Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод балансування для керування даними на базі смарт-контрактів

(тема)

Виконав:
здобувач 2 року навчання,
групи СПм-23-4
Антон ШАПОВАЛ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: зав. каф. Андрій КОВАЛЕНКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Шаповалу Антону Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Метод балансування для керування даними на базі смарт-контрактів

затверджена наказом по університету від " 21 " квітня 2025 р. № 296 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи PHP- інфраструктура Laravel; MySQL; DecStore$
Docker використовувався для вузлів Hyperledger.

4. Перелік питань, що потрібно опрацювати у роботі

1) Аналіз сучасного стану вирішення питань підвищення ефективності розподілу даних і доступом до них в децентралізованих системах.
2) Розподілення даних та балансування в децентралізованих системах.
3) Розробка та дослідження методу балансування для керування даними на базі смарт-
4) Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Огляд літературних джерел та аналіз	22.04.25-29.04.25	
	предметної області досліджень		
2	Вибір та обгрунтування методики дослідження	30.04.25-05.05.25	
3	Вибір інструментальних засобів	06.05.25-09.05.25	
4	Розробка моделей протоколів	10.05.25-21.05.25	
5	Проведення експериментів	22.05.25-02.06.25	
6	Оформлення матеріалів кваліфікаційної роботи	03.06.25-05.06.25	
7	Подання кваліфікаційної роботи керівникові та	06.06.25-09.06.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на	10.06.25-12.06.25	
	рецензування		

Дата видачі завдання " 21 " квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ зав. каф. Андрій КОВАЛЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 90 с., 20 рис., 12 табл., 1 дод., 16 джерел.

СМАРТ-КОНТРАКТ, БЛОКЧЕЙН, ДЕЦЕНТРАЛІЗОВАНА СИСТЕМА, БАЛАНСУВАННЯ, ДЕРЕВО МЕРКЛА.

Об'єктом дослідження є процес балансування для керування даними у децентралізованих інформаційних системах.

Предметом дослідження є методи підвищення продуктивності децентралізованих інформаційних систем на базі смарт-контрактів.

Мета кваліфікаційної роботи полягає у підвищенні продуктивності децентралізованих інформаційних систем на базі смарт-контрактів за рахунок удосконалення методу балансування для керування даними.

У ході виконання кваліфікаційної роботи було проведено аналіз сучасного стану вирішення питань підвищення ефективності розподілу даних і керування доступом до них в децентралізованих системах.

Визначені підходи до розподілу даних в децентралізованих системах та контролю доступу в децентралізованих системах зберігання даних.

Удосконалений та досліджений метод балансування для керування даними на базі смарт-контрактів та проведена оцінка його ефективності.

ABSTRACT

Master’s thesis: 90 pages, 20 figures, 12 tables, 1 appendices, 16 sources.

SMART CONTRACT, BLOCKCHAIN, DECENTRALIZED SYSTEM, BALANCING, MERKLA TREE.

The object of the study is the balancing process for data management in decentralized information systems.

The subject of the study is methods for increasing the productivity of decentralized information systems based on smart contracts.

The purpose of the qualification work is to increase the productivity of decentralized information systems based on smart contracts by improving the balancing method for data management.

During the qualification work, an analysis of the current state of solving the issues of increasing the efficiency of data distribution and controlling access to them in decentralized systems was conducted.

Approaches to data distribution in decentralized systems and access control in decentralized data storage systems are defined.

A balancing method for data management based on smart contracts is improved and researched, and its effectiveness is evaluated.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ІКТ – інфокомунікаційні технології

ПЗ – програмне забезпечення

ЦОД – центр обробки даних

ABAC – контроль доступу на основі атрибутів

DAC – дискреційний контроль доступу

DAO – децентралізована автономна організація

DApp – децентралізований застосунок

EDI – електронний обмін даними

MAC – мандатний контроль доступу

PIP – точка інформації про політику

QoS – Quality of Service

RBAC – управління доступом з урахуванням ролей

RTT – Round trip Time

VC – віртуальний кластер

VD – віртуальний диск

ВСТУП

Інформаційні системи обміну даними на основі блокчейн (DApps) стають дедалі популярнішими. За даними DApps Industry Report 2023, кількість користувачів таких систем на протязі лише одного року зросла в 2 рази і досягла 4 мільйона, а капіталізація ринку DApps застосунків перевищила 100 мільярдів доларів. Особливий розвиток такого роду системи набули у фінансовому секторі та індустрії ігор. При цьому випадки впровадження блокчейн у державних системах також зростають. Основною перевагою DApps, порівняно з централізованими системами, є те, що вони дозволяють організувати довірений обмін інформацій між учасниками, які спочатку один одному не довіряють. Але очевидні й проблеми таких систем – це надмірність даних, швидкість їхньої обробки, а також у деяких випадках підвищене енергоспоживання. Саме необхідність багаторазово дублювати дані, що зберігаються в блокчейн на всіх вузлах системи, є суттєвим обмеженням для впровадження DApps в реальних секторах економки.

Вимоги до обробки даних у сучасних інформаційних системах з часом різко зростають у міру збільшення масштабів цих систем і розміру даних, що ними щодня генеруються. У разі розподілених систем виникає необхідність балансування навантаження між вузлами системи, щоб уникнути проблем зі збільшенням навантаження на окремі вузли системи, кількості даних, що зберігаються в ній, а також кількості вузлів, де ці дані можуть зберігатися.

Рівномірне розподілення даних є складним завданням, оскільки об'єкти мають різні розміри і наявність великої кількості різнорідних об'єктів може призвести до високих вимог до ресурсів всіх вузлів системи (завантаження процесора, кількості виділеної для зберігання пам'яті на носіях), а також до пропускної спроможності між вузлами. Відомі наукові підходи та поширені хмарні платформи не дозволяють балансувати навантаження без багаторазового дублювання даних на вузлах одночасно із збереженням

можливості збільшення вузлів системи та кількості об'єктів у ній (властивість масштабованості системи).

Основним елементом DApps є смарт-контракти, які можуть виконуватися на вузлах блокчейна і по суті є програмами, що самовиконуються. Саме за допомогою смарт-контрактів можна організувати балансування навантаження без участі користувача. Однак у науковій літературі не описані підходи, які б вирішували таке завдання. Насамперед це обумовлено наявністю технічних обмежень мови програмування смарт-контрактів.

Необхідно відзначити, що, як і в будь-якій інформаційній системі, DApps потрібно керувати доступом до об'єктів системи. Що стосується розподіленої системи важливо витрачати цього мінімум ресурсів зі збільшенням кількості об'єктів і вузлів у ній, і навіть зберегти стійкість від збоїв, тобто. вона повинна містити централізованих компонент. Найбільш поширені моделі контролю доступу в інформаційних системах (DAC, RBAC) є гнучкими та зручними для адміністрування доступу до об'єктів, але по суті є централізованими та не підходять для використання у DApps.

У цій роботі пропонуються алгоритми та моделі розподілу даних, які дозволяють реалізувати за їх допомогою DApps і одночасно оминути протиріччя надмірності даних. Також робота присвячена створенню моделі управління доступом до даних, яка мінімізує розмір сховища даних прав доступу до об'єктів DApps, забезпечуючи при цьому здатність збільшувати кількість вузлів та загальний обсяг даних системи без обмежень.

Важливим аспектом впровадження DApps, крім проблеми надмірності даних, є оцінка доцільності впровадження такого роду систем у порівнянні з системами централізованої архітектури. Основним критерієм тут є забезпечення певного рівня надійності (не гірше ніж у централізованих системах) при мінімумі ресурсів. Для коректного порівняння надійності розподілених DApps та централізованих систем необхідно розробити новий

підхід до розрахунку надійності, який враховує особливості роботи балансувальника навантаження, що є одним із завдань цієї роботи.

Розглянутий у цій роботі клас інформаційних систем не підходить задач зберігання об'єктів, потребують безпосередньої обробки всередині DApps (пряма зміна даних усередині системи, запис відео безпосередньо на диски системи тощо). Всі операції читання, запису та видалення об'єктів здійснюються тільки через балансувальник навантаження, а розмір файлу повинен бути заздалегідь відомий і не повинен поступово змінюватися, оскільки розмір файлу є важливим параметром для вибору розташування файлу.

Отже, науково-технічне завдання щодо підвищення продуктивності децентралізованих систем на базі смарт-контрактів є актуальним.

Об'єктом дослідження є процес балансування для керування даними у децентралізованих інформаційних системах.

Предметом дослідження є методи підвищення продуктивності децентралізованих інформаційних систем на базі смарт-контрактів.

Мета кваліфікаційної роботи полягає у підвищенні продуктивності децентралізованих інформаційних систем на базі смарт-контрактів за рахунок удосконалення методу балансування для керування даними.

Для досягнення поставленої мети потрібно вирішити такі основні завдання:

- провести аналіз сучасного стану вирішення питань підвищення ефективності розподілу даних і керування доступом до них в децентралізованих системах;

- визначити підходи до розподілу даних в децентралізованих системах;

- визначити підходи до контролю доступу в децентралізованих системах зберігання даних;

- удосконалити метод балансування для керування даними на базі смарт-контрактів та провести оцінку його ефективності.

Методи дослідження включають теоретико-множинні методи системного аналізу, принципи розподілу ймовірностей та теорії ймовірностей, моделі надійності, теорію ігор, теорію прийняття рішень, методи математичного моделювання.

Наукова новизна дослідження полягає в такому:

удосконалений метод балансування для керування даними на базі смарт-контрактів за рахунок формування гібридної архітектури системи, де частина даних зберігається в блокчейн, а частина в локальних сховищах вузлів системи, що дозволило підвищити продуктивність децентралізованої інформаційної системи.

Практична цінність дослідження полягає в підвищенні продуктивності децентралізованих інформаційних систем на базі смарт-контрактів.

За результатами проведених досліджень була надрукована наукова стаття в фаховому виданні за 123 спеціальністю «Системи управління, навігації та зв’язку» [1].

1 АНАЛІЗ СУЧАСНОГО СТАНУ ВИРІШЕННЯ ПИТАНЬ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОЗПОДІЛУ ДАНИХ І КЕРУВАННЯ ДОСТУПОМ ДО НИХ В ДЕЦЕНТРАЛІЗОВАНИХ СИСТЕМАХ

1.1 Інформаційно-теоретичний аналіз ефективності розподілених систем зберігання даних

У період цифрової трансформації урядам і великим компаніям необхідно обмінюватися конфіденційними даними. Цей процес є частиною розвитку електронного уряду у багатьох країнах, наприклад, коли компанії надають державним органам звіти з метою оцінки податків або ліцензування [6]. Великі корпорації та фінансові установи розробляють власні галузеві рішення для розподіленого та безпечного обміну даними. Електронні документи складають для цих корпорацій важливу частину щоденного обміну даними B2B в Інтернеті. Необхідність обміну електронними документами між організаціями існує початку 1970-х років. У процесі розвитку концепції EDI (електронного обміну даними) перший стандарт було розроблено 1996 року [3].

Перші програмні рішення працювали за принципом p2p, використовуючи безпечні з'єднання (VPN тощо). Пізніше з'явилися рішення з урахуванням VAN (мереж із доданою вартістю), організованих як локальні проміжні вузли, управляючі передачею документів з допомогою різних транспортних протоколів (FTP, AS2 та інших.) [10]. Ця конфігурація досі є найпопулярнішою і зазвичай управляється великою телекомунікаційною компанією, консорціумом провайдерів чи державними установами (наприклад, Peppol, EDF+) [6–8].

Більшість корпорацій створюють власні послуги для управління електронними документами та контролю доступу користувачів за допомогою централізованих рішень. Централізовані рішення складаються з

централізованого сервера, на якому зберігаються необхідні дані, і якого клієнти можуть отримати доступ на основі правил, створених на цьому сервері. Цей сервер обробляє запити різних користувачів, обробляє їх та повертає відповідь на основі параметрів запиту. Централізовані рішення вважаються простими у розгортанні. Вони добре працюють у разі використання з низькими вимогами до ресурсів. Однією з переваг використання централізованих рішень є їхня відносно низька вартість, а також той факт, що ними нескладно керувати, оскільки всі дані розташовані в одному місці. Централізований сервер без застосування технології резервного копіювання призводить до малого необхідного простору для зберігання даних (S_{Output}). Проте централізовані системи щодо вразливіші до атак і загроз, що призводить до критичних наслідків для всієї системи [2, 5].

Якщо сервер зламаний або пошкоджений, дані стають недоступними (що призводить до низької надійності – R). Щоб вирішити цю проблему, багато компаній використовують рішення резервного копіювання (рисунок 1.1), у яких один або кілька серверів резервного копіювання використовуються для дублювання даних, що зберігаються на основному сервері. Хоча це може ефективно підвищити надійність системи, це означає додаткові витрати, пов'язані з додатковим розміром сховища – S.

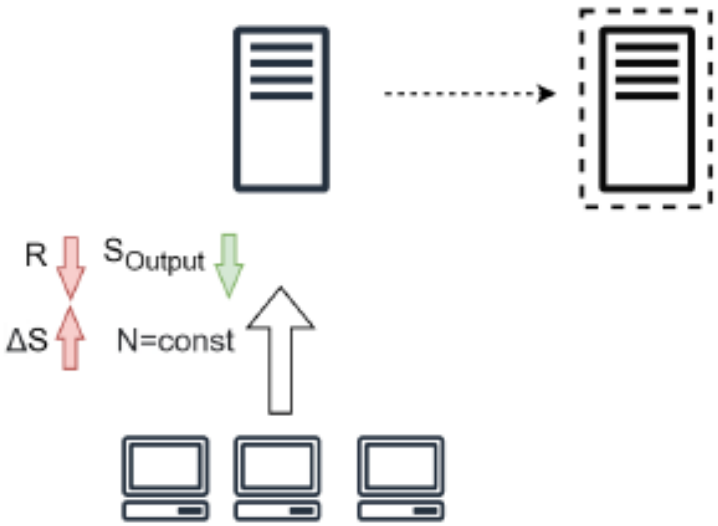

Рисунок 1.1 – Топологія мережі централізованих серверів

Ще одна проблема, від якої страждають централізовані рішення, – масштабування. За останні кілька років обсяг даних, якими обмінюються в Інтернеті різко збільшився. Очікується, що у 2025 році буде генеруватися 463 ЕБ даних на день [2, 5]. Оскільки дані компаній з часом зростають, для зберігання даних може знадобитися використовувати більше одного фізичного файлового сховища. Це означає, що централізовані рішення мають суттєві обмеження, т.к. не можуть збільшувати ємність дискового простору нескінченно (S`).

З іншого боку, децентралізовані системи складні у розгортанні та дороги, але загалом вони надійніші, незважаючи на те, що окремі компоненти таких систем залишаються вразливими. Блокчейн сприймається як одна з перспективних технологій підвищення надійності інформаційних систем (DLT) [6] загалом. Програми, які взаємодіють один з одним або з кінцевим користувачем за допомогою блокчейна замість централізованої бази даних, називаються DApps [6, 9]. Основною перешкодою для впровадження DApp є висока вартість, яка істотно знижує кількість потенційних користувачів [9]. Блокчейн зберігає дані у вигляді транзакцій у блоках. Якщо потрібно оновити деякі дані, що зберігаються, нові дані додаються в нові блоки замість зміни існуючих даних, що гарантує незмінність даних. Історію змін можна відстежувати. Блокчейн використовує механізм, званий механізмом консенсусу [6], який перевіряє транзакції та запобігає зміні даних недовіреними користувачами в одному або деяких вузлах блокчейну.

Найбільш популярними механізмами консенсусу є два: Proof of Work та Proof of Stake [8]. Використання консенсусу та особливість зберігання блоків роблять DApps стійкими до компрометації даних.

Багато блокчейн-платформ дозволяють запускати на своєму боці програмне забезпечення, зване смарт-контрактом [9], яке можна використовувати для управління операціями, пов'язаними з блокчейном. Смарт-контракти можна розглядати як програми, які працюють на стороні

серверів і контролюють керування даними в системі та те, як користувачі взаємодіють із даними. Мережі блокчейна можуть бути загальнодоступними, де всі транзакції видно будь-кому або приватними. Найпопулярнішими публічними мережами блокчейнів є Біткойн [8] та Ethereum [7]. Hyperledger Fabric [7, 8] є прикладом приватних мереж блокчейнів. Ці мережі та платформи набули великої популярності за останні кілька років, і очікується, що їхній ринок зросте набагато більше до 2027 року [5] (рисунок 1.2).

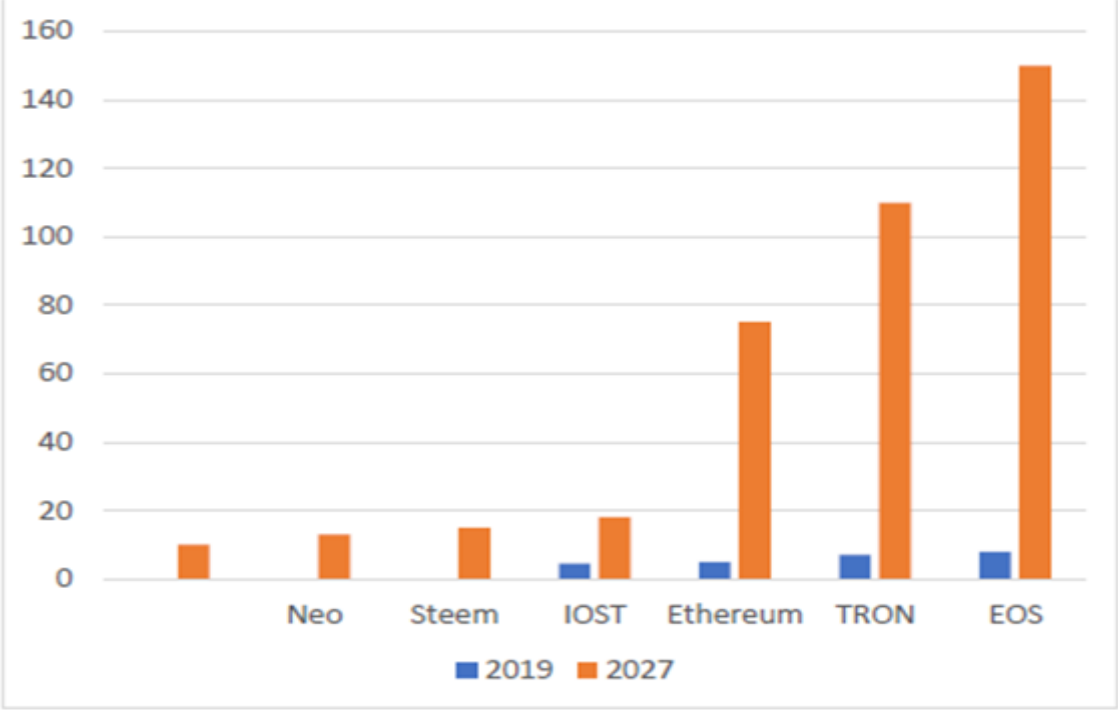

Рисунок 1.2 – Маркет DApps

Блокчейн зберігає ті самі дані на багатьох вузлах, а це означає, що він має високу надмірність (що означає високе значення S_{Output}), що також призводить до високої доступності та надійності (R). На рисунку 1.3 показано, як пов'язані вузли та користувацькі застосунки блокчейна. Крім того, блокчейн не враховує, що різні вузли мають різну ємність зберігання, а отже, не гарантує рівного розподілу даних (ΔS).

Оскільки блокчейн реплікує дані, він зазвичай не використовується для зберігання великих обсягів даних, наприклад, файлів. Натомість для цього

завдання зазвичай використовуються інші типи розподілених систем. Однак розподілені системи розрізняються за різними аспектами, такими як продуктивність, розмір сховища даних, надійність та інші параметри, засновані на різних параметрах, таких як топологія та архітектура, і навіть пов'язані з нею спосіб зберігання. За результатами аналізу у цьому дослідженні виділено кілька параметрів з метою оцінки ефективності систем зберігання [8].

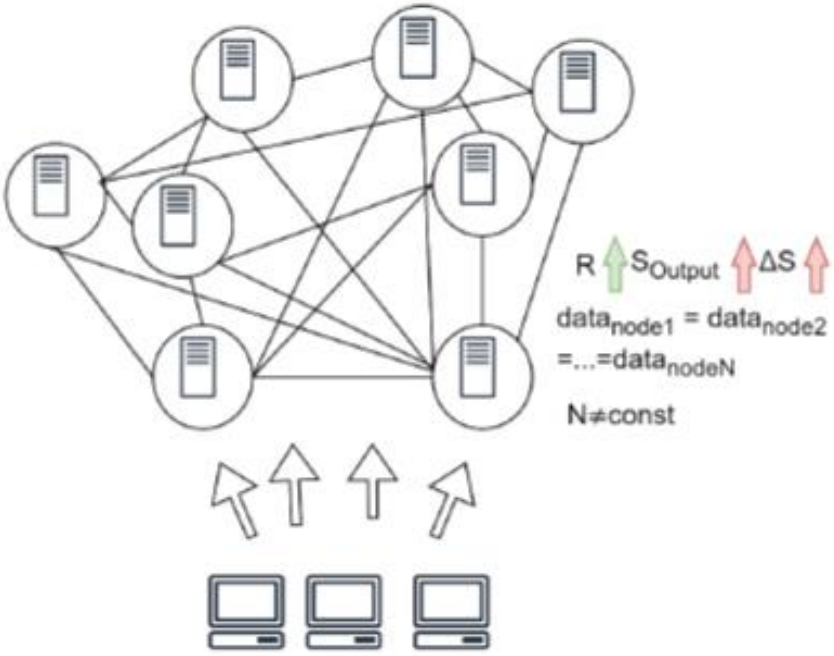

Рисунок 1.3 – Топологія мережі блокчейна

Кількість вузлів у системі вважається важливим показником ефективності системи зберігання [4]. Тут і далі під масштабуванням системи ми розумітимемо можливість нескінченного збільшення вузлів системи та загального доступного для зберігання місця без зміни інших параметрів (насамперед, продуктивності та надійності). В ідеалі системи зберігання повинні бути динамічними та підтримувати масштабування, щоб відповідати зростаючим вимогам до зберігання даних. Це може бути відображено у зміні кількості вузлів, оскільки вона не повинна бути константною ($N \neq const$), тому можна додавати нові вузли для масштабування системи та зберігання

більшої кількості даних. Надійність системи (R) також має бути максимально високою. Більшість систем дублюють дані за допомогою серверів резервного копіювання для їх збільшення, але це призводить до збільшення розміру даних у системі ($SOutput$), що призводить до збільшення вартості. У цьому вся дослідженні розглядається зменшення обсягу даних при одночасному підвищенні надійності всієї системи.

Ще одним важливим параметром системи є обсяг вільної пам'яті у системі (S`). При додаванні нового вузла важливо, щоб у новому вузлі був вільний обсяг пам'яті для зберігання нових даних, а не просто дублювання існуючих даних у системі, як у блокчейні, що робить додавання нових вузлів марним з точки зору збільшення вільної пам'яті в системі. Важливо збільшувати вільний простір у системі S` при додаванні нових вузлів.

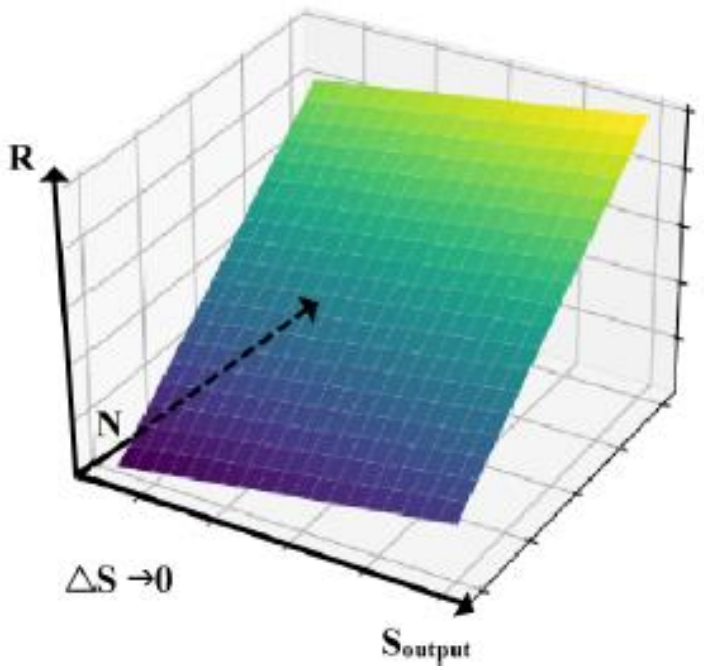

Рисунок 1.4 – Співвідношення між N,R і S_{Ootput} у розподілених системах

При додаванні нових вузлів або видаленні вузлів також важливо в будь-який момент розподіляти ресурси між доступними вузлами, оскільки не повинно бути вузла, який повністю використовується і розглядається як вузьке місце, той час як інші вузли мають невелику відсоток зберігання. З

цієї причини оптимальний розмір сховища кожного вузла повинен бути розрахований SS_i, який повинен враховувати розмір сховища всіх вузлів та розмір сховища вузла і, і на основі цього фактичний розмір сховища на вузлі і (S_i) має бути якомога ближче до $SSi\ Si) \rightarrow 0$: $\forall i \in \boldsymbol{N}$.

Зі збільшенням кількості вузлів має збільшуватись вільний простір, доступний для зберігання нових файлів, а також має підвищуватись надійність системи, як на рисунку 1.4.

1.2 Особливості смарт- контрактів

Смарт-контракти – це програми, що працюють у блокчейні. Вони автоматизують виконання умов між сторонами. Смарт-контракти не потребують посередників або третіх сторін. Це підвищує довіру і знижує витрати.

Рисунок 1.5 – Складові смарт-контракту

Уперше поняття з’явилося в 1994 році. Його ввів криптограф Нік Сабо. Ідея була такою: замінити юридичні договори кодом. Пізніше це реалізували в блокчейн-системах.

Смарт-контракт – це код, який зберігається в блокчейні. Він запускається, коли виконуються певні умови. Після запуску його вже не змінити. Це забезпечує прозорість і безпеку (рисунок 1.5).

Розберемо, яким чином працює смарт-контракт. Сторони погоджуються на умови контракту. Ці умови записуються у вигляді коду. Код публікується в блокчейн-мережі. Контракт чекає на активацію умов. Якщо умови виконано — код запускається. Виконується передбачене: транзакція, передача даних чи інше. Контракт сам перевіряє умови. Ніхто не може втрутитися в процес.

Виділимо головні переваги смарт-контрактів:

- смарт-контракти зменшують людський фактор;
- автоматизація знижує ризик помилок;
- процеси відбуваються швидше і дешевше;
- не потрібні юристи чи нотаріуси;
- вони підвищують прозорість. усі дії видно в блокчейні;
- дані не можна видалити або змінити, це створює довіру між незнайомими сторонами;
- контракти працюють цілодобово: мережа не має вихідних і свят, це особливо корисно для міжнародних операцій.

Але застосування смарт-контрактів також пов’язано з низкою деяких обмежень:

- код – це теж договір, помилки в коді можуть бути критичними;
- якщо є баг – його важко виправити;
- потрібна перевірка коду перед запуском;
- юридичний статус не завжди чіткий, закони не всюди визнають такі контракти – це створює ризики у правовому полі;
- також вони залежать від блокчейн-мережі: якщо мережа уповільнюється, то контракт уповільнюється теж;
- вартість транзакцій іноді зростає.

Смарт-контракти на сьогодні застосовуються у багатьох галузях.

У фінансах вони автоматизують перекази. Вони є основою для DeFi – децентралізованих фінансів. Смарт-контракти керують криптокредитами і біржами без участі банків чи брокерів.

У логістиці смарт-контракти використовуються для відстеження товарів. Кожен етап фіксується у блокчейні, тому забезпечується прозорість доставки.

У страхуванні смарт-контракти використовуються для проведення автоматичних виплат, наприклад, якщо сталося ДТП, то кошти можуть бути у деяких випадках виплачені автоматично, без потреби в перевірці людиною.

У нерухомості смарт-контракти використовуються для складання цифрових угод, а також можуть сприяти здійсненню купівлі-продажу без посередників, тобто документи фіксуються в блокчейні.

У сфері інтелектуальної власності застосування може стосуватися авторських прав, тобто смарт-контракти можуть слідкувати за правами на музику, книги тощо. Кожна транзакція при цьому автоматично фіксується.

Смарт-контракти дозволяють створити DAO – децентралізовану організацію, усі рішення в якій ухвалюють смарт-контракти, кожен учасник має голос, а рішення автоматично набирають чинності.

Розглянемо, яким чином працює DAO. DAO має власний токен. Токени надають право голосу. Кожен учасник голосує за пропозиції. Результати зберігаються у блокчейні. Смарт-контракти реалізують правила управління DAO. Вони контролюють бюджет, доступи та зміни. Без дозволу спільноти змінити нічого не можна. Це забезпечує прозорість та рівність.

Перейдемо до розгляду конкретного прикладу роботи DAO. Припустимо, DAO управляє фондом. Хтось пропонує витратити частину коштів, а інші учасники голосують “за” або “проти”. Якщо більшість “за”, то кошти переказуються автоматично.

Усі транзакції видно в блокчейні. Ніхто не може скасувати рішення більшості. Система не має одного власника або керівника. Це гарантує

децентралізацію влади. Децентралізована організація DAO має багато переваг, серед яких можна виділити такі основні:

- DAO не потребує централізованого контролю, працює без директорів або рад, а учасники самі формують політику, при цьому все автоматизовано, без втручання ззовні;
- DAO мають високу прозорість, тому що кожне рішення доступне для перевірки, а жодні зміни неможливі без колективної згоди;
- DAO працює цілодобово, тому що код не знає вихідних, отже рішення ухвалюються суттєво швидше, ніж у традиційних структурах.

Але DAO мають й низку недоліків, які треба теж враховувати:

- помилки в коді можуть дорого коштувати;
- юридичний статус DAO часто незрозумілий, тому що закони далеко не всюди визнають такі організації, отже це може створити юридичні конфлікти;
- DAO складно змінити після запуску, отже, якщо код застарів, то потрібен новий контракт, що вимагає колективного рішення і голосування.

Наведемо декілька відомих прикладів DAO у дії:

- MakerDAO – децентралізована система кредитування, вона керує стабільною монетою DAI, а її учасники контролюють правила випуску й застав;
- Uniswap DAO – децентралізована біржа, власники токенів UNI голосують за розвиток, рішення автоматично впроваджуються у протокол;
- Aragon дозволяє створювати власні DAO, це платформа для керування спільнотами, де кожен може створити DAO без програмування.

DAO мають широкі перспективи, вони стають дедалі популярнішими. Їх використовують у бізнесі, культурі, технологіях. DAO можуть змінити державне управління, наприклад, можливі місцеві бюджети на блокчейні, де громади самі ухвалюють рішення через голосування.

Розглянемо мови програмування смарт-контрактів. Найпоширеніша серед них – Solidity. Її використовують у мережі Ethereum. Інші мови, що

використовуються, – це Vyper, Rust, Move. Вони мають різні рівні складності. У будь-якому випадку смарт-контракти компілюються у байт-код, який зберігає блокчейн. У разі виклику код виконується нодами та усі користувачі бачать однаковий результат. Особливе питання стосується безпеки смарт-контрактів. Необхідний аудит створеного коду, який здійснює перевірку на вразливості. Аудит повинні проводити незалежні компанії, такі перевірки знижують ризик зломів. Відомі були атаки на контракти, наприклад, атака на DAO в 2016 році, в результаті якої хакери вкрали мільйони доларів.

Майбутнє смарт-контракті маєширoку перспективу. Технологія активно розвивається, з’являються нові блокчейни і стандарти, кКонтракти стають ще безпечнішими. Інтеграція з IoT і AI – наступний крок розвитку.

Отже, смарт-контракти змінюють цифрову економіку, відкривають нові моделі бізнесу.

1.3 Аналіз ефективності використання методів керування доступом на основі блокчейну

Оскільки використання централізованих компонентів знижує надійність системи загалом, необхідно розглянути наукові підходи, що використовують блокчейн для управління контролем доступу та інших метаданих об'єктів у системах.

Управління контролем доступу вважається критично важливим компонентом будь-якої системи обміну конфіденційними даними. Будь то система зберігання файлів, веб-сайт соціальної мережі або будь-яка інша система, що забезпечує доступ до її приватних ресурсів, необхідно керувати тим, як різні сторони можуть отримати доступ до цих ресурсів. Таким чином, система зберігання недостатньо забезпечувати можливість зберігання файлів. Вона необхідна для керування та контролю доступу користувачів до його файлів, інакше він не може бути застосована.

Контроль доступу можна розділити на основні моделі [8]:

1. Мандатний контроль доступу (MAC). Це тип контролю доступу, який має рівні конфіденційності. Прикладом може бути ситуація, коли ресурси класифікуються за грифом секретності {загальні, секретні, абсолютно секретні}. Таким чином, користувачі можуть отримати доступ до ресурсів залежно від наданого їм рівня безпеки.

2. Дискреційний контроль доступу (DAC) [12]: при цьому типі контролю доступу власник або автор ресурсу визначає, кому дозволено доступ до нього. Зазвичай DAC використовує матричну модель доступу [9], яка є таблицею, що описує привілеї в стилі суб'єкта (наприклад: користувача) і об'єкта (наприклад: ресурсу). У соціальних мережах, таких як Facebook [7], користувач може обмежити кількість людей, які можуть отримати доступ і переглянути вміст профілю або конкретних повідомлень, що розглядається як приклад DAC.

3. Управління доступом з урахуванням ролей (RBAC) [5]. При цьому типі керування доступом користувачі мають різні ролі, а доступ до ресурсів здійснюється залежно від їхньої ролі. Наприклад, на веб-сайтах електронної комерції деякі сторінки доступні лише адміністраторам, інші сторінки можуть бути доступні менеджерам та адміністраторам тощо.

4. Управління доступом з урахуванням правил [6]. Цей тип керування доступом визначає правила доступу до ресурсів, які можуть бути різними. Наприклад, деякі банки можуть поставити умову, яка забороняє обмін валюти у нічний час. Зазвичай він використовується у поєднанні з контролем доступу на основі ролей [5].

5. Контроль доступу на основі атрибутів (ABAC) [9]. Він керує доступом шляхом призначення політик для різних атрибутів користувачів, ресурсів та середовища [6].

Важливо, що у сучасних системах зазвичай використовується комбінація різних типів моделей контролю доступу.

Практично всі моделі доступу застосовуються спільно з DAC, оскільки ця модель забезпечує найпростіші базові правила доступу суб'єкт-об'єкт.

Наприклад, іноді потрібно надати конкретному користувачеві доступ до певного ресурсу, не надаючи дозволу іншим користувачам з тією ж роллю або набором атрибутів. Проблема в тому, що реалізація DAC може спричинити створення великої матриці доступу ресурсів та користувачів. У файлових системах зберігання, якщо всього є 1 мільйон файлів і 1000 користувачів, ми отримаємо мільярд записів, що може бути проблематично при розгляді розподілених систем з величезною надмірністю даних, таких як блокчейн. При використанні RBAC потрібно зберігати менше даних, оскільки одна політика може охоплювати багато файлів. Однак результуючий розмір даних може виявитися великим і непридатним для зберігання в блокчейні. Метою даного дослідження є розробка розподіленої моделі управління доступом DAC+RBAC, яка вирішує проблему масштабування для систем на базі блокчейну без централізованих компонентів.

Існує безліч досліджень, у яких обговорюється застосування контролю доступу у блокчейні. Більшість цих досліджень присвячено тому, як реалізувати контроль доступу з використанням блокчейну для конкретної галузі, особливо Інтернету речей [1, 6]. Cheng та ін [2] пропонують розгорнути точки прийняття рішень з політики (PDP) у блокчейні та зберігати точки адміністрування політики (PAP) поза блокчейном, щоб зменшити розмір навантаження на блокчейн. Однак це дослідження не показує, як зберігаються дані поза мережею і як вони реплікуються або розташовуються, щоб запобігти втраті даних у разі втрати сервера зберігання. Таким чином, правила контролю доступу в основному централізовані (поза блокчейном) і доступні децентралізованій системі (блокчейн), яка не є справжньою децентралізованою системою. Крім того, згідно з їх рішенням, коли запити надсилаються до блокчейну, запит перенаправляється на ресурси поза блокчейном для кожного окремого запиту, що може бути неефективно у великомасштабних системах.

Maesa та ін. [7] запропонували гібридний підхід, який дозволяє зберігати політики прямо в блокчейні або посилатися на нього. Нові політики можуть бути створені шляхом створення нових біткоїн-транзакцій певного типу. Для оновлення або відкликання доступу після цього створюється нова транзакція, яка посилається на створену політику, і разом із нею зберігаються нові політики. Існує також особливий тип політик, який називається «Транзакція передачі прав», який дозволяє передавати права від користувача. А користувачеві В. Останній тип введених авторами політик дозволяє обмінюватися правами між власниками, тому він аналогічний передачі прав від користувача А до В та прав від В до А одночасно.

Цей підхід переписує політики, щоб мінімізувати обсяг даних, і зберігає їхню стислу версію в блокчейні. Це рішення може добре працювати з системами доступу ABAC, але воно не підходить для систем DAC, оскільки ці політики базуються на користувачах, а файли політик можуть бути більшими.

Paillisse та ін [8] запропонували розподілену систему, в якій політики доступу зберігаються в блокчейні. Цей підхід пропонує розширення групових політик (GBP) та застосуємо у мережі блокчейн на базі Hyperledger Fabric.

Отже, цифрова трансформація вимагає нових підходів у проектуванні розподілених систем обміну даними та зберігання файлів. Вимоги до зберігання даних зростають, оскільки обмін даними останніми роками зростає в геометричній прогресії.

Централізовані сервери використовуються багатьма організаціями, оскільки вони дешеві та прості у розгортанні. Однак централізовані сервери мають одну або кілька єдиних точок відмови та не підтримують масштабованість (збільшення загального розміру сховища даних без обмежень). Надійність централізованих серверів можна підвищити за рахунок використання серверів резервного копіювання, які дублюють дані, збільшуючи цим загальну вартість. Крім того, сервери резервного

копіювання не вирішують проблему масштабованості, яка вважається вимогою для організацій, де потрібно зберігати величезну кількість файлів.

Блокчейн набув популярності за останні кілька років, оскільки це розподілена система зберігання, може організувати обмін даними між учасниками з нульовою довірою завдяки незмінності даних, що зберігаються на вузлах, своїм алгоритмам консенсусу. Однак на вузлах дублюються дані, що призводить до їх надмірності.

У результаті огляду виділено кілька показників ефективності систем розподілу даних: надмірність даних, рівномірність їх розподілу, надійність та властивість збільшення кількості вузлів та даних на них (масштабованість).

Розподілені системи зберігання різняться за поведінкою залежно від представленої ними моделі зберігання. Аналіз архітектури відомих IT-проектів та підходів показав, що кожен з них не може вирішити одне з трьох завдань одночасно: надмірність даних, рівномірний розподіл даних по вузлах, високу надійність при мінімізації необхідного місця зберігання для зберігання даних та їх резервних копій.

2 РОЗПОДІЛЕННЯ ДАНИХ ТА БАЛАНСУВАННЯ В ДЕЦЕНТРАЛІЗОВАНИХ СИСТЕМАХ

2.1 Формалізована постановка завдання оптимізації розподілу даних у розподіленому середовищі

У таблиці 2.1 наведено перелік параметрів, необхідних для оцінки ефективності систем зберігання.

Таблиця 2.1 – Необхідні показники для аналізу систем

Показник	Символ	Одиниця виміру
Множина вузлів у системі	$\boldsymbol{N}$	Вузол
Надійність системи	R	Відсоток
Об'єм вільного місця, доступного для зберігання об'єктів	$S`$	Байт
Відхилення значення розміру сховища у системі від рівномірного розподілу	ΔS	Байт
Розмір вхідних об'єктів у системі	S_{Input}	Байт
Розмір вихідних об'єктів у системі	S_{Output}	Байт
Розмір даних контролю доступу	$S_{AccessControl}$	Байт
Розмір метаданих	$S_{Metadata}$	Байт
Розмір додаткових даних, що використовуються для підвищення доступності/надійності об'єктів	$S_{Reducancy}$	Байт

Розмір використовуваної системи зберігання можна розрахувати таким чином:

$$S_{Output} = S_{Input} + SR_{educancy} + SM_{etadata} + S_{AccessControl}. \quad (2.1)$$

Однак S_{Input} визначається розміром вхідного об'єкта, тому його неможливо оптимізувати, оскільки користувачі визначають які об'єкти

зберігати. У порівнянні з вхідними об'єктами метадані та дані контролю доступу вважаються невеликими:

$$S_{Metadata} + S_{AccessControl} \ll S_{Input}; \quad (2.2)$$

$$S_{Output} \approx S_{Input} + S_{Reducancy}. \quad (2.3)$$

З урахуванням цих параметрів систему розподілу можна уявити як

$$DistrSystem <S_{Output}, \Delta S, R, \boldsymbol{N}>, \quad (2.4)$$

де $\Delta S = \sum_{i=1}^{N} |\Delta S_i|$, $\Delta S_i = (SS_i - S_i) \rightarrow 0 \quad \forall i \in \boldsymbol{N}$; $SSi = \frac{C_i \cdot S}{S' + S} \quad \forall i \in N$, C_i – розмір сховища на вузлі i. Відповідно, мета дослідження може бути визначена як результат такої оптимізації:

$$max_R min_{SOutput,\, \Delta S}(DistrSystem < S_{Output}, \Delta S, R, \boldsymbol{N}>). \quad (2.5)$$

2.2 Моделі та алгоритми розподілу даних в DApps

Для рішення проблеми надмірності даних в DApps, а також збереження надійності на рівні централізованих систем і збереження властивості масштабованості, пропонується нова модель розподілу. Для її описи розглянемо наступні елементи:

- архітектура системи на базі моделі, що описує, які компоненти існують у системі, як вони розподілені і як взаємодіють друг з другом;
- алгоритми балансування даних, які використовуються для розподілу даних між вузлами в будь-який момент часу, щоб зробити систему збалансованою і забезпечити необхідний рівень надійність, розподіл даних оптимізується во час різних подій, таких як додавання нових вузлів зберігання або видалення вузлів тощо.

Архітектура типової DApps системи складається із частин, наведених на рисунку 2.1.

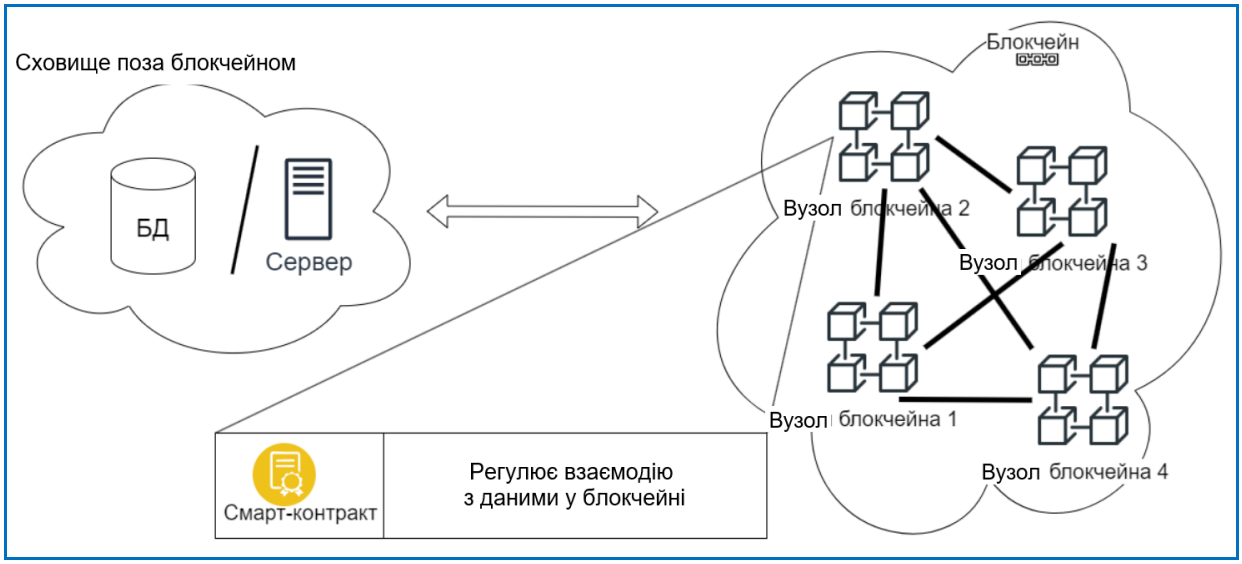

Рисунок 2.1 – Архітектура DApps системи

Важливо, що технологія кодування з корекцією стирань адаптована і розгорнута в системі зберігання. Кодування з корекцією стирань – це тип кодування, який приймає k вхідних даних однаковою довжини, які можуть бути потоком даних або сигналів і т. д., та видає на виході n сигналів, створених на основі вхідних сигналів і функції стираючого кодування. На основі підмножини вхідних даних і вихідних даних (k`) можна отримати всю виключену підмножину вхідних даних, як показано на рисунку 2.2.

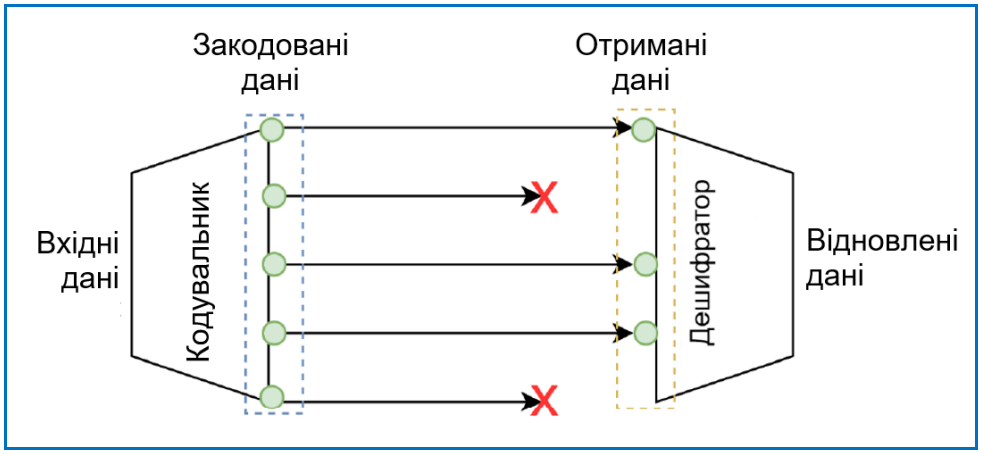

Рисунок 2.2 – Модель кодування з корекцією стирань

Функція XOR розглядається як приклад функції стирального кодування. У XOR, як і в таблиці 2.2, видно, що довжина вихідного сигналу дорівнює довжині першого і другого вхідних сигналів. Якщо B втрачено, його можна відновити, використовуючи вихід XOR і A.

Таблиця 2.2 – Функция XOR

Входи		Виходи
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Наприклад, якщо відомо, що A дорівнює 0, а вихід дорівнює 1, це означає, що B повинно бути 1. З іншого боку, якщо A втрачено, а B все ще існує, можна знайти A, використовуючи B, і вихід XOR, використовуючи ту ж логіку. Виходячи з цього, можна відновити або A, або B, використовуючи вихідні дані функції стирального кодування і вхідний сигнал того ж розміру, що залишився, замість реплікації A і B, що призводить до зменшення необхідного обсягу пам'яті/пропускної здатності на 25%.

Архітектуру децентралізованої системи зберігання, що використовує блокчейн для децентралізованого управління, можна уявити таким чином, як це зображено на рисунку 2.3.

Існує кілька алгоритмів, які використовують спеціальні кодування для мінімізації розміру даних. RAID-5 і RAID-6 вважаються одними з самих популярних реалізацій стираючого кодування. Інші типи RAID не розглядаються, оскільки вони менше популярні або засновані на цих реалізаціях, таких як комбінація використання стираючого кодування з чергуванням даних по групам дисків (RAID-50).

1. Балансувальник навантаження. Балансувальник навантаження забезпечує необхідні функції управління і розподілу даних. Воно відповідає за наступні завдання:

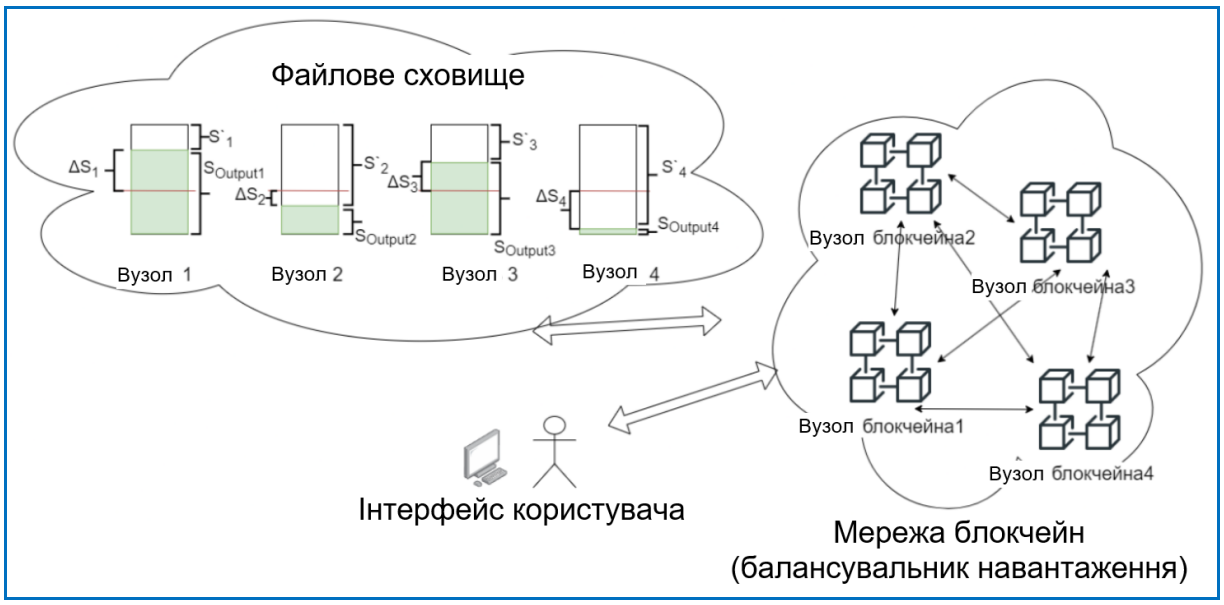

Рисунок 2.3 – Архітектура децентралізованої системи розподілу даних на основі блокчейну

- управління вузлами зберігання: управління існуючими вузлами, контроль процесів приєднання та виходу до вузлів тощо;

- управління даними: управління розміщенням існуючих файлів (або об'єктів) і визначення місця додавання нових файлів. Пропонована модель розподілу використовує віртуальні об'єкти зберігання, звані віртуальними кластерами (VC) і віртуальними дисками (VD), які будуть пояснено в пункті «Вузли зберігання». Балансувальник навантаження відповідає за процеси, пов'язані з цими віртуальними об'єктами, наприклад за їх поширення. Розроблено чотири алгоритму розподілу даних між вузлами;

- управління користувачами: додавання і видалення користувачів, а також управління контролем доступу.

Балансувальник навантаження у багатьох системах, таких як Lustre і HDFS, представлений одним вузлом. Деякі системи, такі як Ceph, використовують кілька резервних вузлів для балансування навантаження, тому, якщо основний балансувальник навантаження виходить із ладу, один із резервних вузлів використовується в якості балансувальника навантаження. Хоча цей метод допомагає уникнути єдиних точок відмови, основний вузол

балансування навантаження як і раніше вважається вузьким місцем, оскільки все користувачі повинні підключатися до одному і тому ж вузлу. У пропонованій системі блокчейн використовується для запуску балансувальника навантаження в формі смарт-контракт. У цьому випадку користувачі можуть підключитися до будь-кому вузлу системи блокчейн, і все вузли синхронізуються через алгоритм консенсусу.

2. Вузли зберігання. Система зберігання об'єктів системи (у узагальненому випадку – файлів) складається з вузлів (однорангова мережа P2P, де все вузли мають однакову роль). Файли організовані в віртуальні кластери (VC). У свою чергу, кожен з яких складається з трьох віртуальних дисків фіксованого розміру (VD). Це зроблено для того, щоб при збільшенні VC і файлів в системі на основі аналізованої моделі час роботи алгоритмів управління розподілом файлів по віртуальним дискам не залежало від спільної кількості файлів у системі. Очевидно, що кількість VD в системі буде набагато менше кількості файлів у ній. Важливість мінімізації складності роботи алгоритмів зростає, оскільки пропоновані алгоритми реалізовані на смарт - контракти. Коли файл завантажується користувачем або зовнішньої інформаційної системою в систему зберігання, він розбивається на дві частини і розраховується відповідний пральний код.

Важливо відзначити, що розмір стираючого коду дорівнює розміром половини файлу. У кожному кластері віртуальні диски організовані як індексований масив. Перший VD використовується для зберігання перших половин файлів, другий VD для відповідних друге половин і третій для зберігання стираючого коду.

Така схема розподілу даних наводить до мінімізації необхідного сховища резервних даних на 25% по порівнянні з системами зберігання з одним сервером резервного копіювання. на рисунку 2.4 представлена концепція пропонованої моделі. VC6 складається з 3 - х віртуальних дисків. Перший віртуальний диск в VC6 розташований у вузлі 2, другий віртуальний диск - у вузлі 6, а останній - у вузлі 4 .

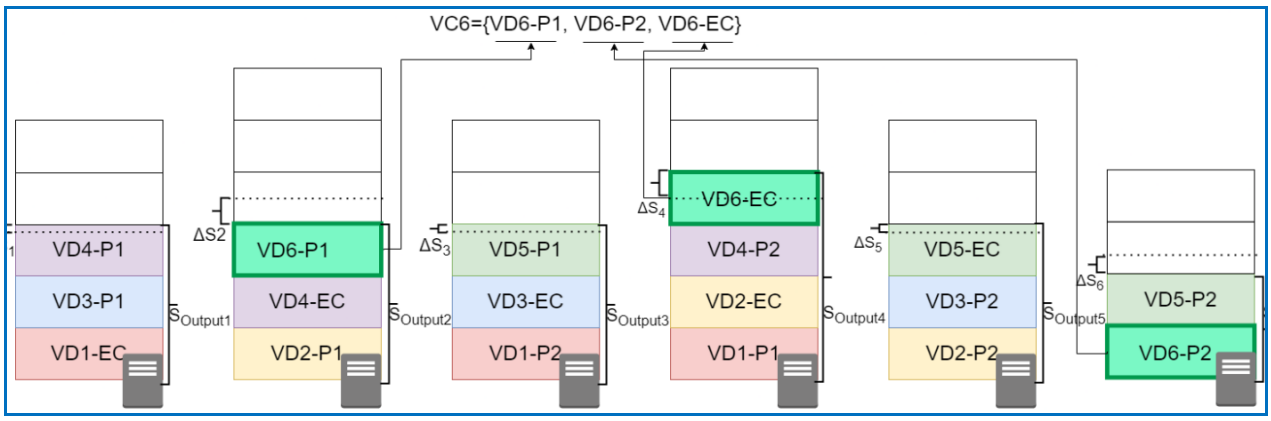

Рисунок 2.4 – Розподіл VCs та VDs по вузлах зберігання

3. Інтерфейс користувача: дозволяє користувачеві взаємодіяти з системою. Користувальницький інтерфейс також відповідає за операції поділу і об'єднання файлів замість вузлів зберігання, щоб знизити навантаження на систему.

Необхідні визначення для алгоритмів управління даними в таблиці 2.3.

Таблиця 2.3 – Необхідні визначення для моделі розподілу файлів

Змінна	Значення змінної
1	2
VDs	Віртуальний кластер у системі
VDs	Віртуальний диск у системі
VC	Множина віртуальних дисків у системі
VD	Множина порожніх слотів віртуальних дисків у системі
VD	Множина віртуальних кластерів у системі
VD `	Підмножина VDs розташованих у вузлі i
$state(d)$	Підмножина VDs розташованих у вузлі i, які мають стан x
VD (i)	Підмножина VDs розташованих у вузлі i.
VDs (x)	Функція , що повертає стан віртуального диска d.
$vd\ vc, p$	Елемент множини VDs, розташований у вузлі i і має порядок j

Продовження таблиці 2.3

1	2
VC $_{n\,(i)}$	Підмножина VDs, які мають стан x
VC $_{i}$	Елемент множини VDs, що представляє vd , що належить віртуальному кластеру vc і має частину p. p може бути 1 для першої частини, 2 для другої частини, 3 для прання кодування
$fsr(i)$	Підмножина VC , яким належать вузли i
vcd	Елемент множини VC з індексом i
$p(d)$	(Free space to request ratio) функція, що повертає відношення вільного місця до запиту для вузла i
$mcvd(x)$	Елемент множини VC , який містить конкретний віртуальний диск d
$MTTF\ system$	Функція, яка повертає номер частини віртуального диска d
$MTTF\ node$	(Mean count of VDs) – функція, яка повертає середню кількість VD у стані x на вузол
$MPRC$	Час напрацювання на відмову всієї системи
$PSFANF$	Час напрацювання на відмову вузла
$MTTR$	Максимально можлива кількість процесів відновлення

2.2.1 Алгоритми балансування даних

При роботі зі смарт-контрактами існує кілька обмежень, які враховуються при реалізації алгоритмів балансування даних, оскільки ці алгоритми працюють на смарт-контрактах. Наведемо ці обмеження.

1. Обмеження мови програмування: не все мови програмування підтримуються в мережі блокчейн : Багато мереж блокчейнів використовують мова програмування, спеціально розроблені для смарт - контрактів, наприклад Solidity.

2. Складні типи даних і структури не підтримуються, що обмежує можливості роботи із змінними. Наприклад, мережа блокчейн Ethereum (Solidity) обмежує складність алгоритмів за рахунок витрачання комісій (Ethereum gas).

3. Обмежені бібліотеки і функціональність: смарт-контракти мають обмежені бібліотеки і функціональність по порівнянні з програмуванням спільного призначення. Наприклад, Solidity (один з самих популярних мов для написання смарт-контрактів) не підтримує модифікатори, успадкування, навантаження функцій та операторів, нескінченні цикли, рекурсії та числа з плаваючою точкою.

Не всі алгоритми підходять для роботи в рамках смарт- контрактів, особливо алгоритми, які включають складні переходи між станами або складні структури даних. Запропоновані алгоритми враховують кілька обмежень при балансування даних, в том числі підтримка рівного розподілу даних на основі вільного простору вузлів та розміру самих даних. Також у смарт - контрактах реалізується вимога про те, щоб система не містила на одному вузлі двох VD, що належать одному і тому ж VC, щоб зробити можливість відновлення даних в випадку відмови вузлів і перевірку стану використання VD (гарячі – VD, до яких звертаються частіше всього, холодні - VD використовуються для стираючого кодування і звичайні VD – це всі інші VD). На розподіл даних впливають кілька подій, які впливають на баланс даних в системі (рисунок 2.5).

Є чотири основних події, які безпосередньо впливають на цей розподіл:

- додавання вузла;
- додавання файлу;
- відмова вузла;
- поновлення вузла.

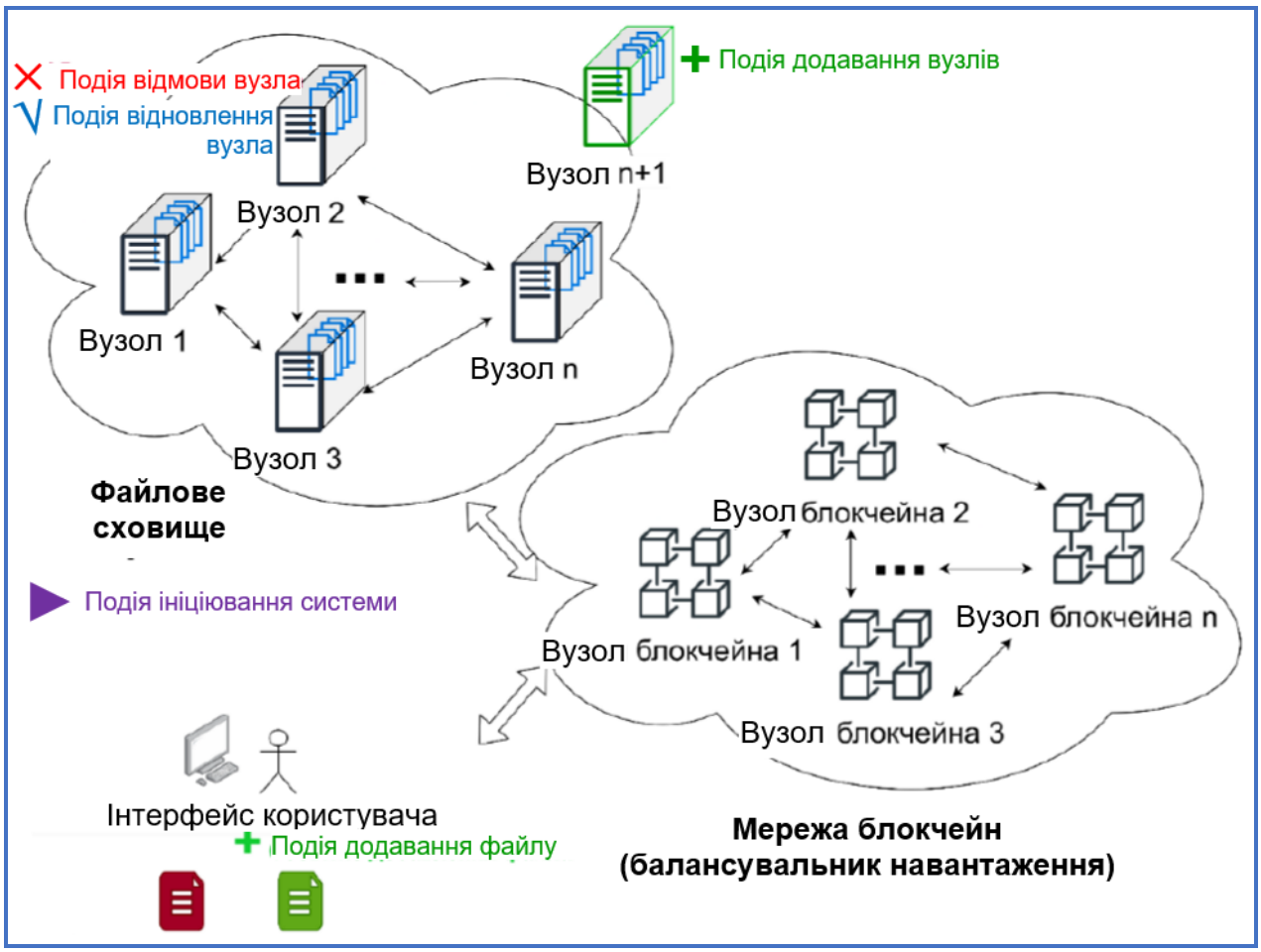

Рисунок 2.5 – Події, що впливають на розподіл даних у системі

Існують і інші події, які можуть впливати на розподіл даних по вузлах, але відповідні події можна сформувати на основі чотирьох згаданих вище подій. До таких подій відносяться запуск системи, видалення файлів тощо.

2.2.2 Події додавання нових вузлів та файлів

При додаванні нового вузла розраховуються загальний вільний простір і максимальний простір для зберігання віртуальних пристроїв. Також необхідно враховувати кількість гарячих VD (віртуальних пристроїв, які використовуються частіше всього). Гарячі VD повинні бути розподілені порівну. Для визначення списку «гарячих» VD можна використовувати алгоритм Least Recently Used (LRU), де кількість слотів для алгоритму LRU визначається як загальне кількість кластерів, розділене на кількість вузлів.

Вузли сортуються по кількості «гарячих» вузлів в порядку спадання. Для кожного вузла списку, починаючи з першого елемента, вибирається набір гарячих пристроїв, виходячи з існуючих умов.

Існуючі умови визначають спосіб вибору віртуального диска, який враховує кількість вибраних віртуальних дисків з кожного вузла, кількість віртуальних дисків певного стану в кожному вузлі, вимога про те, що вузол не може мати два віртуальні диски, що належать одному вузлу. тому ж кластері, щоб мати можливість відновити дані у разі втрати даних тощо.

Процес повторюється для холодних VD (дисків, що містять стирального кодування, оскільки вони використовуються тільки для цілей відновлення даних), тому новий вузол отримує вказане кількість гарячих і холодних пристроїв. Залишилося необхідне кількість необхідних VD досягається вибором VD, які не є ні гарячими, ні холодними.

Балансувальник навантаження при додаванні файлів обирає VC або створює новий на основі розміру файлу та вільного місця у віртуальних кластерах у системі.

Застосунок на боці користувача поділяє файл і обчислює відповідне кодування з корекцією стирань.

2.2.3 Події відновлення втрачених даних та поновлення вузлів

Подія відновлення втрачених даних використовується для відновлення вузлів в випадку збою вузла. Певні контрольні повідомлення регулярно вирушають зі всіх вузлів в балансувальник навантаження. Ці повідомлення показують, що вузол все ще працює. Коли вузол не відправляє повідомлення в перебіг певного кількості послідовних інтервалів, він потрапляє в список «автономних пристроїв». Коли вузол недоступний, його віртуальні диски необхідно перебудувати в інші існуючі вузли. Коли вузол відключається від мережі, для відновлення даних потрібно відновлення втраченого віртуального диска. Цей процес має деякі вимоги:

- VD не може співіснувати в одному вузлі з іншим VD, що належать тому ж самому VC;

- дані повинні розподілятися справедливо, так як процес відновлення даних повинен враховувати все доступні вузли.

Математично це завдання можна розглядати як класичне завдання теорії ігор, оскільки виграш може бути досягнуто з допомогою стратегії (способу розподілу даних). Отже, при великій кількості VD і вузлів, завдання відновлення неможливо вирішити за поліноміальний час. Роздача VD – це стохастична гра, так як це повторювана гра, в якій майбутні дії залежать від поточної дії. У цьому випадку вчинення дії призводить не до абсолютно нової гри, а до гри, в якій множина можливих дій менша поточної на одну. Гравцем в цій грі є система, і вона має кінцеву множину вузлів, і кінцеву множину VD. Мета цієї ігри – справедливо розподілити віртуальні диски по вузлах із врахуванням двох згаданих вище умов. Цю гру можна визначити кортежем $\langle Q, N, A, P, R\rangle$, де введені такі позначення:

- Q – кінцева множина станів гри, де в кожному стані є розподіл VD;

- N – кінцева кількість гравців, що дорівнює одному (системі);

- $A= A_1\times \cdot \cdot \cdot \times A_n$, де A_i – це кінцевий набір доступних дій (наприклад, перебудувати VD5, VD7 та VD15 у вузлі 1, а VD1, VD6 у вузлі 2 тощо);

- $P : Q \times A \times Q \rightarrow [0,1]$ є функцією ймовірності переходу; $P(q, a, \hat{q})$ – ймовірність переходу зі стану q у стан $\hat{q}$ після виконання дії a;

- $R = r_1 ,... , r_n$, де $r_i : Q \times A \rightarrow R$ є результуючим виграшемВ

играш у цій задачі може визначатися декількома змінними. Вибір вузла для VD, де цей вузол має інший VD у тому кластері, дає негативний виграш. Надсилання VD на вузол, який зберігає «справедливий» розподіл призводить до позитивного результату і так далі. Хоча існує стохастична гра з одним агентом, ця проблема представляє собою марківський процес прийняття рішень. Після визначення всіх можливих станів вибирається шлях, гарантуючий найбільший виграш. При великому кількості VD і вузлів

завдання неможливо вирішити за поліноміальне час. Ця проблема вважається NP-повний проблемою. Реалізація такого алгоритму в середовищі смарт-контрактів недоцільна. Замість цього пропонується визначити функцію fsr (Free space to request ratio). При відмові вузла список можливих вузлів для кожного віртуального диска, існуючого на втраченому вузлі, визначається на основі кількох факторів, таких як кількість доступних порожніх слотів віртуального диска на кожному вузлі і відсутність конфлікту з відповідним віртуальним диском. на основі цього кожен вузол складає список віртуальних дисків, які він може прийняти. fsr - це ставлення кількості вільних слотів віртуальних дисків до сумі цих запитів. Складність пропонованого способу складає O(n). Цей підхід забезпечує можливість відновлення кількох послідовних збоїв вузлів, поки в вузлах не залишиться вільного місця чи не залишиться менше трьох вузлів.

Подія поновлення вузлів використовується у випадку, якщо вузол, вийшов з ладу раніше і відновлює роботу. Тоді обчислюється хеш кожного з його віртуальних дисків і порівнюється з відновленими. Віртуальні диски, хеш яких не збігається, замінюються відновленими. Якщо хеш збігається, відповідна запис в смарт-контракт оновлюється. У процесі відновлення, якщо файли змінюються на віртуальному диск, створюється буфер, який містить тільки зміни, і ці зміни об'єднуються після завершення процесу.

Система, в якої реалізована модель розподілу об'єктів з вищеописаними подіями, далі буде називатися DecStore.

2.3 Моделі та алгоритми контролю доступу в децентралізованій системі зберігання даних

Розглянемо розподілену модель контролю доступу, яка мінімізує розмір даних в блокчейне . Ця модель складається з кількох алгоритмів і конкретної архітектури, а також новою техніки кешування.

Ця модель також адаптує особливий тип дерев, званий деревами Меркла.

Дерево Меркла – це тип дерев, який використовується для швидкого пошуку змін даних в дереві або перевірки приладдя вузла дереву без проходження всіх вузлів дерева.

Листя дерева Меркла - це хеші необхідних блоків даних або транзакцій, як в блокчейні. Це дерево є двійковим, тому кожен вузол має два дочірніх вузла.

Кожен проміжний вузол представляє собою хеш суми двох його дочірніх вузлів.

На рисунку 2.6 представлений приклад дерева Меркла. Наприклад, A, B, C, D можуть бути файлами, а листя дерева можуть представляти собою хеші відповідних файлів. Якщо файл A змінюється, в результаті змінюється його хеш, а це означає, що батьківський вузол (хеш(A+B)) теж змінюється. Це означає, що значення кореневого вузла Меркла змінюється, оскільки воно представляє собою суму хешей вузлів 1 і 2.

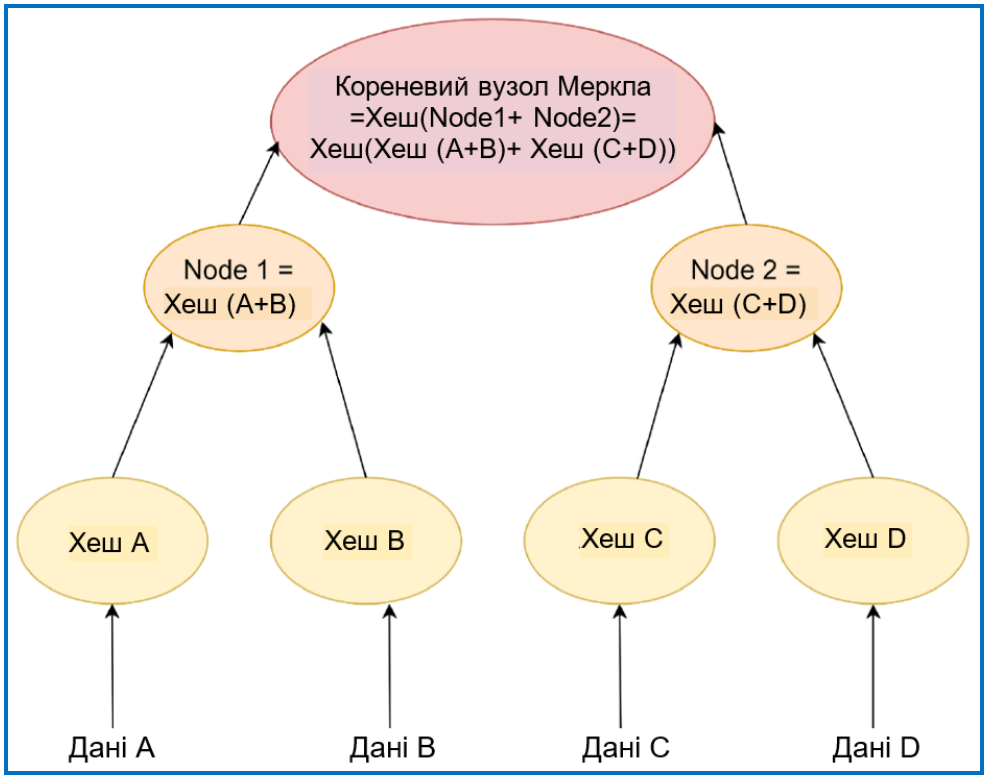

Рисунок 2.5 – Приклад дерева Меркла

Крім того, легко підтвердити, що вузол A існує у дереві. Якщо вказано значення вузла A, на додаток до хешу B і хешу вузла 2, можна підтвердити, що вузол A належить дереву, об'єднавши хеш вузла A з хеш B і обчисливши хеш цієї суми, який буде рівним хеш вузла 1, а потім об'єднати його з x 1 і вузла 2. Отримане значення має дорівнювати хеш-значенню кореня Меркла. Це означає, що для доказу існування вузла в дереві потрібно лише підмножина дерева. Для перевірки приналежності листа дереву необхідно відправити необхідне значення листа на додаток до хеш log2 кількість листя. Це означає, що це невелика кількість вузлів. У цьому прикладі існує 4 листа. Ось чому необхідно було відправити Hash A разом із log24 = 2 хеша. Якщо вузол складається з 1,000,000 записів, крім необхідного аркуша потрібно лише 20 хешів. Необхідні дані, що використовуються для доказу приналежності вузла дереву – доказ Меркла.

2.3.1. Модель контролю доступу

Щоб створити модель контролю доступу для DecStore, яка підтримує масштабованість і вимагає мінімального зберігання даних в блокчейне, архітектуру розглянутою вище моделі розподілу необхідно модернізувати за наступними правилами.

1. Кожен віртуальний диск (VD) забезпечений блоком, який називається «Компонент зберігання дозволів». Для кожного користувача, має доступ хоча б до одному файлу в зазначеному VD, в цьому блоці створюється дерево. Це дерево представляє файли, організовані в каталогах, до яким користувач має доступ через DAC. Одиниці зберігання, розташовані на VD, належать одному VC, мають однакову копію дерев.

2. Кожен вузол зберігання має одне дерево для кожного користувача, яке представляє собою комбінацію його дерев DAC на VD, існуючих в цьому вузлі зберігання.

3. На боці блокчейна у кожного користувача зберігається 1 хеш, який представляє собою загальний хеш кореня його дерева Меркла. Якщо файлів мільйон і користувачів 100, буде збережено тільки 100 хешів. Таблиця 2.4 показує, як інформація о доступі користувачів зберігається в блокчейне. Кореневий хеш дерева Меркла в цієй таблиці представляє собою хеш кореня дерева Меркла користувача DAC.

Таблиця 2.4 – Зберігання даних контролю доступу DAC в блокчейне

Користувач	Хеш кореня дерева Меркла	Ролі
f970e2767d0cfe7587 6ea857f92e319b	006d2143154327a64d 86a264aea225f3	ADMINISTRATOR
7694f4a66316e53c8c dd9d9954bd611d	76d80224611fc919a5d 54f0ff9fba446	DEVELOPER, TEAM-LEADER
…	…	

Також ролі зберігаються в блокчейне аналогічно, як в таблиці 2.5. Для кожноъ ролі будується дерево Меркла, в якому файли, доступні цієй ролі, розглядаються як листя дерева.

Таблиця 2.5 – Зберігання даних контролю доступу RBAC в блокчейне

Роль	Хеш кореня дерева Меркла	Вузли
ADMINISTRATOR	0cc175b9c0f1b6a831c399e269772661	1, 4
DEVELOPER	8a8bb7cd343aa2ad99b7d762030857a2	2, 3
…	…	

Типова гнучка система контролю доступу управляється політиками і складається з [2]таких компонент:

- точка адміністрування політики (PAP): це компонент, який створює політики доступу;
- точка інформації о політиці (PIP): це джерело, містить інформацію о політиці, на основі якої користувачеві надається або забороняється доступ;

- точка прийняття рішення о політиці (PDP): це компонент, який вирішує, надано чи користувачеві доступ чи ні;

- точка застосування політики (PEP): це компонент, який створює запит і відправляє його в PDP. PEP збирає необхідну інформацію з PIP і інших ресурсів, наприклад інформацію о запитах користувачів в PDP.

Точка вилучення політики (PRP): цей компонент не є обов'язковим. Він використовується тільки тоді, коли є кілька PDP і потрібно надати централізовану точку для відправки або отримання політик доступу.

У моделі, запропонованої в цьому дослідження, точка інформації о політиці (PIP) не зберігається на централізованому сервер. Коли PDP (який представлений в системі смарт-контрактом блокчейна) приймає рішення, він отримує інформацію від двох об'єктів: користувача і записів, що зберігаються в блокчейне. Інформація о повному доступі зберігається на вузлах зберігання, і користувачі синхронізують свої власні дерева доступу з деревами, розташованими на вузлах зберігання. Точка застосування політики (PEP), яка також представлена смарт-контрактом, відправляє в PDP доказ дерева Меркла з запиту користувача разом з відповідним очікуваним кореневим хешем дерева Меркла.

Таким чином, PIP представлений трьома сутностями:

- розподілена мережа вузлів зберігання, яка має повні політики і використовується тільки для того, щоб дозволити користувачам синхронізувати свої політики;

- сторона користувача, де кожен користувач зберігає свої політики, будь то DAC або RBAC;

- кореневі хеші дерева Меркла як для користувачів, так і для ролей, які зберігаються в блокчейне;

Дерева користувачів складаються із трьох типів вузлів:

- корінь – представляє собою загальний хеш дерева Меркла;

- листя – представляють файли (або каталоги, якщо політика дозволяє користувачеві доступ до всім файлів в каталозі замість вибору файлів

вручну). Вони можуть мати необов'язкове значення " Compress ". Це буде обговорюватися в розділі «Алгоритм стиснення дерев»;

- проміжні вузли – вони можуть бути каталоги або об'єднуючі вузли. Об'єднує вузол – це вузол, який об'єднує два вузла різних типів, причому будь-який з цих двох вузлів може бути листом, каталогом або об'єднуючим вузлом. Цей тип вузлів буде обговорюватися далі у підрозділі «Перетворення дерева у двійкове».

Кожен вузол дерева містить дві пов'язані структури даних: хеш і значення. Хеш представляє собою хеш значення. У таблиці 2.6 представлені значення кожного типу вузла. На рисунку 2.7 представлена оновлена архітектура системи з підтримкою контролю доступу. Видно, що дерева ролей розподілені по вузлам зберігання, де один вузол зберігання не зберігає роль і її копію разом., а її копію. Також всередині VD додаються дерева доступу.

Таблиця 2.6 – Типи вузлів у пропонованому дереві контролю доступу

Тип вузла	Структура вузла	Приклад
Файл	{ Name = File name ” , Type = File Access = ” Access type ” , Compressed = compressed paths ”, Path = " Path " }	{ Name =“document.pdf” , Type = File Access = R , Compressed = “” , Path = "/ documents /" }
Каталог	{ Name = “ Directory name ” , Type = Directory , Access = ” Access type ”, Compressed = compressed paths ”, Path = " Path " }	{ Name = " documents " , Type = Directory , Path = "/" }
Об'єднує вузол	{ Type = Comb, Paths = [“ paths array ”] }	{ Type = Combiner , Paths = [}

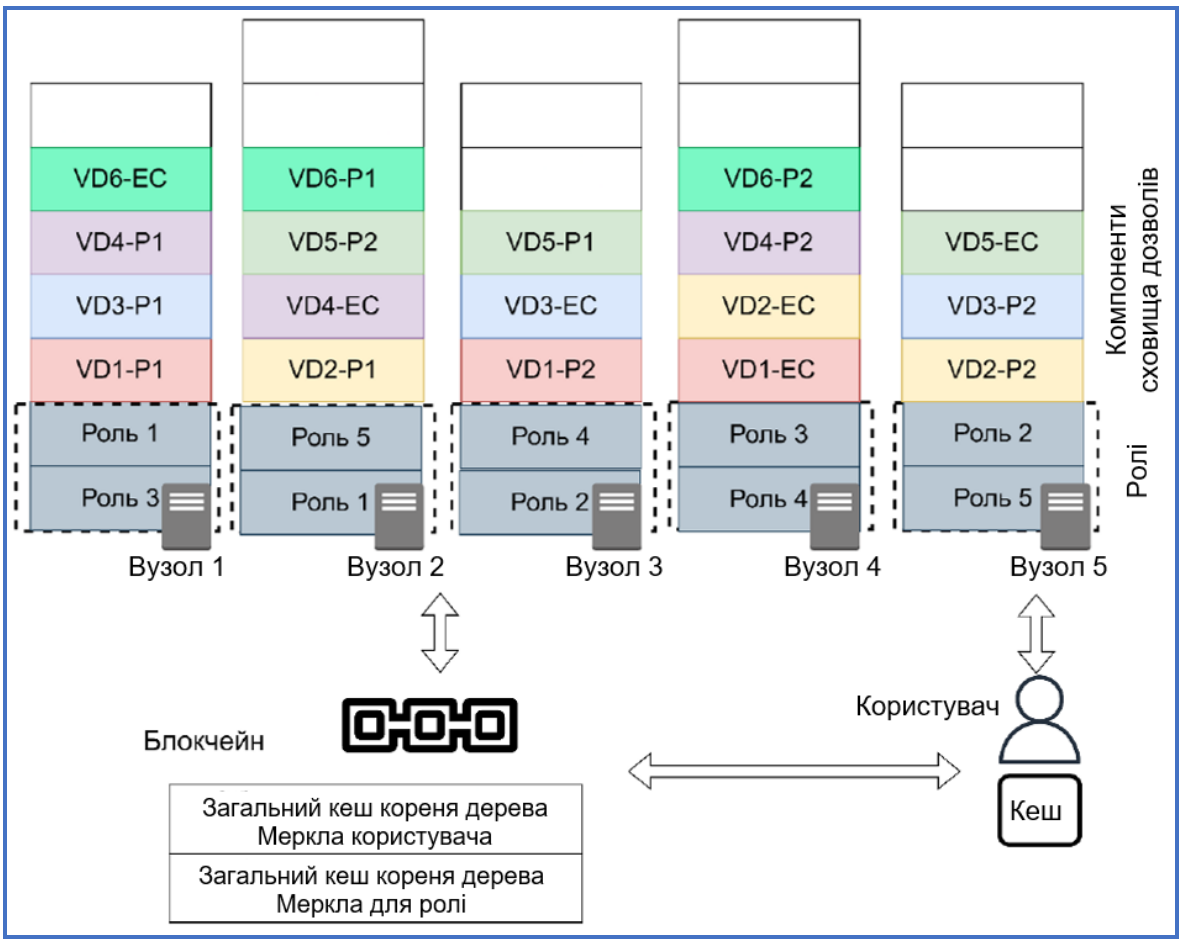

Рисунок 2.7 – Архітектура DecStore з моделлю контролю доступу

Опишемо основні етапи застосування описаної моделі.

Етап 1. У кожному вузлі зберігання створюється дерево для кожного користувача. У цьому вузлі є набір файлів, до яким має доступ користувач, які існують на віртуальних дисках з певним заздалегідь обраним індексом. Для кожного користувача все дерева з VD в вузлі об'єднуються в одне дерево, стискаються і перетворюються на бінарне дерево. Ці дерева використовуються для DAC.

Етап 2. Також кожен вузол зберігання має кілька дерев ролей. Існує додаткова копія кожного дерева ролей, розташована на другом вузлі. Усі вузли зберігання мають однакове кількість дерев. Блокчейн розподіляє дерева та їх копії по вузлам зберігання.

Етап 3. Користувачі синхронізують свої дерева з існуючими на вузлах зберігання. Сюди входять їх особисті дерева (DAC) та дерева ролей (RBAC).

Етап 4. Користувачі використовують свої копії дерев для доступу до файлів, генеруючи доказ Меркла.

Етап 5. Блокчейн обробляє процеси, пов'язані з поширенням і відновленням дерев в різних ситуаціях, щоб забезпечити балансування системи і можливість відновлення файлів або автоматичного відновлення на інший вузол при втрати вузла зберігання.

Далі розглянемо описані алгоритми докладніше.

2.3.2 Формування дерева Меркла

Спочатку випадок DAC буде докладно розглянуто, а RBAC – в кінці цього розділу. Алгоритм створення дерева Меркла складається із трьох основних етапів.

Етап 1. У кожному віртуальному диск, вибраному блокчейному, створюється дерево для кожного користувача. Це дерево показує, до яким файлів на цьому віртуальному диску має доступ Користувач.

Етап 2. Кожен вузол об'єднує ці дерева в одне дерево, яке показує, до яким файлів користувач має доступ в цьому вузлі. У процесі об'єднання файли з різних дерев, розташовані в однієї папці, об'єднуються в один вузол дерево.

Етап 3. Після цього дерево стискається шляхом об'єднання вузлів дерева, мають одного дочірнього елемента, в один вузол дерева, щоб мінімізувати розмір дерево.

Етап 4. Отримане дерево перетворюється в двійкове дерево, щоб мінімізувати розмір доказу Меркла.

Інформація о доступі користувачів розподіляється по вузлам зберігання з допомогою віртуальних дисків. Це означає, що дані реплікуються на віртуальні диски, належать одному і тому ж VC. При створенні об'єднаного дерева Меркла в цьому процесі бере участь тільки один з VD кожного VC, щоб уникнути надмірності, а значить, потрібно вибрати, який VD буде брати

участь в цьому процесі. Вибирати фіксований VD для всіх користувачів недоцільно, оскільки це означає велику навантаження на конкретні VD, тоді як краще розподіляти навантаження по VD. Призначати набір VD кожному користувачеві – не найкраща практика, оскільки це означає додаткове зберігання даних в блокчейне. Вибір VD в цьому алгоритмі динамічний.

Якщо отримане число равно 1, це означає, що для побудови дерева для даного користувача обраний тільки перший VD во всіх VCs. Якщо воно равно 2, це означає, що використовується другий VD.

Якщо воно дорівнює 0, це означає, що використовується VD, який містить кодування з корекцією стирань. Початковий вузол TreeNode в алгоритмі є кореневим вузлом дерево.

Дерева представляються за допомогою множин, де основною множиною є корінь дерева, а елементи цієї множини являють собою дочірні вузли кореня (файли та каталоги), а також є множинами, що представляють одне й те саме поняття. Ці множини (вузли дерева) мають атрибути (ім'я, тип тощо). Файли представлені порожніми множинами з атрибутами. Слова «вузли дерева» та «множини» будуть використовуватися як взаємозамінні. У кожному вузлі зберігання вибираються дерева на основі вибраних VD (на основі вибраного індексу VD), які об'єднуватимуться для побудови єдиного дерева на рівні вузла зберігання. Вузол дерева виражається множиною TreeNode, а корінь дерева виражається як TreeRootNode. Спочатку вибраним вузлом TreeNode в алгоритмі є TreeRootNode.

Атрибут x кореневого вузла дерева може бути виражений як $TreeRootNode_x$, а перший елемент набору – $TreeNode_1$. Це не обов'язково означає, що це дерево VC1. Це означає, що це дерево першого VD, що зберігається на вузлі зберігання, який знаходиться у списку вибраних VD. CombinedTreeRootNode можна розрахувати, вибравши об'єднання кожного TreeRootNode , що бере участь у цьому процесі.

Приклад з'єднання дерев представлений на рисунках 2.8 та 2.9.

Процес об'єднання повторюється для кожного вузла дерева в списку, і функція з'єднання виконується також для всіх отриманих вузлів дерева, поки не залишиться дочірніх вузлів.

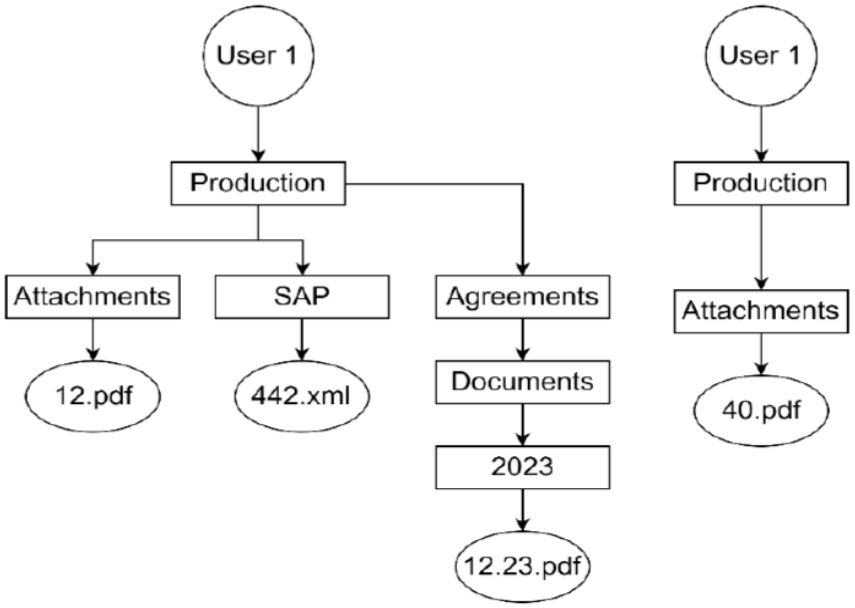

Рисунок 2.8 – Приклад дерев на різних VD (VD3-P 2 і VD6-P 2) в одному вузлі зберігання

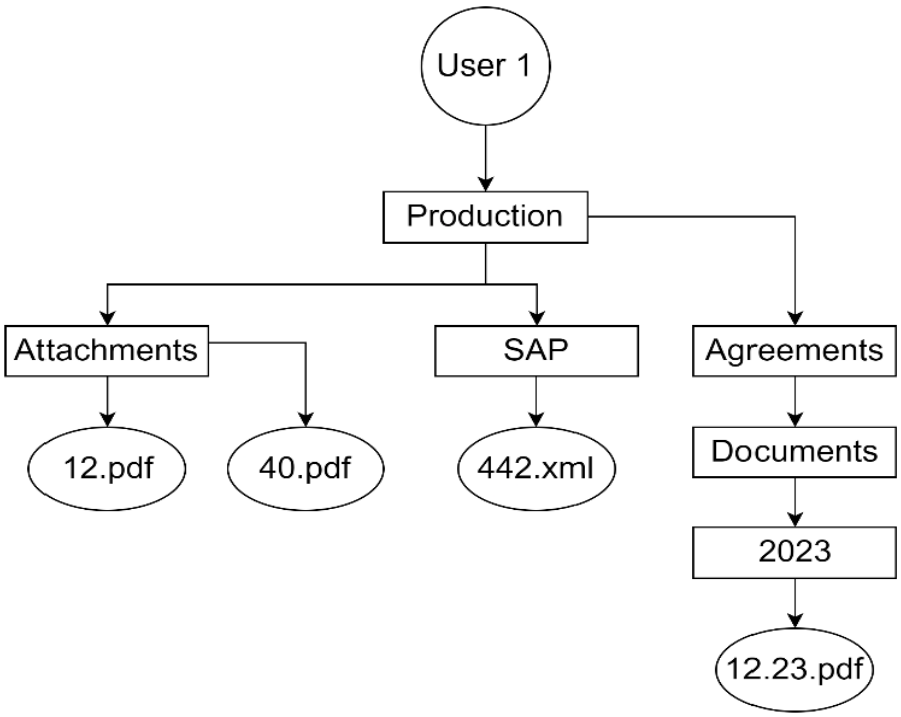

Рисунок 2.9 – Об'єднане дерево VD3-P 2 і VD6-P 2 користувача 1

Використовувані позначення показують, що вузли дерева, що мають однаковий атрибут імені, об'єднуються друг з другом і додаються в TreeNode, а також вузли дерева, що мають унікальні імена.

2.3.3 Процес стиснення дерев

Після об'єднання дерев з обраних для кожного користувача VD викликається алгоритм стиснення. Цей алгоритм необхідний для мінімізації розміру дерева шляхом стиснення вузлів дерева, мають одного прямого дочірнього елемента, в один, в результаті чого в цілому виходить менше дерево. Це означає, що каталоги, містять тільки один об'єкт (файл або каталог), стискаються в один.

Атрибут «Compressed» на листі дерева представляє шлях до стиснутим вузлам дерева, якщо вони існують. Атрибути «Access type» і «Compressed» в випадку каталогів використовуються тільки тоді, коли політика дозволяє користувачеві доступ до всім файлів в папці, тому в цьому випадку каталог є листом. Починаючи з кореневого вузла, алгоритм стиснення працює в моделі зверху вниз,

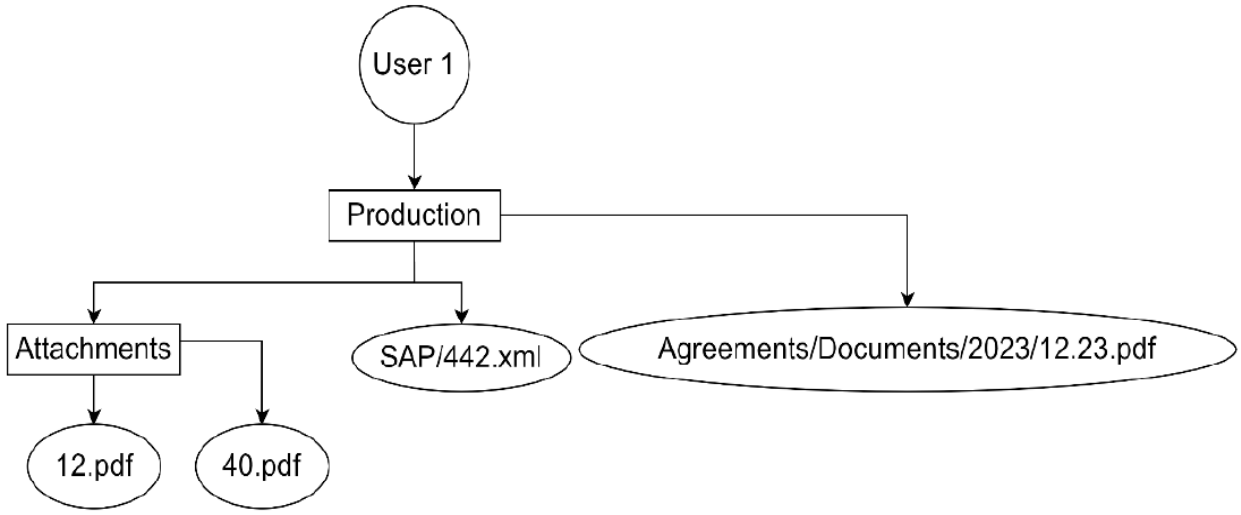

Рисунок 2.10 – Приклад стисненого дерева

Версія стисненого дерева, представлена на рисунку 2.9, показана на рисунку 2.10. Зрозуміло, що атрибут «Compressed» для 12.23.pdf дорівнює Agreements/Documents/2023.

2.3.4 Побудова двійкового дерева Меркла

Наступний крок – перетворення стисненого дерева в двійкове дерево. Для виконання цього кроку додається новий тип проміжних вузлів (об'єднувач). Об'єднувачі використовуються для об'єднання дочірніх вузлів таким чином, що тільки 2 вузла дерева вважаються прямими дочірніми вузлами. Цей тип вузлів має атрибут "Paths". Цей атрибут встановлюється, якщо об'єднуючий вузол має каталог в одному з своїх прямих нащадків, що не об'єднують вузли. На рисунку 2.11 показано, як створюються шляхи в двійкових дерев.

У об'єднувачі 3 значенням «Path » є doc і ppt. Це пов'язано з тим, що у нього є дві прямі папки: doc, який представляє собою «Compressed» значення для вузлів doc/1.doc і ppt. Папка 2023 не додається в шляхи, оскільки це дочірній вузол ppt, який вже доданий до шляхів.

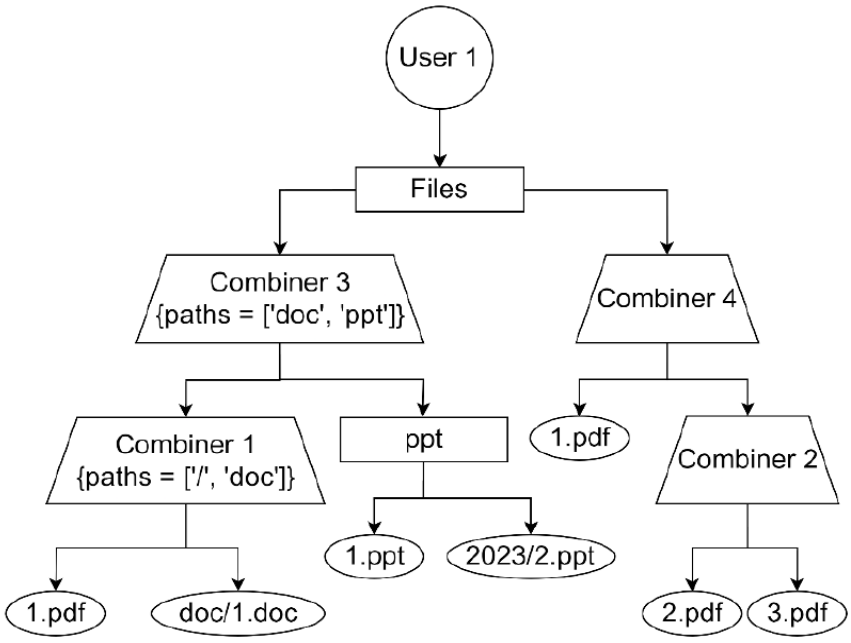

Рисунок 2.11 – Приклад використання атрибуту Paths

При доступі до 2.ppt, враховуючи, що повний шлях – / files / ppt /2023/2.ppt, об'єднувач 3 доступний безпосередньо, оскільки у шляхах є ppt. Після цього доступ до 2023/2.pdf здійснюється безпосередньо.

Перетворення дерева в двійкове дерево важливо для зменшення розміру докази дерева Меркла. можна розглядати приклад дерева на рисунку 2.12, в якому не використовується об'єднання вузлів.

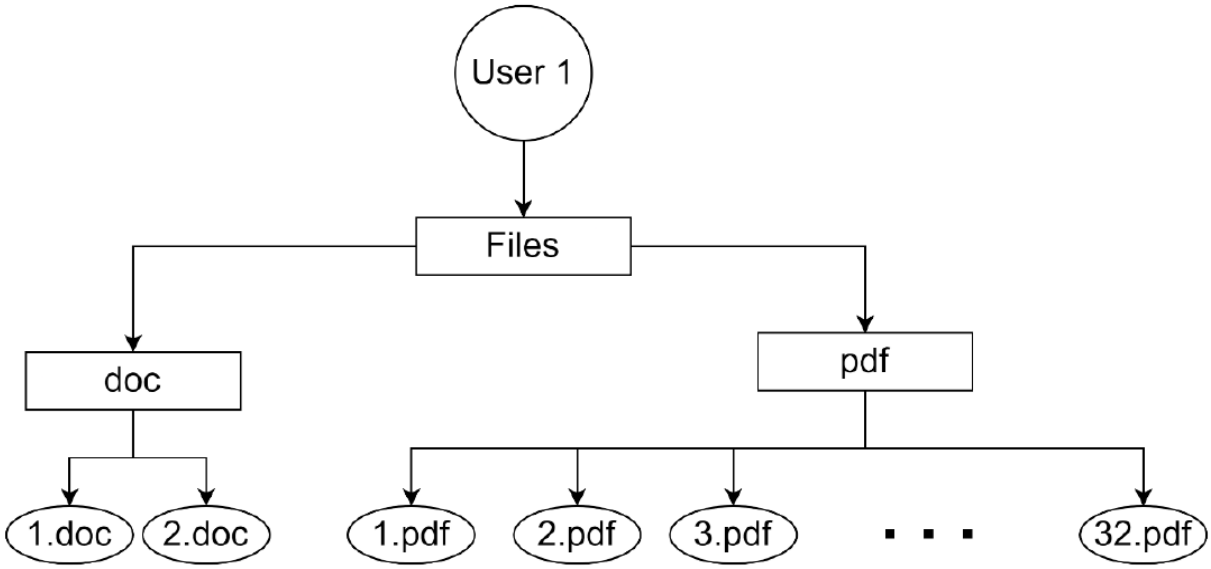

Рисунок 2.12 – Приклад дерева, не використовує об'єднання вузлів

Доказ Меркла для файлу 1.pdf вимагає перевірки 33 вузлів дерева: 1.pdf + хеші всіх файлів pdf + хеш каталогу doc. Якщо перетворити дерево в бінарне, то буде потрібно перевірка всього 7 вузлів: 1.pdf + хеш 2.pdf + хеш 4-х об'єднувачів (об'єднувачі 2 + 18+26+30) + хеш каталогу doc, як на рисунку 2.13.

При перетворенні стисненого дерева в двійкове зрозуміло, що кожні 2 дочірніх вузла будь-якого дерева об'єднуються в один об'єднувач, і процес є рекурсивним до тих пір, поки всього не залишиться лише 2 дочірніх вузла. Потім процес треба повторювати для кожного дочірнього вузла.

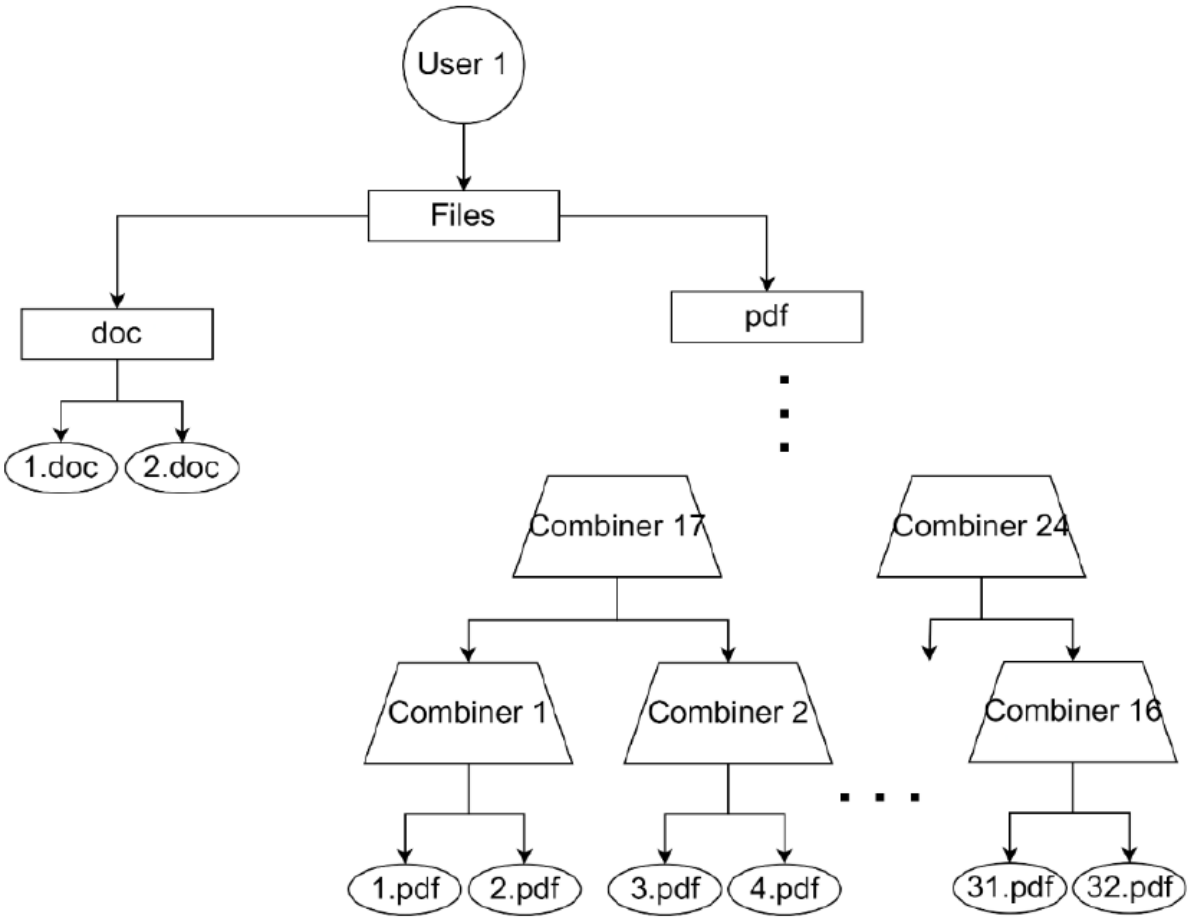

Рисунок 2.13 – Приклад дерева, в якому використовується об'єднання вузлів

Після того як все вузли зберігання збудують користувальницьке дерево, кореневі вузли об'єднуються таким ж чином, використовуючи об'єднання вузлів для побудови єдиного дерева Меркла. Хеш отриманого кореневого вузла цього дерева зберігається для кожного користувача в блокчейне, як у таблиці 2.4.

У випадку використання ролей (RBAC) для кожною ролі будується дерево. Дерево ролей містить файли, до яким політика, пов'язана з цією роллю, дозволяє доступ. У RBAC використовуються ті ж вузли дерева і атрибути, які запропоновані для DAC. Ролі дублюються і розподіляються по вузлам зберігання, де вузол зберігання не може зберігати роль та її дублювання.

2.3.5 Перевірка доступу користувачів

Перевірка доступу користувачів використовується для перевірки того, має чи користувач доступ до файлу або ні. Коли користувач відправляє запит на доступ до файлу, будь то запит на читання або запис, він відправляє доказ Меркла разом з запитом. Тіло запиту залежить від методу доступу, використовуваного для цього файлу. Таблиця 2.7 показує заголовок повідомлення в випадку RBAC та DAC.

Таблиця 2.7 – Заголовок запиту доступу до файлу

DAC	RBAC
{ Access = "DAC" , MerkleProof = […] , fileInfo = { … }	{ Access = "RBAC" , Role = Role type ” , MerkleProof = […] , fileInfo = { … }

У кожного користувача є копія його особистого дерева + копія дерева / дерев ролей. Перший крок – знайти файл. Цей процес можна виконати, використовуючи метод зверху вниз, без пошуку всіх можливих вузлів дерева, як в у разі пошуку в глибину (DFS) [105] , оскільки дерево Меркла побудовано у структурі, яка зберігає файли організованими в вихідний ієрархії шляхів. Доступ до файлів можна отримати швидко, обравши правильні проміжні вузли. Вузли об'єднання мають атрибут "Paths", який містить імена прямих каталогів, якщо вони існують. У цьому процесі також використовується атрибут Compressed , якщо є стислі вузли.

Як тільки користувач обирає файл з відповідного дерева, він відправляє інформацію о файлі разом з доказом Меркла в балансувальник навантаження, представлений смарт-контрактом блокчейна. Смарт– контракт блокчейна спочатку перевіряє тип доступу і хешує інформацію о файл, а потім обчислює хеш суми хеша інформації о файлі і першого хеша в масиві доказів Меркла і повторює процес для всіх хешей в масиві доказів Меркла. Після

цього він порівнює отриманий хеш з збереженим в таблиці 2.4 в випадку DAC або з збереженим хешем в таблиці 2.5 у випадку RBAC. Блокчейн може підтвердити особистість користувача на основі авторизації його гаманця.

Користувач відправляє значення файлу без його хеша та масив доказу Меркла (обов'язкові вузли дерева) $P = \{ P1, P2, P3, . . . P \}$, де $P1$ – необхідне значення інформації про файл, а $P2, P3, . . . P$ - Хеші необхідних вузлів дерева для доказу Меркла.

2.3.6 Кешування дерева доступу та зміна вузлів зберігання

У випадку DAC, якщо користувачеві надається або скасовується доступ до файлу f, або якщо змінюється тип доступу (читання/запис), цей файл додається/видаляється/оновлюється в дереві доступу на відповідному віртуальному диск. Зміни поширюються на об'єднане дерево і на загальний кореневий хеш Меркла. Користувач може виявити зміни в своєму дереві, порівнявши свій загальний кореневий хеш Меркла з хеш, що зберігається в блокчейне . Якщо хеш відрізняється, то користувач відправляє запит всім вузлам зберігання, містить його хеш кореня Меркла для вузла зберігання. Якщо хеш дерева Меркла відрізняється від хеша, що зберігається на вузлі зберігання, вузол зберігання відправляє назад оновлену версію дерева Меркла вузла зберігання, і користувач оновлює своє дерево на основі цього.

У випадку RBAC використовується той ж алгоритм. Якщо роль користувача змінюється, дерево оновлюється, і користувач виявляє це, порівнюючи хеш кореня дерева Меркла , що зберігається в блокчейне , з його збереженою версією ролей.

Якщо користувачеві додається нова роль, він запитує копію дерева Меркла ролей у одного з вузлів зберігання, на яких воно розміщено, і дерево перевіряє в блокчейне, є чи у користувача ця роль або ні, раніше чим відправити її назад.

Якщо користувач втрачає роль, ця роль видаляється з його записи в блокчейне, отже, таким чином, доступ анулюється.

При втраті вузла зберігання викликається алгоритм відновлення вузла зберігання, описаний вище. Цей алгоритм відновлює файли, використовуючи пральне кодування тільки для відновлення файлів. Як тільки віртуальний диск відновлює все блоки файлів, запит вирушає на один з вузлів зберігання, на якому зберігаються два інших віртуальних диска, належать тому ж кластеру, і виходять відповідні дерева Меркла для користувачів, які мають доступ DAC до файлів в цьому віртуальному диск. скопійовано на відновлений VD.

Після цього балансувальник навантаження ініціює відновлення дерев ролей, які зберігав втрачений вузол зберігання, вибираючи, на які вузли зберігання відправляти дерева, шляхом розрахунку кількості дерев ролей в кожному вузлі зберігання.

Оскільки кожна роль має дві копії, можна відновити втрачені дерева ролей на втраченому вузлі зберігання, скопіювавши їх безпосередньо з вузлів зберігання, на яких є їх копії. Цільовий вузол зберігання для кожного дерева ролей не може бути вихідним вузлом, так як дерево не повинно копіюватися з вузла зберігання на той ж вузол зберігання, щоб можна було відновити цю роль у випадку втрати вузла зберігання.

Якщо вузол зберігання, який був втрачено на короткий період часу (наприклад, кілька днів), відновлює роботу, втрачений вузол порівнює хеш кожного наявного у нього кореневого дерева Меркла з створеними копіями. Якщо хеш кореневого дерева Меркла збігається з копією, запису в блокчейне оновлюються, а зайва копія видаляється. Якщо воно не збігається, дерево копіюється із резервної копії, а зайва копія видаляється.

При додаванні нового вузла зберігання в систему викликається алгоритм додавання вузла зберігання, описаний вище, який переміщає деякі віртуальні диски в залежності від конкретних умов на новий вузол. Разом з ними переміщаються дерева відповідних користувачів. Дерева ролей

переміщаються тим ж механізмом, що і вибір блокчейна унікальні дерева ролей зі всіх вузлів зберігання і переміщає їх на знову доданий вузол.

Отже, наведений метод, який дозволяє розподіляти дані по вузлах із мінімізацією місця зберігання. Ця модель є основою архітектури інформаційної системи, яка визначає, як різні компоненти взаємодіють друг з другом. Пропонована модель розподілу даних призначена для роботи з блокчейн -системами.

Представлені підходи до балансування оперують цією моделлю і керують розподілом даних в зазначеній архітектурі в автономному режимі.

Алгоритми балансування моделі розподілу даних виконуються в середовищі смарт-контрактів, тому мають ряд обмежень.

Представлений метод розподілу даних зменшує загальний обсяг необхідного дискового простору на 25% у порівнянні з системами повного резервного копіювання за рахунок використання методу стираючого кодування, при цьому зберігаючи можливість автономного відновлення даних на інших вузлах.

Розглянута система контролю доступу, яка може працювати з блокчейном в поєднанні з локальним сховищем і механізмом кешування на боці користувачів. У цією системі частина даних контролю доступу зберігається у самих користувачів, які запитують авторизацію, однак модель використовує хеші кореня дерева Меркла, що зберігаються в блокчейне, для запобігання маніпулювання даними і мінімізація розміру даних що зберігаються в блокчейні. Дана модель контролю доступу складається з кількох алгоритмів, які дозволяють керувати контролем доступу в різних сценаріях і зменшує кількість необхідних запитів, а значить знижує навантаження на вузли зберігання. Аналогічну концепцію кешування можна використовувати для зберігання і управління метаданими.

3 РОЗРОБКА ТА ДОСЛІДЖЕННЯ МЕТОДУ БАЛАНСУВАННЯ ДЛЯ КЕРУВАННЯ ДАНИМИ НА БАЗІ СМАРТ-КОНТРАКТІВ

3.1 Розробка методу балансування для керування даними у децентралізованих застосунках

У DApps (децентралізованих застосунках) балансування навантаження та управління доступом є ключовими аспекти для забезпечення масштабованості, надійності та безпеки системи.

Наведемо поетапну послідовність дій, необхідних при забезпеченні даних процедур.

І етап. Балансування навантаження (Load Balancing) у децентралізованих застосунках.

Крок 1.1. Аналіз архітектури DApp, при цьому з’ясовується, які компоненти є on-chain (смарт-контракти) і off-chain (сервери, фронтенд), а також де можливі вузькі місця у децентралізованому застосунку.

Крок 1.2. Розділення навантаження, при цьому для on-chain це є децентралізація логіки між кількома контрактами (наприклад, factory-патерн), а для off-chain це використання балансувальників навантаження (наприклад, NGINX, Cloudflare) для API/web-серверів. На цьому кроці можна застосовувати CDN для швидшої доставки контенту.

Крок 1.3. Кешування та індексація, відбувається кешування частини даних off-chain (наприклад, через The Graph або Redis) та здійснюється використання окремого індексуючого сервера (наприклад, subgraph) замість частого звернення до блокчейну.

Крок 1.4. Географічне розміщення вузлів містить деплой (розгортання) серверів у різних регіонах (наприклад, через AWS, GCP) для зниження затримок.

Крок 1.5. Тестування під навантаженням з використанням спеціфічних інструментів (наприклад, Hardhat, Locust, Artillery) для стрес-тестів.

Крок 1.6. Асинхронна обробка транзакцій виконується для зменшення затримок через черги повідомлень (RabbitMQ, Kafka). Frontend подає транзакцію, Backend/Wallet обробляє її та результат надсилається назад.

II етап. Управління доступом (Access Control) у децентралізованих застосунках.

Крок 2.1. Визначення ролей і прав доступу за допомогою Owner, Admin, User, Oracle, Validator, тощо.

Крок 2.2. Впровадження RBAC (Role-Based Access Control), ролі зберігаються в mapping, для цього використовуються бібліотеки типу OpenZeppelin AccessControl.

Крок 2.3. Аудит функцій, що змінюють стан, з’ясовується, які функції можуть вплинути на баланси, конфігурації чи логіку. В результаті треба обмежити їх доступ тільки для певних ролей.

Крок 2.4. Логування дій (on-chain events).

Крок 2.5. Відкликання та зміна прав, використовуються механізми передачі прав власності (transferOwnership) або відкликання доступу.

Крок 2.6. Захист від атак, можна використовувати Reentrancy Guard (nonReentrant) або Rate limiting через frontend/backend (бо блокчейн сам по собі не обмежує швидкість викликів).

Крок 2.7. Використання проксі-контрактів для оновлень, підтримка оновлюваності (наприклад, через UUPS Proxy), причому доступ до функцій оновлення здійснюється лише для авторизованих ролей.

Крок 2.8. Документація і тестування політик доступу. Всі сценарії перевіряються через юніт-тести. Зовнішній аудит на контроль доступу – обов’язковий у production.

Отже, балансування навантаження забезпечує масштабованість та швидкість, а управління доступом – безпеку та контроль над критичними операціями. При втіленні процедур балансування навантаженням та

управління доступом до децентралізованих застосунків зазвичай виконують певну послідовність дій, наведену у таблиці 3.1.

Таблиця 3.1 – Порядок дій при функціонування балансування навантаження та управління доступом в децентралізованих застосунках

Категорія	Перевірка
Архітектура контрактів	Розділена логіка на окремі модулі
	Впроваджено проксі-контракт (UUPS/Transparent Proxy)
	Обмежено обчислення on-chain
Балансування навантаження	Кешування даних через The Graph/Redis
	Мінімізовано прямі звернення до блокчейну
	Використано CDN / Load Balancer (напр. Cloudflare)
	Стрес-тестування пройдено (Hardhat, Artillery, тощо)
Обробка транзакцій	Асинхронна обробка: frontend ↔ backend ↔ блокчейн
	Застосовані черги транзакцій/подій
Управління балансами	Баланси зберігаються в mapping(address => uint256)
	Валідація через require() перед оновленням балансу
	Застосовано шаблон Checks-Effects-Interactions
	Події (emit) використовуються для журналювання
Контроль доступу	Реалізовано onlyOwner, onlyAdmin та інші модифікатори
	Використано OpenZeppelin AccessControl (якщо потрібно)
	Додано RoleGranted, OwnershipTransferred події
	Є функції transferOwnership, renounceOwnership
Безпека	Впроваджено nonReentrant для чутливих функцій
	Кожна критична функція має рольовий захист
Тестування та документація	Покрито юніт-тестами сценарії зміни доступу
	Документовані ролі та права у README або технічній документації
	Проведено аудит або peer-review

Схема пропонуємого методу балансування для керування даними у децентралізованих застосунках, що враховують особливості смарт-контрактів, наведені на рисунку 3.1.

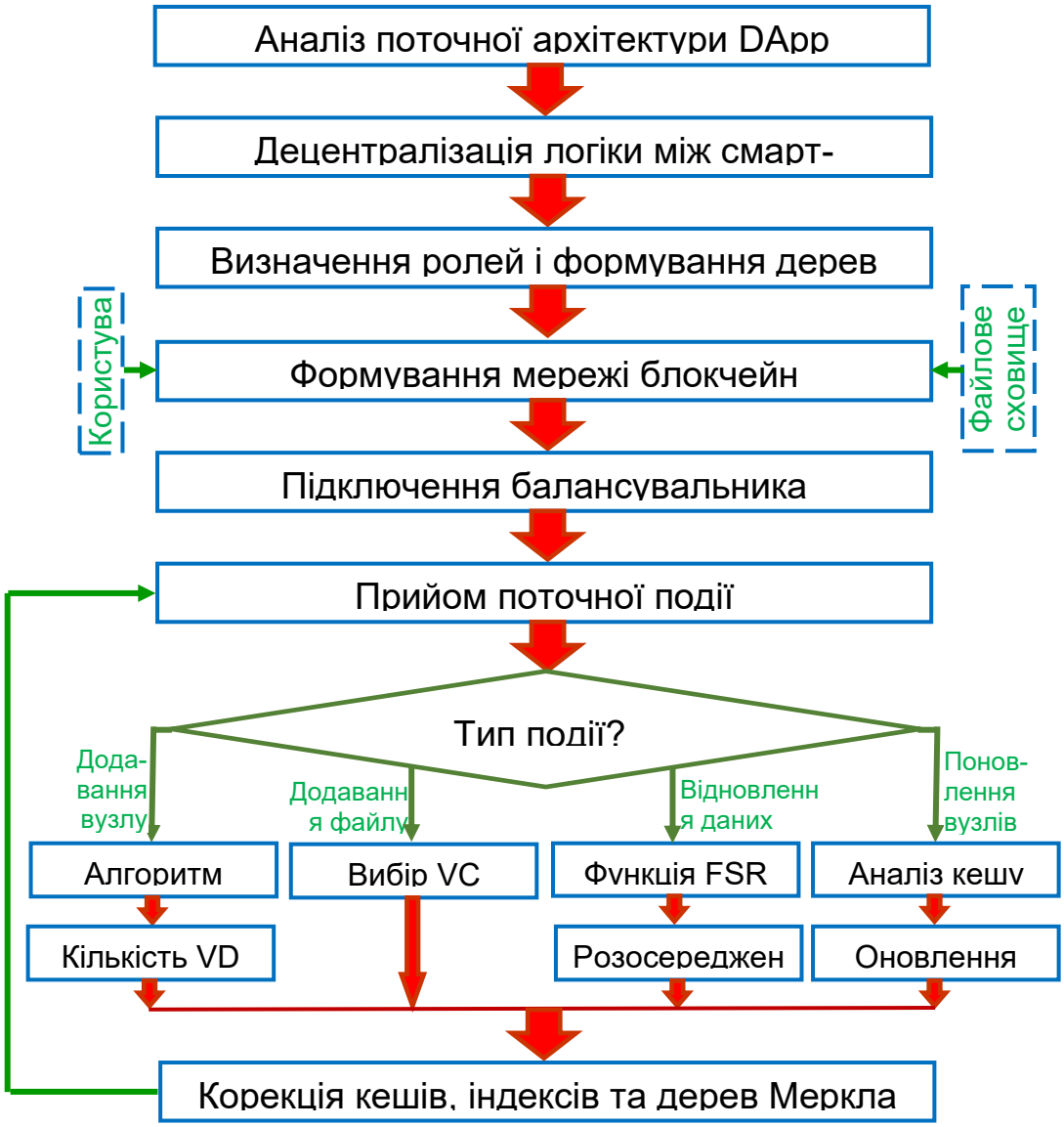

Рисунок 3.1 – Схема методу балансування для керування даними

3.2 Вибір інструментарію для оцінки ефективності методу

Існує кілька програмних платформ для реалізації приватного блокчейна. На початковому етапі для перевірки ефективності запропонованого методу потрібно провести низку експериментів, щоб вибрати платформу. За цієї причини в навчальній локальній мережі був

розгорнутий тестовий стенд, що включає сервер-балансувальник, кластер з трьох віртуальних файлових машин та мережа 100 Мбіт/с.

Мета експерименту – вибрати блокчейн платформу, яка працює відповідно запропонованим вимогам, і проаналізувати продуктивність системи. Для цього була розроблено базова система, яка обробляє завантаження і скачування файлів. При завантаженні файлу він розбивається на дві половини і розраховується кодування з корекцією стирань, а також обчислюється його хеш, а потім надходить запит в мережу блокчейну. Після цього кожна половина (і кодування з корекцією стирань) завантажується на сервер. Цей експеримент був повторений на трьох блокчейн – мережах: Hyperledger, Ethereum Enterprise (Hyperledger Besu) та Ethereum Ropsten (публічна мережа Ethereum).

Hyperledger і Ethereum Enterprise є більше підходящими блокчейн – рішеннями для реалізації запропонованого підходу по таким причинам:

- дані платформи призначені для створення корпоративних рішень по моделі b2c;
- немає необхідності використовувати криптовалюту;
- наявність смарт - контрактів і мови їх програмування – Solidity (Ethereum Enterprise) або GoLang (Hyperledger);
- підтримка конфіденційних транзакцій (можливість налаштування доступу до певних транзакцій для певних користувачів).

В експерименті використовувався віртуальний сервер (2 ГБ ОЗУ, 2 віртуальних ЦП Intel Xeon Gold 6230, жорсткий диск 512 ГБ) з пропускною здатністю мережі 30 Мбіт/с. Для Ethereum Enterprise використовувався клієнт Hyperledger Besu на віртуальному сервері (6 ГБ ОЗУ, 2 віртуальних ЦП Intel Xeon Gold 6230, 20 ГБ), пропускна здатність мережі 30 Мбіт/с. У цій інфраструктурі вимірювалася загальна швидкість наступних тестів:

- завантаження з балансувальника 1000 нових документів, включаючи XML з метаданими і файл розміром 10 МБ з довільним вмістом;

- завантаження 1000 документів (XML + файли) через балансувальник.

Швидкість тестувалася на балансувальнику навантаження, щоб виключити з вимірювань затримки клієнтського трафіку.

Як видно з результатів (рисунок 3.2 і таблиця 3.2), обидві реалізації цілком можна порівняти по часу з відомими p2p-рішеннями, працюючими по торрент-протоколам. Це підтверджено експлуатаційними тестами.

Крім того, було проведено тестування з використанням смарт-контракту, розміщеного у публічній тестовий мережі Ethereum Ropsten .

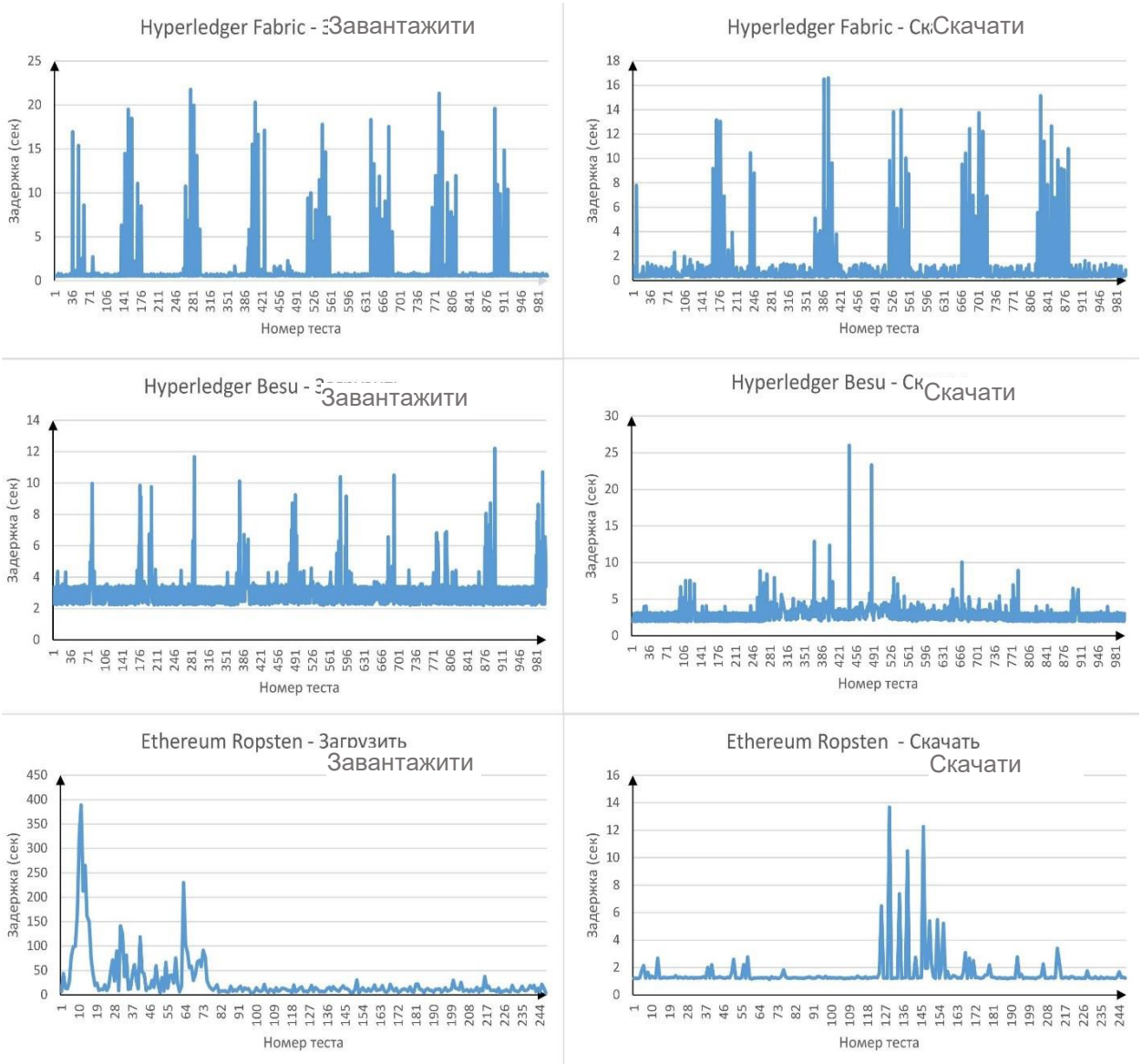

Рисунок 3.2 – Затримка завантаження/скачування файлу в Hyperleger, Ethereum Enterprise (Hyperledger Besu) і Ethereum

Таблиця 3.2 – Hyperledger / Hyperledger Besu / Ethereum затримка передачі інформації про файли (сек)

		Min	Max	Avg	Median
Hyperledger	Завантажити	0.535	21.762	1.242023	0.585
	Скачати	0.27	16.615	0.927483	0.432
Hyperledger / Besu	Завантажити	2.21	12.214	3.203192	3.294
	Скачати	1.937	26.012	2.986525	2.902
Ropsten	Завантажити	3.512	388.917	27.27899	11.935
	Скачати	1.142	13.673	1.597874	1.258

У випадку з Ropsten, хоча завантаження файлів відбувається досить швидко, видно, що завантаження файлів відбувається в кілька раз повільніше. Це пов'язано з завантаженістю тестової мережі і використанням в ній алгоритму консенсусу PoW на відмінність від Hyperleger Fabric і Hyperledger Besu, які за замовчуванням використовують менше ресурсомісткі алгоритми (PBFT або PoA). Основна проблема загальнодоступних рішень DLT полягає в тому, що середній час обробки транзакцій занадто великий. Отримані результати показують, що публічні мережі блокчейнів не підходять для створення такої системи.

Наведений експеримент показав, що таку систему можна швидко масштабувати по горизонталі з використанням недорогих серверів. При цьому для зберігання файлів 1000 електронних документів достатньо спільного об’єму не більше 15 ГБ (файл 10МБ + метадані xml в DLT). Таким чином, система може зберігати до 1'000'000 електронних документів з використанням відносно дешевих вузлів загальної ємністю 15 ТБ. Для цього можна організувати файлове сховище на SAS -дисках з IOPS 10 \000 обертів/хв.

Для розробки системи зберігання була обрано Hyperledger Fabric, щоб провести наступні тести, так як ця мережа підходить більше всього, виходячи з проведених тестів швидкості, а також характеристик, розглянутих вище.

Hyperledger Fabric – це блокчейн – мережа з обмеженим доступом (транзакції не обов'язково повинні бути публічно опубліковані), а її архітектура підтримує наявність кількох організацій. Ще одним перевагою використання Hyperledger Fabric є те, що його механізм консенсусу набагато швидше і менше ресурсомісткий, чим інші загальні алгоритми консенсусу, такі як PoW, використовувані в інших популярних мережах блокчейнів, таких як Ethereum. Консенсус в Hyperledger Fabric припускає прийняття транзакції на основі кількості підписів, отриманих серед всіх можливих підписів, і додавання прийнятих транзакцій в спеціальний блок, після чого цей блок перевіряється.

Усі алгоритми були реалізовані в вигляді чейнкод в Hyperlager Fabric з використанням мови Go. Крім запуску ланцюгового коду, вузли блокчейна також запускають власний API, що дозволяє взаємодіяти з користувачами і вузлами зберігання. Віртуальні диски були реалізовані як томи Docker, де віртуальні диски, призначені для конкретної компанії, доступні за допомогою ізольованого контейнеру. Вузли зберігання також використовують власний API, щоб мати можливість взаємодіяти з вузлами блокчейна і користувачами. Цей API був реалізований з використанням PHP. Користувальницький інтерфейс був реалізований з використанням JS.

Реалізація прототипу DApps системи для перевірки моделі включала в себе такі елементи:

- модель розподілу віртуальних дисків (що включає реалізацію розглянутих вище алгоритмів розподілу даних), який дозволяє скоротити використовуване простір в 0,25 рази, має більше високу надійність, чим інші системи (при певних умовах) і використовує блокчейн для додаткового довіри до всю систему в загалом, а також надання можливості приймати централізовані рішення в децентралізованою інфраструктури;

- чейнкод Hyperledger (який запускає запропоновані алгоритми з використанням мови Go) API для взаємодії з чейнкодом;

- веб-сервіс, працюючий на вузлах, для управління файлами на вузлі + API для взаємодії з іншими вузлами;
- прототип користувальницького інтерфейсу.

Для оцінки продуктивності системи були проведено тести на трьох віртуальних машинах, що репрезентують собою вузли зберігання, а також ще на одній віртуальній машині, що представляює собою балансувальник навантаження. Кожна віртуальна машина має 3 ГБ ОЗУ, 1 ядро ЦП, жорсткий диск ємністю 20 ГБ і працює під управлінням Lubuntu. Кожен тест повторювався 100 раз і зазначалося середнє значення часу, необхідного для виконання необхідного завдання. Ці тести включали завантаження файлу, відключення одного з вузлів для запуску функції відновлення і вимір часу, необхідного для відновлення вузла на решти вузлах, а також повторне включення цього вузла для запуску функції поновлення. Важливо відзначити, що час, необхідне для виявлення втрати вузла, не враховується, оскільки воно залежить від конфігурації системи. Час вимірювався, починаючи з виклику алгоритмів відновлення або поновлення роботи. Результати показані в таблиці 3.3.

Таблиця 3.3 – Середній час тестування DecStore

Назва тіста	Необхідний час (мс)	Цільові показники (мс)
Додавання вузла	974	5000
Додавання одного файлу	1512	3000
Відновлення даних з втрачених вузлів	65 041	600 000
Відновлення вузла	1062	60 000

Для проведення експерименту був сформульовано результат базового сценарію тестування інформаційної системи, що складається з 3 вузлів зберігання, яка буде масштабуватись. Це сценарій використання реальною системи ТОВ «РІІТ» в великий фінансової організації. Цільові показники для тестування були отримані з цієї системи.

Перший тест полягав в вимірі часу, необхідного для додавання вузла, тому всього було використано 4 вузла зберігання. Додавання вузла не зайняло багато часу, так як в системі був один VD, і ніяких операцій по ребалансування не знадобилося. Додавання файлу розміром 10 МБ займає трохи більше часу, так як крім оновлення записів в блокчейне потрібно розбити файл і завантажити його блоки. Відновлення вузла зайняло близько однієї хвилини (для 100 файлів, то є 1 ГБ), і це пов'язано з тим, що до блокчейну було зроблено менше запитів по порівнянні з додаванням файлів по окремо. Час поновлення роботи вузла склало всього 1062 мс , оскільки після відключення вузла файли не були змінено, тому оновлювалися тільки записи в блокчейне , а відновлений віртуальний диск був видалено.

Цільові показники отримані з експлуатованою замовником системи з централізованою архітектурою, яку розробило ТОВ "РІІТ". У системі працює одночасно до 1500 користувачів і вона складається з 10 мережевих вузлів. затримка операцій в тесті цими тестами може бути непомітна для кінцевих користувачів при умови наявності швидких дисків і достатньою пропускний здібності в інфраструктурі, на якої розгорнуть DecStore . на основі проведених тестів видно, що все основні операції виконуються щодо швидко по порівнянні з очікуваними. Важливо відзначити, що для тіста використовувалися віртуальні вузли, тому час, необхідне для передачі мережевого трафіку, не враховується. Необхідний відзначити, що мінімальні технічні характеристики вузлів достатньо низькі (з використанням virtual box 2 ГБ ОЗУ, 1 процесор, ОС LUbuntu), а це значить, що систему можна побудувати з мінімальними витратами на інфраструктури.

На основі експерименту вдалося сформулювати правило, визначаюче відсоток скорочення простору при впровадженні системи в порівнянні з системами повного резервного копіювання який складається з резервних копій.

Пропонована система використовує на 25% менше місця для зберігання по порівняно з одним сервером резервного копіювання, при цьому на 50%

менше місця при використання 2 резервних серверів. Розглянемо наступний приклад.

У системі повного резервного копіювання, що складається з 1 резервного сервера на кожен сервер, створюється одна копія кожного об'єкт. Таким чином, для зберігання файлу розміром 10 МБ буде потрібно 20 МБ. У пропонованої системі файл розбивається на дві половини (5 + 5) і розраховується пральний код (5 МБ), таким чином, всього потрібно 15 МБ по порівнянні з системою повного резервного копіювання з одним рівнем резервного копіювання (на 25% менше). У системі повного резервного копіювання з двома серверами резервного копіювання потрібно 30 МБ (що означає, що пропонованої системі потрібно на 50 % менше).

3.3 Оцінка ефективності запропонованого методу

Для оцінки часових характеристик DApps, що реалізують запропонований метод і процедури контролю доступу, був розроблений ряд тестів емуляції роботи системи контролю доступу (DAC+RBAC) в різних архітектурах. Тести були написані на C++ на фреймворку QT, а дерева контролю доступу були представлені двома різними способами: файлами SQLite і файлами JSON, оскільки технології реалізації впливають на результати продуктивність.

Тестове оточення було розгорнуто під управлінням Ubuntu 22.04 LTS на процесорі Core i7-8575u та 16 ГБ оперативної пам'яті.

У начальному експерименті була поставлено мета оцінити час створення об'єднаного дерева з точки зору часу і місця для зберігання, а також зробити вибір технологій для його зберігання і обробки. Експеримент включав створення з нуля двох користувальницьких дерев DAC, що складаються з 500 нових файлів кожне, для створення об'єднаного дерева. Для створення хешів вузлів використовувалась хеш – функція SHA – 256. Тест проводився автоматично 100 разів. З таким одноразовим навантаженням

стикається інформаційна система, де одночасно працює 1000 користувачів зі середнього ступеню інтенсивності роботи з файлами (системи зовнішнього і внутрішнього документообігу, CRM системи і тощо) Середній час цього процесу склав 9,48 секунди у випадку використання SQLite і 0,55 секунди в випадку файлів JSON. Загальний розмір дерева в випадку SQLite складає 249,9 КБ, а при використання JSON - 552 КБ. Реалізація включала об'єднання 2 дерев в одне, стиск і перетворення в двійковий формат, щоб мінімізувати розмір користувальницького запиту.

У реальних застосунках DAC, мабуть, набагато менше, чим 1000 файлів на вузол, оскільки доступ зазвичай надається на основі ролей, а також шляхом застосування політик до каталогів і їх підкаталогів замість вибору окремих файлів один за іншим, що означає, що розмір отриманих дерев повинен бути ще менше.

Щоб дати оцінку методу контролю та балансування доступу і порівняти з іншими можливими реалізаціями, для тестування були розглянуті архітектури трьох систем:

- централізована система: для її реалізації використовувалася PHP-інфраструктура Laravel , оскільки вона вважається популярною платформою для реалізації API. MySQL використовувався для зберігання політик доступу;

- Blockchain Hyperledger : дані зберігаються безпосередньо в блокчейне , а не на зовнішньому ресурсі для зберігання політик, Docker використовувався для вузлів Hyperledger;

- DecStore : реалізація моделі контролю доступу, запропонованої в цьому дослідженні.

Було створено 1000 файлових записів, а також 1000 користувачів. Всім користувачам був надано доступ до всім файлів (1000 * 1000). Було виконано три тестових випадкИ:

- надання нового дозволу файлу: яке висловлює швидкість додавання нового правила доступу для користувача x до файлу y (1x1);

- повний відгук доступу користувача: це означає відгук всіх типів доступу, наданих користувачеві x, до будь-якого файлу у системі (1x1000);

- перевірка доступу: це таймер, необхідний системі для перевірки наявності у користувача x доступу до файлу y (1x1);

- процеси, необхідні для кожного випадку, перераховані в таблиці 4.4, а в таблиці 3.5 представлені результати тестів.

Таблиця 3.4 – Необхідні процеси для тестування продуктивності моделей контролю доступу

Тест	DecStore	Блокчейн	Централі-зований сервер
Надання нового дозволу файлу	Додавання доступу до дереву користувачів на VD. Додавання доступу до об'єднаного дереву користувача. Запуск стиснення перетворен-ня дерева в двійкові алгоритми. Оновлення кореня дерева MerkleTree в блокчейне	Додавання запису в Ledger .	Додавання записи доступу у БД
Повна заборона доступу користувача	Видалення дерев користувача з VD (2 дерева). Вилучення об'єднаного дерева. Оновлення записи користувача в блокчейні.	Оновлення записи користувача в блокчейне .	Вилучення записи доступу з БД
Перевірка доступу	Перевірка докази Меркла	Порівняння збігів збережених значень у Ledger	Перевірка наявності записи у БД

Таблиця 3.5 – Результат тестування швидкості процесів контролю доступу

Тест (вимірюється в мс)	DecStore		Блокчейн	Централізова ний сервер (цільові показники)
	JSON	SQLite		
Надання нового дозволу файлу	157	190	100	60
Повна заборона доступу користувача	330	330	300	60
Перевірка доступу	130	140	100	10

За результатами тестів продуктивності можна зробити висновок, що швидкість при використанні запропонованого методу в різних операціях несуттєво менше по порівнянні з іншими системами, що не є критичним, так як на відмінність від решти архітектур DecStore підтримує масштабованість. Зроблено висновок, що файли JSON можуть бути гарним способом представлення дерев. Однак продуктивність може відрізнятися при реалізації з використанням іншого стеку технологій.

Пропонована модель контролю доступу використовує вузли зберігання для зберігання всіх даних управління доступом, в то час як користувачі мають копію своїх власних даних, пов'язаних з доступом, що можна розглядати як метод кешування. Без цією техніки кешування здійснюється 4 запити, тому що користувач відправляє запит на доступ до файлу, потім запит вирушає в блокчейн , після чого пересилається на вузол зберігання, який містить інформацію о доступ, після цього відповідь вирушає назад на вузол блокчейна , і тільки після цього клієнту.

При використання кешування необхідне кількість запитів скорочується до 2 замість 4. Припустимо, що кожен користувач має доступ до 1000 файлів і права доступу до 3 файлів змінюються в тиждень. Також припустимо, що користувач звертається до 10 файлів в день і до всім файлів у нього вже є доступ. Це означає, що щодня вирушає 20 запитів + 3 операції синхронізації, виконувані в тиждень, які включають тільки торкнувся вузол зберігання, на якому розміщено тільки торкнуте об'єднане дерево. Це скорочення означає набагато меншу навантаження на вузли зберігання, враховуючи, що система може утримувати тисячі користувачів.

Що стосується сторони блокчейна , відповідні запити (20,000) розподіляються між вузлами блокчейна , оскільки вони відіграють рівну роль у цьому процесі.

Користувач також може швидко знайти властивості файлу, оскільки проміжні вузли дерева представляють папки, а вузли об'єднання мають

список прямих папок, які вони містять, тому файл можна знайти безпосередньо, замість перевірки всіх вузлів один за іншим.

При додаванні нового доступу до об'єднаному дереву користувача запуск алгоритму стиснення і перетворення дерева в двійковий файл може виконуватися швидше, чим при створенні з нуля, так як дерево може взагалі не вимагати стиснення, а результат запуску алгоритму перетворення перетворення дерева в двійковий формат може привести до додавання кількох вузлів без зміни більшості вузлів, оскільки воно вже знаходиться в двійковій формі.

Однак пропонована система має деякі обмеження:

- користувачі повинні регулярно синхронізувати свої дерева, щоб отримати доступ до системі. Однак синхронізація не означає копіювання всіх дерев користувачів. Зміни можна виявити шляхом порівняння спільного хеша з тим, який зберігається в блокчейне. Якщо хеш відрізняється, вузол із зміненим деревом можна визначити по його хешу;
- часті зміни правил доступу означають частую синхронізацію змін і запуск пов'язаних алгоритмів. Однак доступ можна призначати по каталогах замість вибору окремих файлів, а в деяких випадках замість DAC можна використовувати RBAC, що дозволяє ефективно мінімізувати кількість необхідних процесів.

3.4 Врахування вимог щодо використання методу балансування для керування даними на базі смарт-контрактів

Запропонований метод був розроблений із врахуванням вимог щодо продуктивності, безпеки, масштабованості та доступності:

Продуктивність підтверджена за рахунок таких модифікацій:

- введено технологію кешування для зменшення кількості запитів до зовнішнього сховища;

- коли користувач запитує файл, файл може бути швидко знайдений, оскільки вузли файлів і каталогів зберігаються за схемою, яка відповідає їх абсолютним шляхам у системі зверху вниз;

- зміни в будь-кому вузлі дерева при використанні запропонованого методу можуть бути виявлено негайно, оскільки змінюється хеш, і в результаті змінюється хеш кореневого вузла;

- при наданні користувачеві доступу до нової ролі, або до файл,у або до папки один вузол зберігання оновлює своє дерево, а інших вузлів така зміна не торкається;

- при повному видалення користувача видаляються його дерева, що вважається швидким, оскільки дерева користувачів ізольовані і можуть бути представлені одним файлом на кожному віртуальному диску, що набагато швидше, чим пошук файлів у системі та оновлення списку дозволено користувачам в цих файлах по одному.

Питання безпеки у методі обгрунтовуються таким підходом.

Кореневий хеш дерева Меркла в блокчейне змінюється всякий раз, коли доступ до файлу надається або відгукується. Незважаючи на те, що у користувача є своя версія дерева, він не може маніпулювати нею, додаючи новий вузол дерева або змінюючи тип доступу до листа. Якщо користувач змінює тип доступу до аркуші, його хеш змінюється, і коли запит вирушає в блокчейн , блокчейн хешує це значення і поєднує його з іншими хешами, як описано в алгоритмі перевірки доступу користувачів. Це приведе до іншому спільному хешу, що приведе до неможливості доступу користувача до ресурсу, що можна розглядати як безпечну модель в мережі з нульовим довірою.

Запропонований метод має високу масштабованість, оскільки виконані такі вимоги:

- на боці блокчейна для кожного користувача і ролі кореневого дерева Меркла зберігається тільки хеш кореневого дерева Меркла, замість

зберігання матриці контролю доступу користувачів до файлів. Це означає, що потрібно набагато менше місця для зберігання;

- що стосується сховища вузлів, то DecStore гарантує, що система буде залишатися збалансованою при додаванні або втрати вузла зберігання, переміщаючи віртуальні диски в рамках запропонованих алгоритмів, а також дерев ролей. Дерева доступу користувачів переміщаються разом з їх віртуальними дисками, тому вони розподіляються по вузлам зберігання, незалежно від розміру системи;

- кеш підтримується на боці користувача, що означає менше навантаження на систему.

Доступність підтверджується таким:

- система використовує алгоритми, що дозволяють мати дві копії ролей і три копії приватних дерев користувачів. У випадку втрати вузла зберігання його дерева копіюються з резервних копій в інші вузли зберігання, щоб зберегти дані доступними у разі втрати іншого вузла;

- якщо по якій-небудь причині користувальницька копія дерева Меркла втрачена, то в наступний раз, коли він відправить запит на синхронізацію, все дерева зі всіх вузлів будуть вважатися несинхронізованими, і користувач отримає копії всіх з них, а підсумкове дерево після цього буде відновлено.

Запропонований метод був розроблений із врахуваннямм сумісності з запропонованої моделлю зберігання файлів.

Однак цю роботу можна адаптувати для інших типів систем, які використовують блокчейн і автономне сховище для зберігання даних будь-якого типу.

3.5 Врахування обмежень щодо використання методу балансування для керування даними на базі смарт-контрактів

Слід зазначити, що існують деякі обмеження і вимоги, які слід враховувати при реалізації методу балансування для керування даними на базі смарт-контрактів:

- кількість вузлів повинно бути більшою, ніж три 3 вузли, причому чим більше вузлів, тим вища надійність;

- відсоток використання сховища вузлів повинен орієнтуватися на таку вимогу: системні вузли не можуть бути використані повністю, щоб мати достатньо місця для відновлення даних при відмові вузлів;

- максимальний розмір файлу визначається ємністю віртуального диска, збільшеною у 2 рази, оскільки файли поділяються, тобто якщо розмір віртуального диска встановлений на 20 ГБ, максимальний розмір файлу – 40 ГБ.

- розглянутий в даної роботі клас інформаційних систем, для яких підходять запропоновані моделі і алгоритми, не підходить для завдань зберігання об'єктів, вимагають безпосередньою обробки всередині DApps (пряма зміна даних всередині системи, запис відео безпосередньо на диски системи тощо), тому що всі операції читання, записи і видалення об'єктів повинні здійснюватися тільки через балансувальник навантаження.

Підсумовуючи, зазначимо, що були проведені тести для вибору відповідної мережі блокчейна для поставленого завдання як балансувальника навантаження. Результат тестів показав, що використання найбільш відомих блокчейн-платформ цілком прийнятне для кінцевих користувачів.

Реалізація запропонованого підходу в публічних мережах, таких як Ethereum, затруднена через непередбачувану швидкість обробки транзакцій та надлишкових обчислень при досягненні консенсусу PoW алгоритмами. Згідно з отриманими результатами, Hyperledger Fabric можна розглядати як відповідну блокчейн-мережу для представлених моделей, оскільки вона

працює краще в порівнянні з іншими мережами, підтримує смарт-контракти та не вимагає криптовалюти.

Було проведено ряд експериментів для оцінки продуктивності запропонованої системи зберігання та алгоритмів балансування, включаючи додавання файлу, додавання вузла, відновлення втраченого вузла та повторне приєднання втраченого вузла до системи. Результати показують, що продуктивність запропонованої системи зберігання є дещо вищою порівняно з очікуваними цільовими показниками.

Було обговорено деякі аспекти запропонованої системи зберігання, що складається з процедур, які стосуються продуктивності, масштабованості, необхідної пам'яті та запобігання вузьким місцям.

Було проведено кілька тестів для оцінки продуктивності та необхідного розміру сховища відповідної запропонованої моделі контролю доступу. Показано, що ця модель вимагає невеликого обсягу пам'яті на вузлах зберігання та мінімальної пам'яті для зберігання даних у блокчейні, при цьому швидкість її роботи можна порівняти з централізованими моделями контролю доступу. Також обговорювалися деякі аспекти цієї моделі контролю доступу щодо продуктивності, доступності та масштабованості.

Систему, яка використовує запропоновані моделі розподілу та контролю доступу, має ряд додаткових переваг, які дозволяють збільшувати кількість вузлів та кількості об'єктів на них (масштабування). При додаванні нового вузла частина даних переміщається новий вузол, щоб підтримувати баланс системи. У деяких системах, таких як системи повного резервного копіювання або Gluster , неможливо додати жодного вузла. Крім того, система забезпечує вертикальну масштабованість.

Метод можна використовувати для вирішення багатьох прикладних завдань. Потенційні завдання застосування запропонованого методу у корпоративних системах:

- системи, що вимагають зберігання великих обсягів транзакцій, наприклад, файли SAP XML;

- бібліотеки зображень, аудіо та відео, призначені для продюсерських компаній, таких як Sony Pictures;
- заміна хмарних послуг, таких як Amazon AWS;
- метод також можна використовувати у сервісах для кінцевих користувачів;
- системи, що пропонують безліч наборів даних для застосунків нейронних мереж, такі як Kaggle;
- NFT-системи, що зберігають метадані файлів, такі як OpenSea;
- заміна систем зберігання файлів, таких як Google Drive.

ВИСНОВКИ

Сукупність отриманих у кваліфікаційній роботі результатів дозволило вирішити актуальне науково-технічне завдання, спрямоване на підвищення продуктивності децентралізованих систем на базі смарт-контрактів.

В результаті проведених досліджень отримані такі наукові та практичні результати:

1. Проведений аналіз сучасного стану вирішення питань підвищення ефективності розподілу даних і керування доступом до них в децентралізованих системах. Визначено, що блокчейн набув популярності за останні кілька років, оскільки це розподілена система зберігання, може організувати обмін даними між учасниками з нульовою довірою завдяки незмінності даних, що зберігаються на вузлах, своїм алгоритмам консенсусу. Однак на вузлах дублюються дані, що призводить до їх надмірності. У результаті огляду виділено кілька показників ефективності систем розподілу даних: надмірність даних, рівномірність їх розподілу, надійність та властивість збільшення кількості вузлів та даних на них (масштабованість). Аналіз архітектури відомих IT-проектів та підходів показав, що кожен з них не може вирішити одне з трьох завдань одночасно: надмірність даних, рівномірний розподіл даних по вузлах, високу надійність при мінімізації необхідного місця зберігання для зберігання даних та їх резервних копій.

2. Визначені підходи до розподілу даних в децентралізованих системах. Наведений метод, який дозволяє розподіляти дані по вузлах із мінімізацією місця зберігання. Він є основою архітектури інформаційної системи, яка визначає, як різні компоненти взаємодіють друг з другом. Пропонований метод розподілу даних призначена для роботи з блокчейн -системами. Представлені підходи до балансування керують розподілом даних в зазначеній архітектурі в автономному режимі. Алгоритми балансування моделі розподілу даних виконуються в середовищі смарт-контрактів, тому

мають ряд обмежень. Представлений метод розподілу даних зменшує загальний обсяг необхідного дискового простору на 25% у порівнянні з системами повного резервного копіювання за рахунок використання методу стираючого кодування, при цьому зберігаючи можливість автономного відновлення даних на інших вузлах.

3. Визначені підходи до контролю доступу в децентралізованих системах зберігання даних. Розглянута система контролю доступу, яка може працювати з блокчейном в поєднанні з локальним сховищем і механізмом кешування на боці користувачів. У цієї системи частина даних контролю доступу зберігається у самих користувачів, які запитують авторизацію, однак модель використовує хеші кореня дерева Меркла, що зберігаються в блокчейні, для запобігання маніпулювання даними і мінімізація розміру даних що зберігаються в блокчейні. Дана модель контролю доступу складається з кількох алгоритмів, які дозволяють керувати контролем доступу в різних сценаріях і зменшує кількість необхідних запитів, а значить знижує навантаження на вузли зберігання.

4. Удосконалений метод балансування для керування даними на базі смарт-контрактів за рахунок формування гібридної архітектури системи, де частина даних зберігається в блокчейн, а частина в локальних сховищах вузлів системи, що дозволило підвищити продуктивність децентралізованої інформаційної системи. Було проведено кілька тестів для оцінки продуктивності та необхідного розміру сховища запропонованого методу. Показано, що метод вимагає невеликого обсягу пам'яті на вузлах зберігання та мінімальної пам'яті для зберігання даних у блокчейні, при цьому швидкість його роботи можна порівняти з централізованими методами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Коваленко А.А., Куценко Т.Г., Шаповал А.С., Ситник О.В., Ні Я. С. Огляд методів аналізу безпечного програмного забезпечення. Системи управління, навігації та зв'язку. Полтава : Національний університет «Полтавська політехніка імені Юрія Кондратюка», 2025. Вип. 1(79). С. 156–162.

2. Zhang G., Zheng W., Li K. Rethinking RAID-5 Data Layout for Better Scalability // IEEE Transactions on Computers. 2014. – Т. 63. – №. 11. – С. 2816–2828. – DOI. 10.1109/TC.2013.143.

3. Lee B., Ye Y., Qiao Y. Ethereum Transaction Performance Evaluation Using Test-Nets // Euro-Par 2019: Parallel Processing Workshops. под ред. U. Schwardmann [и др.]., Cham. Springer International Publishing, 2020. – С. 179–190. – DOI. 10.1007/978-3-030-48340-1_14.

4. Anand T. Ethereum Architecture and Overview // Blockchain and Ethereum Smart Contract Solution Development: Dapp Programming with Solidity. Apress, 2022. – С. 209–244. – DOI. 10.1007/978-1-4842-8164-2_6.

5. Kasahara S., Shen Y., Jiang X., Wan J. Smart Contract-Based Access Control for the Internet of Things // IEEE Internet of Things Journal. 2019. – Т. 6. – №. 2. – С. 1594–1605. – DOI. 10.1109/JIOT.2018.2847705.

6. Yang C.-T., Chen C.-J., Chen T.-Y. Implementation of Ceph Storage with Big Data for Performance Comparison // Information Science and Applications 2017. под ред. K. Kim, N. Joukov, Singapore. Springer, 2017. – С. 625–633. – DOI. 10.1007/978-981-10-4154-9_72.

7. Yang J., Tan M.-S., Ding L. Discretionary Access Control Method to Protect Blockchain Privacy // Cyberspace Data and Intelligence, and Cyber-Living, Syndrome, and Health. Springer, 2019. – С. 161–174. – DOI. 10.1007/978-981-15-1922-2_11.

8. Xu J. Case Study: Ceph // Block Trace Analysis and Storage System

Optimization: A Practical Approach with MATLAB/Python Tools. под ред. J. Xu, Berkeley, CA. Apress, 2018. – С. 209–227. – DOI. 10.1007/978-1-4842-3928-5_9

9. Xie J., Li Z., Jin J., Zhang B., Hua Y. Research on Blockchain Storage Extension Based on DHT // Proceedings of the 4th International Conference on Big Data Technologies. New York, NY, USA. Association for Computing Machinery, 2022. – С. 79–85. – DOI. 10.1145/3490322.3490335

10. Wu K., Ma Y., Huang G., Liu X. A first look at blockchain-based decentralized applications // Software: Practice and Experience. 2019. – Т. 51. – DOI. 10.1002/spe.2751

11. Wesselius J., Rooij M. de Permissions and Access Control // Pro Exchange Administration: Understanding On-premises and Hybrid Exchange Deployments. под ред. J. Wesselius, M. de Rooij, Berkeley, CA. Apress, 2023. – С. 677–733. – DOI. 10.1007/978-1-4842-9591-5_10

12. Wang H., Zhang Q. Research on Blockchain-Based Smart Contract Technology // Smart Computing and Communication. под ред. M. Qiu, Z. Lu, C. Zhang, Cham. Springer Nature Switzerland, 2023. – С. 515–524. – DOI. 10.1007/978-3-031-28124-2_49.

13. Tuler De Oliveira M., Reis L.H.A., Verginadis Y., Mattos D.M.F., Olabarriaga S.D. SmartAccess: Attribute-Based Access Control System for Medical Records Based on Smart Contracts // IEEE Access. 2022. – Т. 10. – С. 117836–117854. – DOI. 10.1109/ACCESS.2022.3217201

14. Tarjan R. Depth-first search and linear graph algorithms // 12th Annual Symposium on Switching and Automata Theory (swat 1971). 1971. – С. 114–121. – DOI. 10.1109/SWAT.1971.10

15. Savill J. Azure Stack // Microsoft Azure Infrastructure Services for Architects: Designing Cloud Solutions. Wiley, 2020. – С. 281–296. – DOI. 10.1002/9781119596608.ch8

16. Quan S., Zhao Y., Helil N. Attribute-Based Access Control Policy Generation Approach from Access Logs Based on the CatBoost // Computing and Informatics. 2023. – Т. 42. – С. 615–650. – DOI. 10.31577/cai_2023_3_615