільних програм у вибраному дистрибутиві.

На апаратній стороні до вимог відносяться:

— мінімум 512 МБ оперативної пам’яті та 2 GB рекомендованої;

— багатоядерний процесор рекомендується;

— 5 ГБ вільного місця на жорсткому диску або SSD;

— стабільне інтернет-з’єднання.

## 4.2 Демонстрація інтерфейсу

Користувацький інтерфейс настільного застосунку організовує усі можливості програмного забезпечення в 4 вкладки - «Файли», «Вузли», «Налаштування» та «Сервіс». Основними можливостями є — завантаження та перегляд контенту, моніторинг стану сховища та керування фоновим сервісом.

Як можна побачити на рисунку 4.3 зображено вкладку «Сервіс». Тут можна помітити статус стану сервісу, елементи керування для старту/зупинку та перезапуску. А також поле для налаштування прапорів запуску вузла. У нижній панелі відображено журнал, що відображає останні логи.

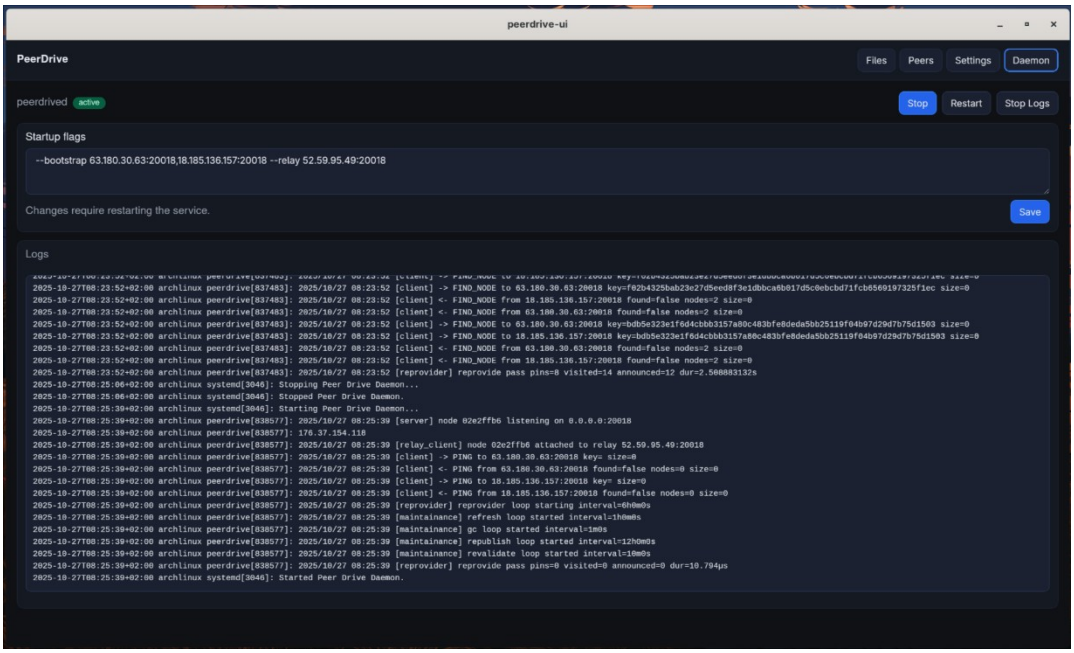


Рисунок 4.3 — Інтерфейс вкладки «Сервіс»

На рисунку 4.4 зображено вкладку «Налаштування». Тут можна побачити загальні налаштування. Тут можна задати конфігураційні параметри, а також вказати спосіб завантаження — із повним зберігання файлу локально чи розосередженням блоків поміж вузлів.

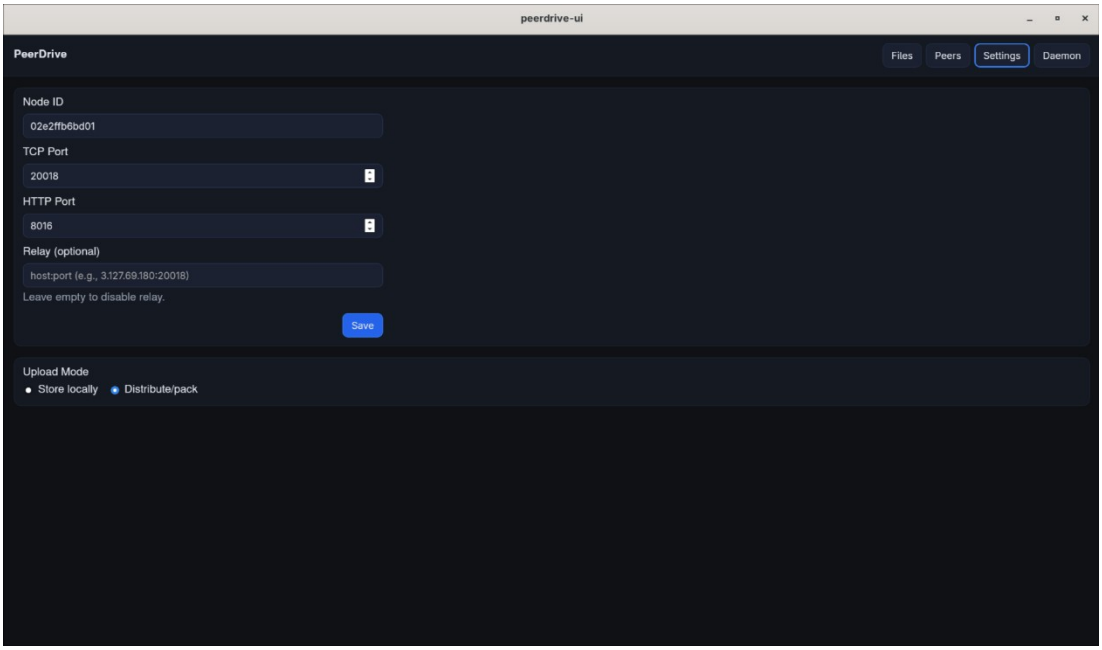


Рисунок 4.4 — Інтерфейс вкладки «Налаштування»

На рисунку 4.5 зображено вкладку «Вузли». Тут знаходиться поточне оточення вузла: активні вузли та їхня контактна інформація — ідентифікатор, публічна адреса та опційно вказано адресу підключеного ретранслятора.

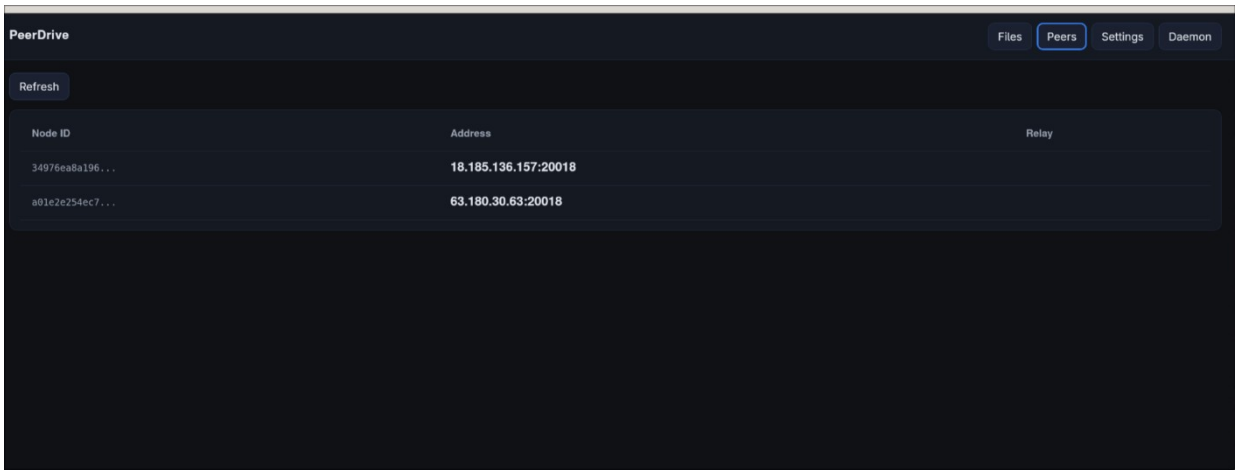


Рисунок 4.5 — Інтерфейс вкладки «Вузли»

На рисунку 4.6 зображено вкладку «Файли». Тут видно панель, що дозволяєдодати файл, оновити стан сховища та завантажити новий з мережі із введенням CID.

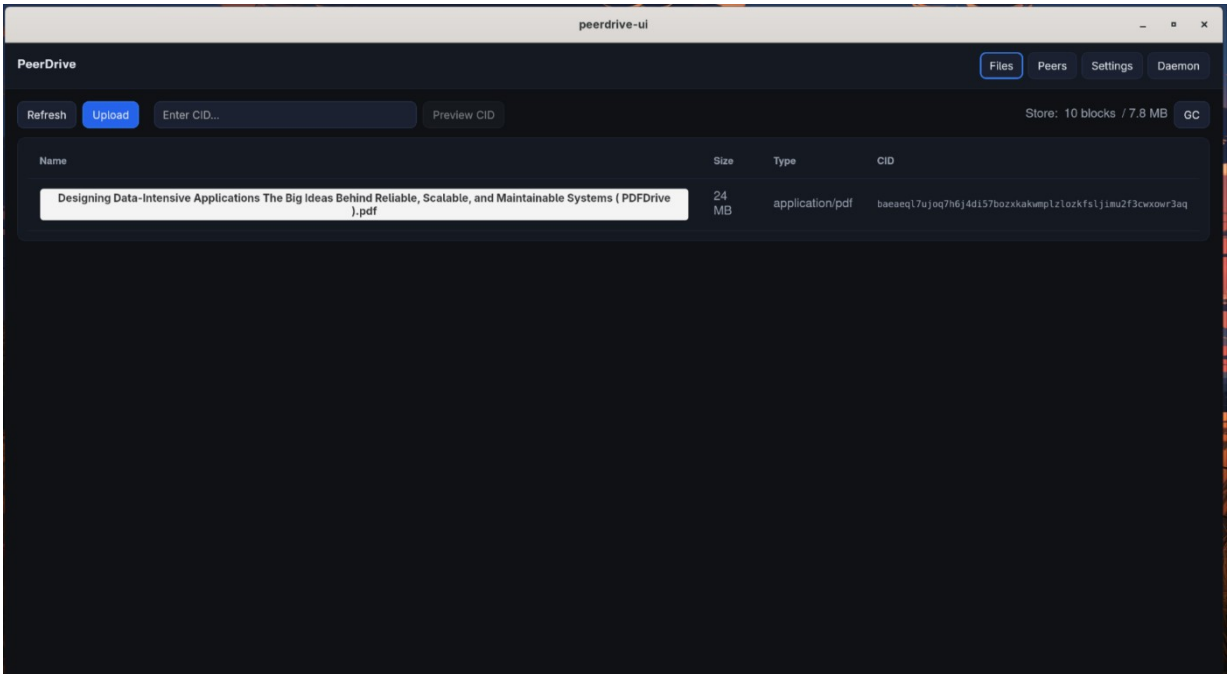


Рисунок 4.6 — Інтерфейс вкладки «Вузли»

У списку можна побачити завантажений файл із розміром 24 мегабайти, при цьому розмір сховища займає всього 7.8 мегабайт, оскільки частина блоків файлу знаходиться на інших вузлах.

Натиснувши на файл, відкриється модальне вікно із переглядом файлу. На рисунку 4.7 зображено інтерфейс відображення вікна перегляду.

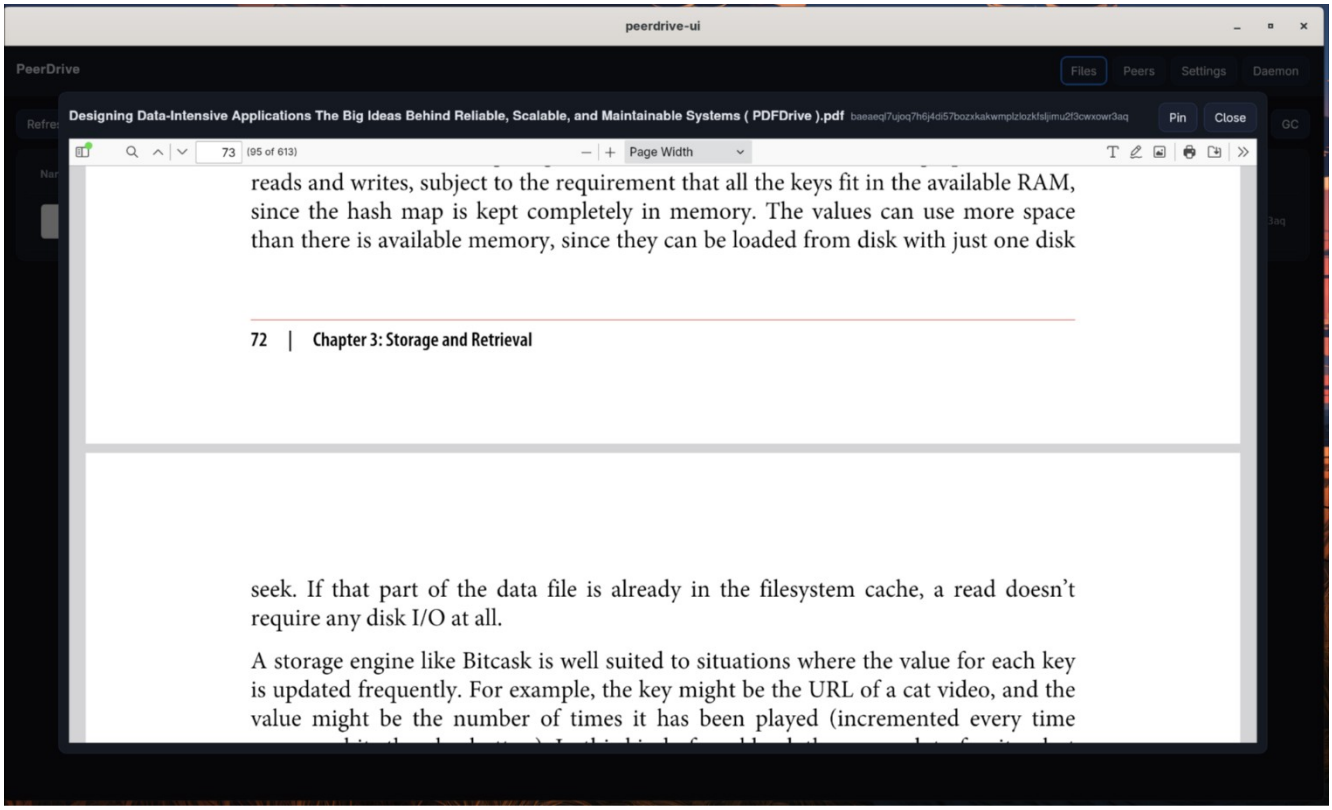


Рисунок 4.7 — Модальне вікно перегляду файлу

Як можна побачити, файл повністю завантажився та відкрився у модальному вікні. При цьому, якщо перевірити зайняте місце у сховищі, то можна помітити, що файл був повністю кешований локально. На рисунку 4.8 зображено оновлений стан сховища.

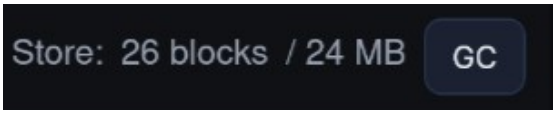


Рисунок 4.8 — Стан сховища після завантаження PDF файлу

Окрім вище наведених функцій, у даній вкладці є можливість вручну запустити збирач сміття. Як можна помітити на рисунку 4.9, розмір сховища повернувся до початкового.

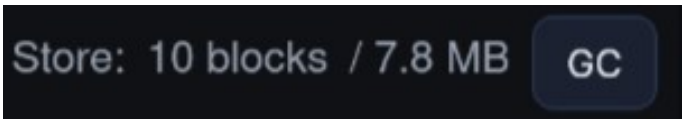


Рисунок 4.9 — Стан сховища після натискання на кнопку «GC»

Останньою функцією даної вкладки є — завантаження файлу з мережі по CID. Як видно на рисунку 4.10, введення відбувається у форму «Enter CID...».

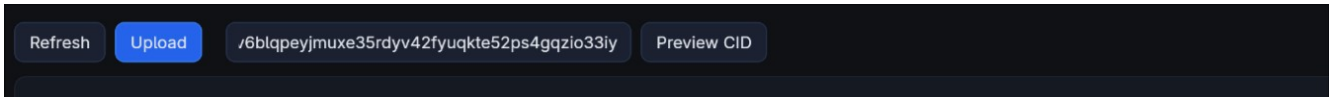


Рисунок 4.10 — Форма введення ідентифікатора контенту

При натисканні на кнопку «Preview CID», знову відкриється модальна форма перегляду файлу як зображено на рисунку 4.11. Окрім того, з даного вікна файл можна закріпити до свого списку.

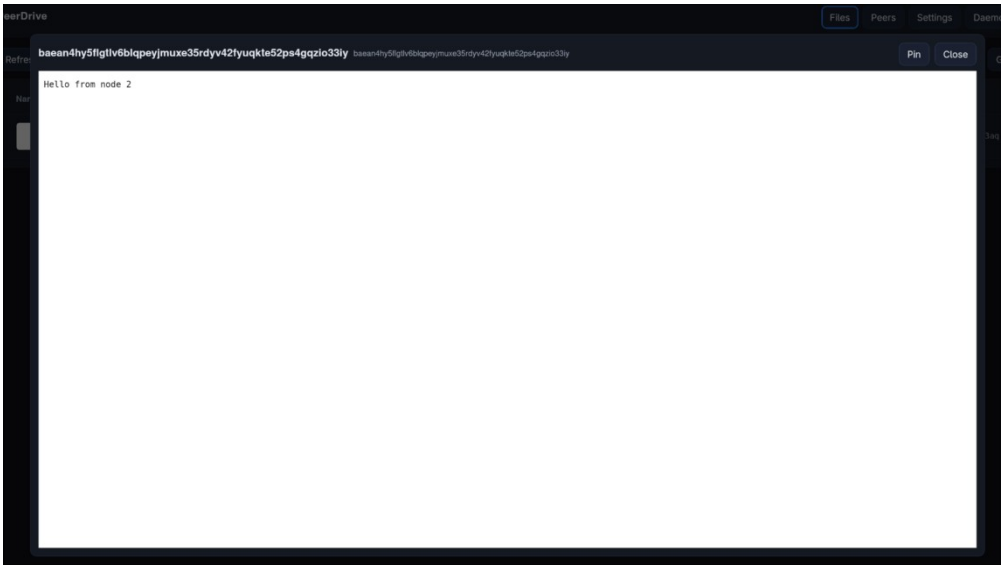


Рисунок 4.11 — Відображення завантаженого файлу

Окрім взаємодії через настільний застосунок, файли можуть бути також переглянуті або завантажені безпосередньо у веббраузері, оскільки доступ до них здійснюється через HTTP-сервер, який повертає вміст файлу у вигляді потоку даних.

## Висновки до розділу 4

У цьому розділі описано процес встановлення та сценарії роботи користувача з програмного забезпечення. Показано що вузол може працювати у режимі фонового сервісу. Як програма командного рядка, так і настільний застосунок комунікують через єдиний HTTP API, що створює розділення між внутрішніми механізмами та зовнішньою взаємодією.

Що стосується взаємодії користувача з програмною системою, було продемонстровано, що базові механізми адресації за змістом та пірингові мережі можуть бути представлені користувачам за допомогою звичних абстракцій.

Операції завантаження та вилучення, закріплення та перегляд вмісту, а також перевірка стану вузлів представлені як операції на рівні файлів, без необхідності для користувачів розбиратися у внутрішніх механізмах системи.

CLI пропонує інтерфейс, придатний для автоматизації у серверних середовищах, а настільна програма надає графічний інтерфейс для інтерактивного використання. Результати показують, що розроблена програмна система є придатною для практичного використання. Операції зберігання та завантаження файлів можуть бути вбудовані в стандартні робочі процеси, приховуючи більшу частину складності запропонованої системи.

# 5 РОЗРОБКА СТАРТАП-ПРОЄКТУ

Даний розділ присвячений розробці стартап-проєкту у якому сформульовано ідею стартапу та представлено загальну концепцію продукту, проведено технічний аудит запропонованого рішення та надано оцінку його готовності до використання на ринку. Проведено аналіз середовища ринку, визначені потенційні клієнти, ризики та можливості, а також конкурентне середовище. Розроблена відповідна ринкова стратегія та окреслено маркетингову стратегію, що відповідає технічному стартапу.

## 5.1 Ідея стартап-проєкту

Ідея стартап-проєкту є децентралізована платформа для зберігання даних. По суті, продукт є розподіленою мережею зберігання та розповсюдження файлів, де користувачі можуть зберігати, отримувати та обмінюватися файлами без координації центрального сервера. Файли, додані до системи, розділяються на блоки за змістом і поширюються мережею вузлів-учасників; будь-який вузол, що має відповідні блоки, може згодом надати файл за його ідентифікатором змісту. Таким чином, ця платформа служить альтернативою хмарному сховищу.

На додаток, децентралізована реплікація означає, що не існує єдиної точки відмови або єдиної сторони, яка контролює доступ до даних, що надає користувачам більшу автономність.

Основні напрямки застосування включають: резервне копіювання та архівування, розподіл великих файлів та інтеграція додатків, що потребують зберігання об’єктів даних.

Незалежність від вендорів та відкритість мережі дозволяють користувачам уникнути прив’язаності до постачальника хмарних послуг. Нарешті, включення високорівневого API, інтерфейсу командного рядка та настільної програми робить продукт зручним для користувача, незалежно від його технічних знань.

У таблиці 5.1 наведено загальний огляд ідеї стартап-проєкту, в якому підсумовано пропозиції, основні сфери застосування та переваги для користувачів.

Таблиця 5.1 — Опис ідеї стартапу

| Зміст ідеї | Напрямки застосування | Вигоди для користувачів |
|---|---|---|
| Розробка децентралізованої платформи для зберігання даних | Резервне копіювання та архівування даних | Відмовостійкість даних завдяки надмірності копій |
| | Децентралізована мережа розповсюдження контенту(CDN) | Адресація за вмістом гарантує, що отримані дані відповідають тому, що зберігається |
| | Інтеграція рішення для збереження даних з іншими додатками | Відсутнісність залежності від вендорів хмарних технологій |

Для чіткішого позиціювання стартап-ідеї в конкурентному середовищі децентралізованих і хмарних сховищ, основні економічні та технічні характеристики пропонованої системи порівнюються з декількома еталонними концепціями. Таке порівняння висвітлює відносні слабкі та сильні сторони проєктної ідеї. У таблиці 5.2 представлено порівняння основних характеристик з конкурентами.

Таблиця 5.2 — Визначення характеристик ідеї проєкту

| № з/п | Економічні характери-стики | Концепція (стратегія) конкурентів | | | | Слабкі сторони | Нейтральні сторони | Сильні сторони |
|---|---|---|---|---|---|---|---|---|
| | | Власний проєкт | IPFS | Filecoin | Storj | | | |
| 1 | Гарантія збереження даних | Закріплення/ реплікація даних | Закріплення даних | Блокчейн | локчейн | | + | |

Кінець таблиці 5.2

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | Продуктивність | Помірна | Помірна | Варіюється | Висока | | + | |
| 3 | Платіжна модель | Безкоштовна | Безкоштовна | На основі використання | Комерційні тарифи | | | + |
| 4 | Простота використання/ інтеграції | CLI/HTTP API/Relay/ Desktop; відсутність блокчейну | CLI/API; відсутність блокчейну | В имагає блокчейн | SDK/ сумісність з S3;Вимагає блокчейн | | | + |
| 6 | Контроль над даними | Самостійний хостинг | Самостійний хостинг | Дані зберігаються валідаторами блокчейну | Локація даних визначається сервісом | | | + |

Результати порівняння показали, що запропонований проєкт займає збалансовану позицію відносно вибраних концепцій конкурентів. З економічної точки зору, відсутність плати і можливість повторного використання наявного обладнання роблять його особливо привабливою, розробників, невеликих організацій і спільнот, які прагнуть мінімізувати поточні витрати.

З технічної точки зору, поєднання простого HTTP API, CLI та настільного клієнта забезпечує порівняно високу легкість інтеграції та використання, що частково компенсує помірну продуктивність, очікувану від повністю децентралізованої системи. Відсутність механізму нагороди за зберігання даних означає, що довгострокова стійкість даних залишається відносною слабкістю в

порівнянні з Filecoin або Storj, які можуть запропонувати більш надійні гарантії, підкріплені економічними механізмами. Водночас використання тими ж сервісами блокчейн технологій робить їх складними для інтеграції.

Загалом, запропонований стартап найкраще підходить для сегментів, які надають пріоритет відкритості, контролю та зручності використання над суворими гарантіями збереження та максимальною продуктивністю.

## 5.2 Технологічний аудит стартап-проєкту

Для оцінки можливості створення та подальшого розвитку програмного продукту на основі реалізованої системи важливо визначити доступність і зрілість технологій розроблення. У таблиці нижче наведено основні програмні технології, які використовуються або плануються до використання в рамках проєкту.

Таблиця 5.3 — Технологічна здійсненість проєкту

| Ідея проєкту | Технологія | Наявність технології | Доступність технологій |
|---|---|---|---|
| Реалізація вузла P2P-системи | Мова Go та її стандартні бібліотеки | + | Безкоштовна; Доступна; Open-source |
| Реалізація API для взаємодії з вузлом | Стандартна бібліотека Go | + | Безкоштовні; Доступні |
| Інструменти командного рядка | Cobra | + | Безкоштовно; Доступно |
| Управління вузлом як службою | systemd | + | Безкоштовно; Доступно |

Представлені у таблиці технології розроблення свідчать про високу технологічну здійсненність проєктної ідеї. Усі ключові компоненти є зрілими, відкритими та безкоштовними для використання, що усуває технологічні бар’єри

для створення та підтримки програмного продукту, а також мінімізацію ліцензійних ризиків та залежності від пропрієтарних рішень.

Додатковою перевагою є широке поширення та хороша документованість обраних технологій, що спрощує залучення нових розробників до проєкту і створює передумови для довгострокового супроводу системи.

## 5.3 Аналіз ринкових можливостей

Для визначення комерційних перспектив стартапу необхідно встановити ключові параметри потенційного ринку, його динаміку, конкурентний рівень та фактори, що визначають попит. Таблиця 5.4 узагальнює основні ринкові індикатори, що є релевантними для впровадження децентралізованого сховища даних.

Таблиця 5.4 — Характеристика потенційного ринку стартап–проєкту

| Показники стану ринку | Характеристика |
|---|---|
| Розмір і темпи зростання | Світовий ринок децентралізованого зберігання даних демонструє швидке зростання, з високою динамікою попиту з боку розробників, організацій та користувачів, орієнтованих на конфіденційність. |
| Географічне охоплення | Продукт може розповсюджуватися глобально завдяки відкритій моделі та мінімальним технічним бар’єрам. |
| Рівень конкуренції | Сегмент містить кілька сильних гравців (IPFS, Filecoin, Storj), але нішеві рішення з акцентом на самохостингу та простоті інтеграції залишаються недостатньо покритими. |
| Регуляторне середовище | Посилення вимог до захисту даних, локалізації та прозорості обробки стимулює зростання попиту на самостійне розгортання сховищ. |

Кінець таблиці 5.4

| | |
|---|---|
| Цінова чутливість клієнтів | Малі та середні підприємства активно шукають способи скорочення витрат на централізовані хмарні сервіси та зменшення залежності від вендорів. |

Аналіз потенційного ринку свідчить про наявність сприятливих умов для впровадження децентралізованого сховища даних, орієнтованого на самохостинг та простоту використання.

Інтенсивне зростання сегмента, продиктоване попитом на незалежність від вендорів, підтверджує, що користувачі та організації дедалі частіше прагнуть повного контролю над власною інфраструктурою. Наявність сильних конкурентів не нівелює ринкових можливостей, оскільки існує недостатньо заповнена ніша між високотехнологічними, але складними для інтеграції системами та комерційними централізованими сервісами з високою вартістю. У сукупності це дозволяє стверджувати, що ринкове середовище є сприятливим для запуску стартап-проєкту, а його позиціювання як самохостингова, доступна та технологічно зрозуміла платформи відповідає актуальним тенденціям та запитам цільової аудиторії.

Для коректного позиціонування стартап-проєкту необхідно сформулювати основні групи потенційних споживачів, описати їхню поведінку на ринку та ключові вимоги до продукту. Таблиця 5.5 систематизує основні типи попиту, що формують ринок децентралізованого сховища даних, а також відповідні їм категорії користувачів.

Таблиця 5.5 — Характеристика потенційних користувачів стартап–проєкту

| Попит, який формує ринок | Користувачі | Поведінка цільової групи | Вимоги споживачів до продукту |
|---|---|---|---|
| Надійне зберігання даних | Наукові установи, держоргани | Переважно самостійно розгортають інфраструктуру | Гарантія цілісності, прозорі формати, відтворюваний доступ |

Кінець таблиці 5.5

| | | | |
|---|---|---|---|
| Економічно ефективне резервне копіювання з контролем над даними | Малі та середні підприємства, SaaS-провайдери | Прагнуть знизити витрати | Просте розгортання, передбачуване споживання ресурсів |
| Сховище з адресацією вмісту | Розробники ПЗ, Web3-проєкти | Працюють з API та “інфраструктурою як код”, тестують рішення через пілотні проєкти | Документовані API, можливість автоматизації, базові засоби моніторингу |
| Спільний доступ до файлів | Технічно підковані користувачі | Готові запускати власну інфраструктуру | Зручний GUI, кросплатформність, гнучкість до налаштування |

Проведена сегментація демонструє, що потенційні користувачі стартап-проєкту характеризуються різними, але взаємодоповнювальними типами попиту. З одного боку, інституційні користувачі та підприємства очікують від системи високих гарантій цілісності, контрольованості та економічної ефективності резервного копіювання, що безпосередньо відповідає можливостям запропонованої системи. З іншого боку, розробники та технічно підготовлені індивідуальні користувачі формують попит на гнучкість до налаштування, відкриті API та зручні користувацькі інтерфейси для повсякденної роботи з файлами. Таким чином, цільова аудиторія не зводиться до однієї вузької ніші, а поєднує вимоги інфраструктурних команд (DevOps, SaaS-провайдери) та кінцевих користувачів. Це створює сприятливі передумови для гібридного позиціонування системи одночасно як розробницької платформи та сервісу резервного копіювання/синхронізації, що підвищує потенціал масштабування проєкту.

Оцінка ризиків є необхідним етапом формування стартап-проєкту, оскільки вона дозволяє визначити можливі перешкоди, що можуть вплинути на стійкість та

успішність його реалізації. У таблиці 5.6 систематизовано основні фактори ризику та передбачені реакції компанії.

Таблиця 5.6 — Фактори ризику

| Фактор | Зміст ризику | Можлива реакція компанії |
|---|---|---|
| Низьке залучення користувачів | Користувачі можуть віддавати перевагу звичним централізованим сервісам через інерційність або недовіру до P2P-рішень | Спрощення встановлення, покращення UX, створення демо-сценаріїв, розвиток документації та активної підтримки спільноти |
| Сильна конкуренція | Наявність потужних конкурентів у вигляді хмарних сервісів та популярних децентралізованих платформ | Фокус на унікальних перевагах (самохостинг, прозора архітектура, комбінована модель платформ/backup), стратегічні партнерства |
| Інциденти безпеки або втрата даних | Уразливості чи помилки можуть знизити довіру до продукту | Регулярні аудити, використання перевірених бібліотек, рекомендації з безпечного розгортання, вбудовані механізми резервування |
| Обмежені фінансові та людські ресурси | Недостатність ресурсів може уповільнити розвиток продукту | Пріоритизація критичних функцій, залучення open-source спільноти, пошук грантів або партнерств |

Проведений аналіз ризиків демонструє, що попри специфічні виклики децентралізованих систем, більшість ідентифікованих ризиків можуть бути пом’якшені коштом правильного технічного планування та організаційних підходів. Найсуттєвішими загрозами є конкуренція з боку великих гравців, а також можливі побоювання користувачів щодо складності P2P-рішень.

Водночас ці ризики нівелюються за рахунок унікальних характеристик продукту: самохостингу, прозорої контент-адресації, відкритого коду та низького порогу входу для розробників. У сукупності це дозволяє стверджувати, що ризиковий профіль проєкту є прийнятним і не створює критичних бар’єрів для його успішної комерціалізації.

Окрім ризиків, для стартап-проєкту важливо ідентифікувати зовнішні можливості, які можуть прискорити його розвиток та посилити конкурентні позиції. До таких можливостей належать технологічні, ринкові та регуляторні тенденції, що створюють сприятливий фон для впровадження децентралізованих рішень зберігання даних. У таблиці 5.7 наведено ключові можливості та передбачені реакції компанії.

Таблиця 5.7 — Фактори можливостей

| Фактор | Зміст можливості | Можлива реакція компанії |
|---|---|---|
| Розвиток Web3 та децентралізованих застосунків | Зростання кількості проєктів, яким потрібне зручне та надійне сховище для контенту й метаданих без складної токеноміки | Розроблення SDK та прикладів інтеграції, участь у хакатонах і Web3-подіях, партнерства з окремими проєктами як референс-кейсами |
| Зростання попиту на суверенітет даних | Організації прагнуть контролювати місце зберігання, рух та обробку даних, уникаючи повної залежності від глобальних провайдерів | Позиціонування продукту як самохостингової платформи, підготовка матеріалів щодо відповідності вимогам конфіденційності та локалізації даних |
| Збільшення вартості хмарних сервісів | Підвищення тарифів та прихованих витрат стимулює пошук більш економічних моделей зберігання | Демонстрація сценаріїв повторного використання наявної інфраструктури, |

Кінець таблиці 5.7

| | | |
|---|---|---|
| Збільшення вартості хмарних сервісів | Підвищення тарифів та прихованих витрат стимулює пошук більш економічних моделей зберігання | Демонстрація сценаріїв повторного використання наявної інфраструктури, |
| Поширення open-source підходів | Багато організацій надають перевагу відкритим, прозорим технологіям з можливістю зовнішнього аудиту | Підтримка відкритої розробки, прозоре управління проєктом, пропозиція професійної підтримки та консалтингу для корпоративних користувачів |

Перелічені можливості свідчать, що зовнішнє середовище загалом є сприятливим для розвитку децентралізованого сховища даних із акцентом на самохостинг та відкритий код.

Розвиток Web3 та пов’язаної інфраструктури формує додатковий сегмент попиту, де потрібні гнучкі, програмовані сховища без обов’язкової прив’язки до конкретної блокчейн-платформи.

Посилення уваги до суверенітету даних та зростання вартості традиційних хмарних сервісів роблять гібридні та самохостингові рішення економічно й регуляторно привабливими для організацій різного масштабу.

Поширення open-source-підходів, у свою чергу, створює додатковий канал довіри, оскільки потенційні користувачі можуть самостійно перевіряти якість і безпеку програмного забезпечення. За умови цілеспрямованого використання цих можливостей через партнерства, участь у професійних спільнотах і якісну комунікацію ціннісної пропозиції, стартап здатен суттєво посилити свою ринкову позицію та прискорити масштабування.

Для повного розуміння ринкової позиції стартап-проєкту необхідно оцінити особливості конкурентного середовища, у якому він функціонуватиме. Конкуренція в сегменті децентралізованих та гібридних систем зберігання даних не є однорідною: вона формується впливом різних груп гравців — від великих

хмарних провайдерів до децентралізованих мереж і нішевих open-source рішень. Ступінчастий аналіз дозволяє послідовно висвітлити ключові ознаки цього середовища, охарактеризувати їхні прояви та оцінити можливі наслідки для стратегії компанії. У таблиці 5.8 наведено систематизовані результати такого аналізу, що формують підґрунтя для визначення конкурентних переваг, загроз і напрямів адаптації ринкової стратегії продукту.

Таблиця 5.8 — Ступеневий аналіз конкуренції на ринку

| Ознака конкурентного середовища | Прояв характеристики | Вплив на діяльність компанії |
|---|---|---|
| Тип конкуренції: монополістична | Наявність кількох сильних міжнародних гравців із частково схожими послугами | Акцент на унікальних перевагах: самохостинг, контроль над даними, простота для розробників |
| Масштаб конкуренції: міжнародний | Рішення конкурентів доступні глобально, активно просуваються у міжнародних спільнотах | Підготовка англомовної документації, участь у світових конференціях, створення пілотів у різних країнах |
| Галузевий характер: міжгалузевий | Застосування у наукових, комерційних, креативних, державних та IT-секторах | Розробка галузевих кейсів, адаптація технічних матеріалів під різні сегменти |
| Тип конкуренції за продуктом | Конкуруючі платформи різняться архітектурою, бізнес-моделлю та складністю інтеграції | Підкреслення відкритої архітектури, відсутності обов’язкових оплати чи токеноміки |
| Характер конкурентних переваг: нецінові | Ключові критерії — надійність, безпека, простота та екосистема | Інвестиції у якість, безпеку, документацію, не орієнтація на цінові війни |

Кінець таблиці 5.8

| | | |
|---|---|---|
| Значення бренду та репутації | Користувачі віддають перевагу продуктам із сильним іміджем та перевіреним досвідом | Створення репутації через відкритий розвиток, швидку підтримку та демонстрацію успішних впроваджень |

Результати ступінчастого аналізу свідчать, що конкурентне середовище для запропонованого стартапу є складним, але керованим. На ринку присутні сильні гравці, проте вони здебільшого працюють у моделях, орієнтованих або на комерційні хмарні сервіси, або на складні децентралізовані екосистеми з економічними стимулами. Це залишає відкритою нішу для рішень, що поєднують прозорість, простоту використання, самохостинг і програмованість без прив’язки до токеноекономіки.

Водночас міжнародний масштаб конкуренції вимагає від компанії активної присутності у глобальних технологічних спільнотах і високої якості комунікаційних матеріалів. Неціновий характер конкурентних переваг підкреслює важливість технічної надійності та відкритого розвитку продукту. Загалом аналіз демонструє, що за умови правильно вибраної стратегії позиціонування компанія може ефективно змагатися у багатосегментному й динамічному ринковому середовищі.

Для формування стійкої ринкової позиції стартап-проєкту необхідно визначити ключові фактори конкурентоспроможності, що дають змогу продукту вирізнятися серед альтернативних рішень. Ці фактори охоплюють як технічні характеристики системи, так і властивості, пов’язані з її використанням, підтримкою та впровадженням. У таблиці 5.9 наведено основні конкурентні переваги проєкту разом із коротким обґрунтуванням їх значущості для ринку децентралізованих систем зберігання даних.

Аналіз наведених конкурентних факторів свідчить, що проєкт має низку чітко окреслених переваг, релевантних як для технічних, так і для організаційних клієнтів.

Таблиця 5.9 — Обґрунтування факторів конкурентоспроможності

| Фактор конкурентоспроможності | Обґрунтування |
|---|---|
| Самохостинг | Забезпечує повний контроль над даними та інфраструктурою, мінімізує залежність від централізованих провайдерів і зменшує ризики vendor lock-in |
| Орієнтація на розробників та резервне копіювання | Поєднання API-орієнтованої платформи та інструментів резервного копіювання/синхронізації охоплює широку аудиторію — як технічних команд, так і малих організацій |
| Адресація вмісту та верифікованість | Механізм адресованості вмісту забезпечує криптографічну перевірку даних, що є критичним для аудиту, відтворюваності та довгострокового зберігання |
| Модульність і відкритий код | Полегшує інтеграцію та розширення, сприяє залученню спільноти, спрощує зовнішній аудит безпеки та підвищує довіру до продукту |
| Низькі вимоги до ресурсів | Дозволяє використовувати наявну інфраструктуру (звичайні сервери та робочі станції), що зменшує поріг входу та операційні витрати |

Самохостинг та відкритий код забезпечують високий рівень довіри та контрольованості, які важко досягти у централізованих рішеннях. Адресація вмісту та верифікованість даних формують технічну основу для роботи з критично важливою інформацією, де потрібна гарантія цілісності. Поєднання розробницького потенціалу платформи з функціоналом резервного копіювання і синхронізації створює додаткову універсальність, що підвищує привабливість продукту в різних сегментах. Низькі вимоги до ресурсів, у свою чергу, знижують бар’єри входу для малих і середніх команд, сприяючи швидшому поширенню

технології. У сукупності це дозволяє розглядати проєкт як конкурентоспроможний у своєму ринковому сегменті, особливо в нішах, орієнтованих на суверенність даних, гнучкість архітектури та прозорість технологій.

Для кількісного порівняння запропонованого стартап-проєкту з основними конкурентами доцільно використати спрощену 20-бальну модель. Кожен фактор конкурентоспроможності оцінюється за шкалою від 1 до 20 балів для самого проєкту (чим вищий бал, тим кращий результат), а також порівнюється із середнім рівнем конкурентів за шкалою від −3 до +3. Від’ємні значення свідчать про відставання конкурентів від проєкту, нуль означає приблизну рівність, а додатні значення — перевагу конкурентів. У таблиці 5.10 наведено узагальнені результати такого порівняльного аналізу для ключових факторів.

Таблиця 5.10 – Порівняльний аналіз сильних і слабких сторін проєкту

| Фактор конкурентоспроможності | Оцінка проєкту (1–20) | −3 | −2 | −1 | 0 | +1 | +2 | +3 |
|---|---|---|---|---|---|---|---|---|
| Простота розгортання та використання | 17 | | | + | | | | |
| Вимоги до апаратних ресурсів | 16 | | | + | | | | |
| Модульність та розширюваність | 18 | | | | | | + | |
| Бренд та екосистема | 12 | | | | + | | | |
| Гарантії довгострокового зберігання | 10 | | | | | | | + |

Отримані результати кількісного порівняння підтверджують, що найбільш вираженими сильними сторонами проєкту є простота розгортання, помірні вимоги до апаратних ресурсів і висока модульність системи. У цих аспектах він випереджає середній рівень конкурентів, що особливо важливо для малих команд і організацій, які не мають значних інфраструктурних ресурсів.

Водночас порівняно низький бал за параметром «бренд та екосистема» є очікуваним для ранньої стадії стартапу і відображає потребу в подальшому розвитку спільноти, партнерств та публічних кейсів. Найсуттєвіша відносна слабкість спостерігається в напрямі гарантій довгострокового зберігання даних. Таким чином, стратегічним завданням стає збереження та підсилення наявних переваг (простота, легковагість, модульність) при поетапному посиленні аспектів, пов’язаних із надійністю довгострокового зберігання та формуванням зрілої екосистеми навколо продукту.

SWOT-аналіз є узагальнювальним інструментом стратегічного планування, що дозволяє системно оцінити внутрішні характеристики проєкту (сильні та слабкі сторони) та зовнішні чинники (можливості й загрози), які можуть вплинути на його подальший розвиток. Такий підхід забезпечує комплексне бачення ринкової позиції стартап-продукту та полегшує формування ефективної стратегії виходу на ринок. У таблиці 5.11 представлено структуровану оцінку відповідних факторів

Таблиця 5.11 – SWOT-аналіз проєкту

| Сильні сторони (Strengths) | Слабкі сторони (Weaknesses) |
|---|---|
| Самохостинг і повний контроль над даними; адресація вмісту; просте розгортання; низькі вимоги до інфраструктури; орієнтація на розробників; відкритий вихідний код і прозора архітектура. | Відсутність вбудованих економічних стимулів для зовнішніх вузлів; низька впізнаваність бренду на ранньому етапі; обмежені ресурси для маркетингу та масштабування. |
| Можливості (Opportunities) | Загрози (Threats) |
| Зростання попиту на суверенність та контроль над даними; поширення Web3 та децентралізованої інфраструктури; підвищення вартості хмарних рішень | Сильна конкуренція з боку великих хмарних і децентралізованих екосистем; ризик безпекових інцидентів, що можуть підірвати довіру; постійна потреба до адаптації |

SWOT-аналіз демонструє, що проєкт має чітко виражені сильні сторони, орієнтовані на цінності сучасних клієнтів: контрольованість, відкритість, легковагу та простоту роботи з даними. Адресація за вмістом, відкритий код та можливість самохостингу створюють фундаментальну конкурентну перевагу. Зовнішні можливості — зростання попиту на децентралізовані технології та підвищення вартості централізованих хмар — формують сприятливий ринковий фон для масштабування продукту.

Разом із тим слабкі сторони, такі як відсутність економічних стимулів для сторонніх вузлів і недостатня пізнаваність бренду, вимагають додаткової уваги. Загрози з боку сильних конкурентів і швидкість технологічних змін підкреслюють необхідність гнучкого, поетапного розвитку.

Вибір стратегії виходу на ринок є ключовим етапом формування бізнес-моделі стартапу, оскільки від цього залежить як динаміка розвитку продукту, так і обсяги ресурсів, необхідних для його підтримки та масштабування. Зважаючи на технічні властивості проєкту, його орієнтацію на самохостинг і відкритий код, можливі різні варіанти ринкової поведінки — від повільного органічного зростання до агресивного масштабування через інвестиції чи партнерства. У таблиці 5.12 наведено ключові альтернативи виходу на ринок із оцінкою ймовірності отримання ресурсів та орієнтовними строками реалізації кожної стратегії.

Таблиця 5.12 – Альтернативи виходу стартап-проєкту на ринок

| Альтернатива ринкової поведінки | Ймовірність отримання ресурсів | Строки реалізації |
|---|---|---|
| Органічне зростання як open-source проєкту (bootstrap) | Висока — опора на спільноту та низькі витрати | Коротко- та середньострокова перспектива (1–3 роки) |
| Залучення інвестицій для розвитку керованого сервісу | Середня — залежить від демонстрації попиту | Середньострокова перспектива (1–2 роки) |

Кінець таблиці 5.12

| | | |
|---|---|---|
| Стратегічне партнерство з інфраструктурним або телеком-провайдером | Середня/низька — потребує збігу інтересів та доведеної технічної цінності | Середньо- та довгострокова перспектива (2–4 роки) |
| Зростання через платну технічну підтримку та консалтинг | Середня — можливе за наявності перших корпоративних клієнтів | Може розпочатися одразу після успішних пілотних впроваджень |
| white-label ліцензування технології | Низька/середня — залежить від упізнаваності бренду та зрілості екосистеми | Довгострокова перспектива (3+ роки) |

Порівняння альтернатив виходу на ринок показує, що проєкт може комбінувати декілька стратегій залежно від доступних ресурсів та динаміки попиту. Найбільш реалістичним на ранніх етапах є органічне зростання через open-source модель, оскільки вона вимагає мінімальних фінансових витрат і водночас сприяє формуванню довіри з боку технічної спільноти. Паралельно може розвиватися напрям забезпечення платної технічної підтримки, що дозволить отримати перші доходи та стабілізувати розвиток продукту. Інвестиційна стратегія є перспективною, але потребує доведення ринкової придатності рішення. Партнерства та white-label ліцензування відкривають можливості для масштабування, проте більш ймовірні на пізніших стадіях, коли продукт матиме сформовану репутацію та перевірену цінність.

## 5.4 Розробка ринкової стратегії

Ринкова стратегія стартап-проєкту визначає, на яких користувачів буде орієнтовано продукт, яким чином здійснюватиметься вихід на обрані сегменти та які позиції планується посісти у конкурентному середовищі. Для децентралізованого сховища, що поєднує функції платформи для розробників та

інструменту резервного копіювання й синхронізації, особливо важливим є виважений вибір цільових груп, здатних сприйняти нову технологію, а також формування такої моделі поведінки на ринку, яка забезпечить поетапне зростання без надмірних ризиків.

У межах ринкової стратегії послідовно окреслюються цільові групи потенційних користувачів, оцінюється їхня готовність сприймати запропоноване рішення, характер формованого попиту, рівень конкуренції та бар’єри входу в кожен сегмент. У таблиці 5.13 узагальнено основні профілі цільових груп, а також їхню орієнтовну готовність сприйняти проєкт, характер попиту, інтенсивність конкуренції та складність виходу в кожен сегмент.

Таблиця 5.13 – Вибір цільової групи потенційних користувачів

| Опис профілю цільової групи | Готовність користувача сприйняти проєкт | Орієнтований попит | Інтенсивність конкуренції | Складність входу |
|---|---|---|---|---|
| Розробники ПЗ, DevOps-команди, Web3-проєкти | Висока | Стабільний попит на інструменти автоматизації | Середня | Середня |
| Малі та середні підприємства, SaaS-провайдери | Середня | Попит на резервне копіювання та зниження витрат на хмару | Висока | Середня |
| Наукові та установи, ініціативи відкритих даних | Середньо–висока | Попит на довгострокове зберігання великих масивів даних | Середня | Середня |

Кінець таблиці 5.13

| | | | | |
|---|---|---|---|---|
| Технічно підковані індивідуальні користувачі, малі команди | Висока | Попит на безпечну синхронізацію, спільний доступ і самохостинг | Низько-середня | Низька |

Аналіз вибору цільових груп показує, що найпривабливішими сегментами для початкового виходу на ринок є, з одного боку, розробники та DevOps-команди, а з іншого — технічно підготовлені індивідуальні користувачі та малі колективи. Обидві групи характеризуються високою готовністю сприймати нові технології, толерантністю до самостійного розгортання систем і достатньо низькою складністю входу за умови наявності зручних інструментів та документації.

Розробка ринкової стратегії вимагає вибору такої моделі поведінки на ринку, яка відповідає ресурсним можливостям стартапу, особливостям продукту та характеристикам цільових сегментів. У таблиці 5.14 узагальнено основні альтернативи та сформульовано базову стратегію розвитку, що найкраще відповідає специфіці стартап-проєкту.

Таблиця 5.14 – Визначення базової стратегії розвитку

| Альтернатива розвитку проєкту | Стратегія охоплення ринку | Ключові конкурентоспроможні позиції відповідно до обраної альтернативи | Базова стратегія розвитку |
|---|---|---|---|
| Швидке масштабування через агресивний вихід на масовий ринок | Широке охоплення, орієнтація на велику кількість користувачів у короткий термін | Висока впізнаваність бренду, значні маркетингові бюджети, швидке нарощування інфраструктури | Для поточного етапу малореалістична через обмежені ресурси; |

Кінець таблиці 5.14

| | | | |
|---|---|---|---|
| Нішеве позиціонування у вузьких сегментах | Вибіркове охоплення окремих сегментів з високою готовністю до впровадження | Самохостинг, відкритий код, простота інтеграції, орієнтація на розробників, висока технічна гнучкість | Реалістична стратегія початкового етапу: формування сильних позицій у нішевих спільнотах і поступове розширення присутності |
| Орієнтація на корпоративні та інституційні ринки | Глибоке опрацювання обмеженої кількості клієнтів із високою цінністю контракту | Можливість самостійного розгортання, контроль над даними | Доцільна як наступний етап після накопичення довіри до продукту |

Порівняння альтернатив розвитку проєкту вказує, що найменш ризиковим і водночас перспективним є комбінований, поетапний підхід. Агресивне виходження на масовий ринок не відповідає наявним ресурсам і ступеню сформованості бренду, тоді як надмірна концентрація лише на корпоративних клієнтах потребує розгалуженої сервісної інфраструктури й високого рівня довіри, які зазвичай формуються протягом тривалого часу. Натомість нішеве позиціонування в спільнотах розробників, наприклад Web3, дозволяє швидше отримати перших користувачів, зібрати зворотний зв’язок і накопичити портфоліо для подальшого переходу до більш консервативних секторів.

Формування конкурентної стратегії дозволяє визначити характер дій компанії у взаємодії з іншими учасниками ринку та способи досягнення стійких конкурентних переваг. У таблиці 5.15 систематизовано ключові параметри, що визначають базову модель конкурентної поведінки стартап-проєкту.

Таблиця 5.15 – Визначення базової стратегії конкурентної поведінки

| Чи є проєкт “першопрохідцем” на ринку? | Чи буде компанія шукати нових користувачів, або забирати існуючих у конкурентів? | Чи буде компанія копіювати основні характеристики товару конкурента? | Стратегія конкурентної поведінки |
|---|---|---|---|
| Ні | Переважно шукати нових | Ні | Диференціація |

Оцінка визначальних аспектів конкурентної поведінки свідчить, що проєкт не позиціонується як абсолютний першопроходець у сфері децентралізованих сховищ. Основний акцент робиться на залученні нових користувачів, яким потрібен контроль над даними, прозора архітектура та простота інтеграції, тобто на сегментах. Стратегія не передбачає копіювання ключових характеристик конкурентів, оскільки їхні бізнес-моделі не відповідають технічним принципам проєкту. Натомість компанія формує конкурентну поведінку на основі диференціації, відкритого коду, гнучкої архітектури та орієнтації на спільноту.

Стратегія позиціонування визначає, як саме стартап-проєкт буде сприйматися цільовою аудиторією відносно наявних альтернативних рішень.

Таблиця 5.16 – Визначення стратегії позиціонування

| Вимоги до товару цільової аудиторії | Базова стратегія розвитку | Ключові конкурентоспроможні позиції власного стартап-проєкту | Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту |
|---|---|---|---|
| Надійність, контроль над даними, простота інтеграції | Стратегія диференціації | Самохостинг; адресація за вмістом; низькі вимоги до ресурсів | Контроль, надійність, простота |

Запропонована структура показує, що стратегія позиціонування стартап-проєкту базується на трьох головних блоках очікувань цільової аудиторії: надійність, контроль над даними та простота інтеграції

## 5.5 Розробка маркетингової стратегії

Розробка маркетингової стратегії потребує чіткого визначення того, які саме потреби користувачів покликаний задовольнити продукт, яку вигоду отримує кожна цільова група та які конкурентні переваги формують цінність рішення порівняно з альтернативами. У таблиці 5.17 наведено порівняння потреб користувачів, запропонованих вигод та сильних сторін проєкту, що формують основу маркетингової аргументації.

Аналіз змісту таблиці 5.17 показує, що концепція потенційного товару базується на поєднанні технічної унікальності та практичної корисності. Найважливішими потребами користувачів є контроль над даними, надійність зберігання та простота інтеграції у власні робочі процеси.

Таблиця 5.17 – Визначення ключових переваг концепції потенційного товару

| Потреба | Вигода, яку пропонує товар | Ключові переваги перед конкурентами |
|---|---|---|
| Контроль над даними, відсутність залежності від провайдерів | Повне управління власною інфраструктурою й даними | Самохостинг, відкритий код, прозора архітектура, відсутність vendor lock-in |
| Простота інтеграції та автоматизації | Зручна робота з даними, сумісність із CI/CD та іншими системами | HTTP API, CLI, підтримка скриптів автоматизації, гнучка інтеграція |
| Низька вартість володіння | Можливість роботи на стандартному обладнанні | Невисокі вимоги до ресурсів, відсутність підписок |

Формування цінової політики для стартап-проєкту має враховувати наявні товари-замінники та товари-аналоги та рівень доходів цільових груп споживачів. У таблиці 5.18 узагальнено основні чинники, що впливають на ціновий діапазон, та окреслено рамки доцільного ціноутворення.

Таблиця 5.18 – Визначення меж встановлення ціни

| Рівень цін на товари-замінники | Рівень цін на товари-аналоги | Рівень доходів цільової групи споживачів | Верхня та нижня межа встановлення ціни на товар/послугу |
|---|---|---|---|
| 10–20 $ / місяць | Високий | Середньо–високий | 8–17 $ / місяць |

Для ефективного просування проєкту необхідно врахувати особливості закупівельної поведінки різних груп клієнтів та визначити оптимальну модель збуту.

У таблиці 5.19 систематизовано ключові особливості закупівель, функції збуту, необхідну глибину збутового каналу та відповідну систему організації продажів.

Таблиця 5.19 – Формування системи збуту

| Специфіка закупівельної поведінки цільових клієнтів | Функції збуту, які має виконувати постачальник товару | Глибина каналу збуту | Оптимальна система збуту |
|---|---|---|---|
| Невеликі команди розробників та стартапи, що самостійно шукають IT-рішення онлайн | Онлайн-продаж, надання пробного доступу, супровід розгортання | Прямий збут | Прямий цифровий збут через вебсайт постачальника |
| Малі та середні підприємства з обмеженими ресурсами, | Формування та підтримка партнерської мережі | Середня глибина | Непрямий збут через партнерів-інтеграторів |

Кінець таблиці 5.19

| | | | |
|---|---|---|---|
| Формалізовані процедури закупівель, вимога до довгострокових рішень | Пілотні проєкти, підготовка технічних пропозицій і супровід тендерних процедур | Більша глибина, багаторівневий канал | Прямий збут для ключових замовників |

Концепція маркетингових комунікацій визначає ключові напрями взаємодії зі споживачами, які забезпечують формування обізнаності щодо продукту, підтримку його ринкового позиціювання та стимулювання попиту. Ефективна комунікаційна стратегія повинна враховувати неоднорідність цільових сегментів, різну глибину їх технічної підготовки та специфіку процесу прийняття рішень. У таблиці 5.20 наведено концепцію маркетингових комунікацій.

Таблиця 5.20 — Концепція маркетингової комунікації

| Специфіка поведінки цільових клієнтів | Канали комунікацій цільових клієнтів | Ключові позиції, обрані для позиціювання | Завдання рекламного повідомлення | Концепція рекламного звернення |
|---|---|---|---|---|
| Розробники, що активно шукають технічні інструменти | Професійні форуми, GitHub, Hacker News, технічні конференції | Контроль над даними і простота використання | Донести технічну перевагу | Лаконічний технічне повідомлення з демонстрацією сценаріїв використання |
| Організаційні клієнти, що потребують безпечного резервного копіювання | Бізнес-платформи, консультаційні вебінари | Простота інтеграції та зниження витрат | Підкреслити економічну ефективність та безпечність використання | Демонстрація вигоди у форматі коротких бізнес-історій |

Запропонована концепція маркетингових комунікацій узгоджує зміст і форму повідомлень із двома ключовими сегментами – технічними користувачами та організаційними клієнтами. Диференціація каналів і акцентів позиціонування дає змогу одночасно підкреслити технологічну унікальність рішення

## Висновки до розділу 5

У розділі 5 сформовано концепцію стартап-проєкту децентралізованої платформи зберігання, резервного копіювання та синхронізації даних. Показано, що запропоноване рішення орієнтоване на підвищений контроль над даними, прозорість архітектури, цифрову незалежність користувачів та зниження ризиків, пов’язаних із використанням централізованих хмарних сервісів.

На основі аналізу цільових сегментів, конкурентного середовища, базових стратегій розвитку, позиціонування, ціноутворення, збуту та маркетингових комунікацій обґрунтовано ринкову й маркетингову стратегії стартапу. Зроблено висновок, що продукт має потенціал комерціалізації у сегментах розробників, малих і середніх підприємств, а також наукових організацій, забезпечуючи їм гнучке, безпечне та економічно доцільне рішення для роботи з даними.

# ВИСНОВКИ

Результати даної роботи демонструють, що розподілена система зберігання файлів із адресацією вмісту гарно працює разом з ідеями збереження даних у графо подібних структурах в поєднанні із маршрутизацією за допомогою алгоритму Кадемлія. Разом з тим, явні політики закріплення та очищення забезпечують практичну основу для надійного обміну та зберігання даних у децентралізованій мережі.

Проєкт інтегрує багаторівневу архітектуру, що складається з клієнтських застосунків, реалізації розподіленої хеш-таблиці із використанням алгоритму Кадемлія, модулів фрагментації, зберігання та розповсюдження даних. Запропонована структура вузла, модель мережевої взаємодії, підсистема сховища блоків та механізми керування закріпленнями підтверджують доцільність обраних архітектурних рішень.

Проведені обчислювані експерименти засвідчили, що при збільшенні розміру мережі доступність даних монотонно зростає, а система демонструє передбачувану поведінку під час відмов окремих вузлів. Оцінка затримок та пропускної здатності для різних розмірів файлів і фрагментів показала, що обрана модель фрагментації та паралельного вивантаження забезпечує прийнятний компроміс між накладними витратами на обробку блоків і швидкістю передавання даних. Це підтверджує, що розроблена програмна система може бути використаний як основа для практичних сценаріїв резервного копіювання, розподілу великих файлів та інтеграції з прикладними сервісами.

Окремим результатом роботи є формування стартап-проєкту децентралізованої платформи зберігання, резервного копіювання та синхронізації даних на базі розробленої системи. Проведений технологічний аудит, аналіз ринкових можливостей, конкурентного середовища та сформована ринкова й маркетингова стратегії показують, що запропоноване рішення є перспективним у сегментах розробників, малих та середніх підприємств, а також наукових організацій, для яких критичними є контроль над даними, прозорість архітектури та відсутність залежності від єдиного вендора.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Handling churn in DHT-based Publish/Subscribe systems / A. Chaabane et al. *2012 Third International Conference on the Network of the Future (NOF)*, Tunis, Tunisia, 21–23 November 2012. 2012. URL: https://doi.org/10.1109/nof.2012.6463988 (date of access: 27.11.2025).

2. Concurrent deletion in a distributed content-addressable storage system with global deduplication / P. Strzelczak et al. *Proceedings of the 11th USENIX conference on File and Storage Technologies*. 2013 (date of access: 27.11.2025).

3. Blomer J. A Survey on Distributed File System Technology. *Journal of Physics: Conference Series*. 2015. Vol. 608. P. 012039. URL: https://doi.org/ 10.1088/1742-6596/608/1/012039 (date of access: 27.11.2025).

4. Privacy and encryption best practices | IPFS Docs. *IPFS Documentation | IPFS Docs*. URL: https://docs.ipfs.tech/how-to/privacy-best-practices/ (date of access: 27.11.2025).

5. *Department of Computer Science, University of Toronto*. URL: https://www.cs.toronto.edu/~demke/2227/S.21/Handouts/nfs-sandberg85.pdf (date of access: 27.11.2025).

6. Scale and performance in a distributed file system / J. H. Howard et al. *ACM Transactions on Computer Systems*. 1988. Vol. 6, no. 1. P. 51–81. URL: https://doi.org/10.1145/35037.35059 (date of access: 27.11.2025).

7. Ghemawat S., Gobioff H., Leung S.-T. The Google file system. *ACM SIGOPS Operating Systems Review*. 2003. Vol. 37, no. 5. P. 29-43. URL: https://doi.org/ 10.1145/1165389.945450 (date of access: 27.11.2025).

8. Mishra K. Guide to Big Data Hadoop Distributed File System: A Book for Beginners/intermediate. Independently Published, 2020. 27 p.

9. Benet J. IPFS - Content Addressed, Versioned, P2P File System. 2014. P. 11. URL: https://www.researchgate.net/publication/263930348_IPFS_-_Content_Addressed_Versioned_P2P_File_System (date of access: 27.11.2025).

10. Lim H., Andersen D. G., Kaminsky M. Towards accurate and fast evaluation of multi-stage log-structured designs. *FAST'16: Proceedings of the 14th Usenix Conference on File and Storage Technologies*, 22 February 2016. P. 149–166.

11. Buford J. F., Yu H. Peer-to-Peer Networking and Applications: Synopsis and Research Directions. *Handbook of Peer-to-Peer Networking*. Boston, MA, 2009. P. 3–45. URL: https://doi.org/10.1007/978-0-387-09751-0_1 (date of access: 27.10.2025).

12. Peer-to-Peer (P2P) Networks. *Jenkov.com Tech & Media Labs - Resources for Developers, IT Architects and Technopreneurs*. URL: https://jenkov.com/tutorials/p2p/index.html (date of access: 27.10.2025).

13. Benefiting from Data Mining Techniques in a Hybrid Peer-to-Peer Network / M. Ebrahimi et al. *2008 International Conference on Advanced Computer Theory and Engineering (ICACTE)*, Phuket, Thailand, 20–22 December 2008. 2008. URL: https://doi.org/10.1109/icacte.2008.173 (date of access: 27.11.2025).

14. Simplified hole punching example. URL: https://www.researchgate.net/figure/Simplified-hole-punching-example_fig2_250030374 (date of access: 27.11.2025).

15. Hyunggon Park, Rafit Izhak Ratzin, Mihaela van der Schaar. Peer-to-Peer Networks: Protocols, Cooperation and Competition. *Streaming Media Architectures, Techniques, and Applications: Recent Advances*. Los Angeles, 2010. P. 502.

16. Maymounkov P., Mazières D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *Peer-to-Peer Systems*. Berlin, Heidelberg, 2002. P. 53–65. URL: https://doi.org/10.1007/3-540-45748-8_5 (date of access: 27.10.2025).

17. Merkle Directed Acyclic Graphs (DAG) | IPFS Docs. *IPFS Documentation | IPFS Docs*. URL: https://docs.ipfs.tech/concepts/merkle-dag/#further-resources (date of access: 27.10.2025).

18. Hash Tree. *Wikimedia Commons*. : https://upload.wikimedia.org/wikipedia/commons/9/95/Hash_Tree.svg (date of access: 27.11.2025).

19. Maarten van Steen, Tanenbaum A. S. Distributed Systems. 4th ed. Maarten van Steen, 2023. 683 p.

20. Teiva Harsanyi 100 Go Mistakes and How to Avoid Them. Manning Publications, 2022. 384 c.

21. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, Incorporated, 2017. 616 p.

22. Abdullah Lajam, O., & Ahmed Helmy, T. (2021). Performance Evaluation of IPFS in Private Networks. У *DSDE '21: 2021 4th International Conference on Data Storage and Data Engineering*. ACM. https://doi.org/10.1145/3456146.3456159 (date of access: 27.11.2025)

23. Daniel, E., & Tschorsch, F. (2022). IPFS and Friends: A Qualitative Comparison of Next Generation Peer-to-Peer Data Networks. *IEEE Communications Surveys & Tutorials*, *24*(1), 31–52. https://doi.org/10.1109/comst.2022.3143147 (date of access: 27.11.2025)

24. Shi, R., Cheng, R., Fu, Y., Han, B., Cheng, Y., & Chen, S. (2025). Centralization in the Decentralized Web: Challenges and Opportunities in IPFS Data Management. *WWW '25: The ACM Web Conference 2025* (с. 4068–4076). ACM. https://doi.org/10.1145/3696410.3714627 (date of access: 27.11.2025)

# ДОДАТОК А

**ПРОАКТИВНА РЕПЛІКАЦІЯ ДАНИХ У ПІРИНГОВИХ СИСТЕМАХ З АДРЕСАЦІЮ ВМІСТУ**

Тези доповіді на

III Міжнародну науково-практичну конференцію

«Innovations in Science: From Theoretical Foundations to Practical Impact»

24-26 листопада, 2025 року, Антверпен, Бельгія

Аркушів 2

Київ — 2025

# ПРОАКТИВНА РЕПЛІКАЦІЯ ДАНИХ У ПІРИНГОВИХ СИСТЕМАХ З АДРЕСАЦІЮ ВМІСТУ

**Отрох Сергій**
д.т.н., професор
**Гуменюк Андрій**
здобувач вищої освіти магістерського рівня
Кафедра цифрових технологій в енергетиці
КПІ ім. Ігоря Сікорського, Україна

**Анотація.** У статті розглядається вплив проактивної реплікації на доступність даних у піринговых системах з адресацією вмісту. Увага акцентується на запропонованому підході, що поєднує використання розподіленої хеш-таблиці для маршрутизації вузлів з адресацією вмісту даних та їх розподілене розміщення.

**Ключові слова:** піринговi мережі; адресація вмісту; DHT; проактивна реплікація.

**Введення.**

З ростом обсягів даних, централізовані сховища швидко впираються в обмеження ємності та пропускної здатності, що створює потребу в розподілених механізмах зберігання. Водночас децентралізація ускладнює забезпечення



Proceedings of the 3rd International Scientific and Practical Conference
"Innovations in Science: From Theoretical Foundations to Practical Impact"
November 24-26, 2025
Antwerp, Belgium
EUROPEAN OPEN SCIENCE SPACE

доступності даних. Більшість наявних рішень зупиняються на пасивній моделі до реплікації та здебільшого підкріплюються на опортуністичних кешах. При малому розгортанні мережі це призводить до недостатньої кількості копій та концентрації даних на окремих вузлах.

Пірингові системи з адресацією вмісту, як IPFS чи BitTorrent, покладаються на розподілену хеш-таблицю(DHT) та ідентифікатори контенту для пошуку та вилучення файлів. Навіть публічна IPFS мережа має дуже низький рівень автоматичної реплікації: лише ~2.7% файлів мають більше ніж 5 копій на різних вузлах[1]. Альтернативні підходи інкорпорують активну реплікацію за допомогою передачі копій даних до певних вузлів. Наприклад, система Swarm гарантує, що кожна частина файлу зберігається не тільки автором, але й набором вузлів-сусідів, проте такий підхід потребує окремого механізму координацію[2].

**Мета та задачі дослідження.** Дана робота присвячена впровадженню та аналізу рішення для покращення продуктивності та стійкості піринговых систем з адресацією вмісту. Запропонований підхід не відмовляється від використання DHT для маршрутизації та адресації вмісту, а розширюється проактивною реплікацією даних у поєднанні з гнучкою моделлю закріплення даних.

**Результати дослідження і їх обговорення.**

Для кожного CID у DHT підтримуються записи провайдерів, що дозволяє клієнтам знаходити кілька джерел і паралельно вилучати блоки. Запропонований протокол проактивно розміщує копії блоку на вузлах-сусідах. Оскільки ідентифікатори даних фактично є результатами хеш-функції від їх вмісту, той самий блок завжди опиняється в одній і тій самій частині ключового простору. Вузли, чиї ідентифікатори розташовані поруч, природним чином стають

кандидатами на зберігання копій.

Збереження файлів у мережі працює за допомогою процесу закріплення блоків. Закріплення виступає показником чи може певний блок бути видалений збирачем сміття. Система визначає жорсткі закріплення (hard pins) та м'які(soft pins) із терміном дії(Time-to-live), який подовжується після кожного успішного вилучення блоку. Це дозволяє відрізняти гарячі дані, до яких часто звертаються, від холодних, що тривалий час не використовувались.

Для надання оцінки розробленої системи було створено середовище для наскрізного тестування процесу розміщення та завантаження файлів у мережі з 25 вузлів із змодельованими мережевими затримками та випадковими короткочасними відмовами. Кожне випробування складається із розміщення файлу до мережі та його вилучення випадковим вузлом, відмінним від першого.

Як показують результати — базовий підхід забезпечив 94% успішних вилучень, тоді як запропонований 97–99%, медіанна затримка зменшилась близько на 15%, що пояснюється наявністю кількох провайдерів для більшості блоків.

**Висновки.** У роботі розглянуто проблему забезпечення передбачуваної доступності даних у піринговых системах з адресацією вмісту та запропоновано



Proceedings of the 3rd International Scientific and Practical Conference
"Innovations in Science: From Theoretical Foundations to Practical Impact"
November 24-26, 2025
Antwerp, Belgium
EUROPEAN OPEN SCIENCE SPACE

підхід, що поєднує детерміноване розміщення блоків на основі їх ідентифікаторів та використанням двох типів закріплень (жорстких і м'яких). Система дає змогу підтримувати достатній рівень доступності даних без надмірної координації. Отримані результати свідчать, що запропонований механізм є перспективним для малих приватних мереж.

**Список використаних джерел**

1. Centralization in the Decentralized Web: Challenges and Opportunities in IPFS Data Management / R. Shi et al. WWW '25: The ACM Web Conference 2025, Sydney NSW Australia. New York, NY, USA, 2025. P. 4068–4076. URL: https://doi.org/10.1145/3696410.3714627 (date of access: 13.11.2025).

2. Daniel E., Tschorsch F. IPFS and Friends: A Qualitative Comparison of Next Generation Peer-to-Peer Data Networks. IEEE Communications Surveys & Tutorials. 2022. Vol. 24, no. 1. P. 31–52. URL: https://doi.org/10.1109/comst.2022.3143147 (date of access: 13.11.2025).