

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Аналіз алгоритмів блокчейну та методи їх
покращення»**

Виконала студентка групи ІПЗм-23-2
ГАЛЬЧИН Анастасія Вікторівна

Керівник роботи:
ЧИЖМОТРЯ Олена Геннадіївна

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«1» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ГАЛЬЧИН Анастасія Вікторівна**

Керівник роботи: **ЧИЖМОТРЕ Олена Геннадіївна**

Тема роботи: **«Аналіз алгоритмів блокчейну та методи їх покращення»**,
затверджена Наказом закладу вищої освіти від «1» жовтня 2024 р. № 520/с

Вихідні дані для роботи: **аналіз блокчейн-системи, що використовують шардинг як механізм масштабування; розгляд методів безпечного розподілу вузлів між комітетами та забезпечення відмовостійкості системи під час адаптивних атак.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О. Г.	1.10.2024 р.	1.10.2024 р.
2	Чижмотря О. Г.	1.10.2024 р.	1.10.2024 р.
3	Чижмотря О. Г.	1.10.2024 р.	1.10.2024 р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена дослідженню ефективності шардінгових систем у блокчейн-мережах, особливо з урахуванням впливу візантійських вузлів та репутаційних систем на консенсус. Пояснювальна записка складається з вступу, трьох розділів та списку використаної літератури. Пояснювальна записка до кваліфікаційної магістерської роботи містить 73 сторінок, 17 ілюстрацій, 2 таблиці, 2 додатки та 47 посилань на зовнішні джерела.

Метою роботи є дослідження та аналіз ефективності алгоритмів консенсусу у шардінгових блокчейн-системах, зокрема PBFT (Practical Byzantine Fault Tolerance), із врахуванням впливу візантійських вузлів та впровадження репутаційних механізмів для покращення стійкості та ефективності системи.

КЛЮЧОВІ СЛОВА: PBFT, ШАРДІНГ, БЛОКЧЕЙН.

ABSTRACT

This master's thesis is dedicated to the study of the effectiveness of sharding systems in blockchain networks, especially considering the impact of Byzantine nodes and reputation systems on consensus. The explanatory note consists of an introduction, three chapters, and a list of references. The explanatory note to the qualifying master's thesis contains 73 pages, 17 illustrations, 2 tables, 2 appendix and 47 references to external sources.

The purpose of the work is to study and analyze the effectiveness of consensus algorithms in sharding blockchain systems, in particular PBFT (Practical Byzantine Fault Tolerance), taking into account the influence of Byzantine nodes and the introduction of reputation mechanisms to improve the stability and efficiency of the system.

KEYWORDS: PBFT, SHARDING, BLOCKCHAIN.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ КОНСЕНСУ	10
1.1 Алгоритми консенсу	10
1.2 Проблема візантійських генералів	13
1.3 Practical Byzantine Fault Tolerance (PBFT)	16
Висновки до першого розділу	19
РОЗДІЛ 2. МЕТОДИ ВИРІШЕННЯ ПРОБЛЕМИ МАСШТАБОВАНOSTІ ПРОТОКОЛІВ КОНСЕНСУСУ	21
2.1 Аналіз алгоритмів які вирішили проблему масштабованості	21
2.2 Проблеми масштабованості протоколів консенсусу, що базуються на візантійській угоді	24
2.3 Метод вирішення проблеми масштабованості	26
2.4 Порівняння Cross Sharding Model та Non-cross Sharding Model	29
Висновки до другого розділу	39
РОЗДІЛ 3. МЕТОДИ ВИРІШЕННЯ ПРОБЛЕМИ МАСШТАБОВАНOSTІ ...	41
3.1 Вирішення проблеми за допомогою шардінгу	41
3.2 Розподіл вузлів в шардінгу за допомогою репутації	46
3.3 Залежність ймовірності відмови при зміні даних	49
3.4 Аналіз безпеки шардінгового блокчейну	52
Висновки до третього розділу	53
ВИСНОВКИ	55
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	57
Додатки	63
ДОДАТОК А	64

ДОДАТОК Б	67
-----------------	----

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

PBFT – (англ. Practical Byzantine Fault Tolerance) – Доказ візантійської відмовостійкості

BFT – (англ. Practical Byzantine Fault Tolerance) – Доказ візантійської відмовостійкості

TEE – (англ. Trusted Execution Environment) – Довірене середовище виконання.

PoW – (англ. Proof of Work) – Доказ роботи

PoS – (англ. Proof of Stake) – Підтвердження частки

DPoS – (англ. Delegated Proof of Stake) – Делегована підтвердження частки володіння

ВСТУП

Актуальність теми. Сучасний світ переживає стрімкий науково-технічний розвиток, і все частіше говорять про криптовалюту, біткоїн і блокчейн. Проте поняття блокчейну не є загальновживаним, тому це дало поштовх для дослідження його еволюції, визначення його основних видів та характеристик і прогнозування можливої перспективи.

Блокчейн – це децентралізований розподілений реєстр, у якому пристрої зберігання даних не підключені до єдиного загального сервера. Копії блокчейна можуть зберігатися на мільйонах комп'ютерів одночасно, а дані відкриті для всіх користувачів. Вперше концепція блокчейна була реалізована у криптовалютах у 2008 році. Однак його потенціал настільки високий і всеосяжний, що поступово проникає в різні сфери нашого життя.

Блокчейн використовується в логістичному секторі для безпеки інформації про всі транзакції у ланцюгу постачання, захищаючи їх від несанкціонованого втручання. Це сприяє збільшенню контролю над ефективністю, відповідальністю та прозорістю у проведенні операцій.

Аналіз та вдосконалення алгоритмів блокчейну стає ключовим напрямом досліджень у зв'язку з постійним розвитком технологій та розширенням застосувань блокчейн у різних галузях.

Мета роботи розробити математичні моделі для оцінки ймовірності відмови комітетів, дослідити ефективність та безпеку алгоритмів консенсусу в блокчейн-системах із використанням шардінгу, також проаналізувати впровадження репутаційного механізму для підвищення надійності та продуктивності системи в умовах адаптивних атак.

У рамках роботи будуть вирішені наступні завдання:

- Дослідження шардінгових моделей;
- Аналіз ймовірності досягнення консенсусу;
- Впровадження репутаційного механізму;
- Тестування на різних параметрах;
- Аналіз стійкості до адаптивних атак.

Об'єктом дослідження є блокчейн-системи, що використовують шардинг як механізм масштабування.

Предметом дослідження є методи безпечного розподілу вузлів між комітетами та забезпечення відмовостійкості системи під час адаптивних атак.

Наукова новизна роботи полягає у розробці підходу до випадкового розподілу вузлів між шардами з використанням технології Trusted Execution Environment (TEE), також запропоновано метод обчислення ймовірності відмови комітетів на основі гіпергеометричного розподілу, що дозволяє визначати оптимальні параметри комітетів для досягнення необхідного рівня безпеки.

Практичне значення роботи полягає в тому, що результати роботи будуть корисні для розробників блокчейн-систем, що використовують алгоритми консенсусу типу PBFT та шардінгові механізми для можливості підвищення ефективності, надійності та безпеки блокчейн-систем, а також як основа для подальших досліджень.

Апробація. За темою кваліфікаційної магістерської роботи було опубліковано 3 тези конференції [1 - 3].

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ КОНСЕНСУ

1.1 Алгоритми консенсу

Ключовим аспектом технології блокчейн є визначення того, який саме користувач публікує наступний блок. Це було вирішено впровадженням одного з багатьох можливих моделей консенсусу. Технологія блокчейн передбачає зберігання однієї бази даних на різних серверах шифрування даних. Однак виникає питання про те, які саме вузли «домовляються» між собою, який примірник вважати правильним, і тут потрібен механізм консенсусу.

Алгоритм консенсусу – це механізм, який дозволяє користувачам чи машинам координувати свої дії у розподіленому середовищі [4]. Він повинен гарантувати, що всі агенти в системі можуть домовитися про одне джерело правди, навіть якщо деякі агенти зазнають невдачі. Іншими словами, система повинна бути відмовостійкою.

Алгоритми консенсусу забезпечують прозорість, показуючи всі транзакції в блокчейні, що полегшує відстеження та запобігання шахрайству. Покращуючи ефективність, вузли можуть швидко погоджувати дійсність транзакцій і додавати нові блоки в блокчейн. Досягнення згоди в мережі блокчейн є складною та важливою справою. Оскільки кожен вузел мережі перевіряє новий блок, нові записи транзакцій будуть додані до блокчейну. Зверніть увагу, що блоки після перевірки неможливо змінити або видалити. Структура блокчейнів розроблена таким чином, щоб функціонувати як в надійних, так і в ненадійних мережах, у яких є користувачі, які можуть бути ворожими. Відповідно до прогресу блокчейна, кількість алгоритмів консенсусу постійно зростає.

Одними з найпопулярніших алгоритмів є: Practical Byzantine Fault Tolerance (PBFT), Proof of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS).

Розглянемо як працює кожен з них. Особливість PBFT в тому, що він забезпечує стійкість до візантійських помилок, коли вузли можуть поводитися

довільно або зловмисно. Алгоритм досягає консенсусу через три фази: pre-prepare, prepare, commit [5].

PoW - це один з перших і найбільш популярних алгоритмів консенсусу, що використовується в блокчейнах, таких як Bitcoin. В основі PoW лежить виконання складних математичних обчислень, які потребують значних обчислювальних ресурсів. Майнери змагаються за можливість додати новий блок до блокчейну, розв'язуючи криптографічні задачі, і перший, хто знайде правильне рішення, отримує винагороду. Роль PoW в блокчейнах полягає у забезпеченні безпеки та децентралізації мережі. Завдяки високим вимогам до обчислювальної потужності, зловмисникам стає складно захопити контроль над мережею, що захищає від атак типу «51%». Проте, PoW має свої недоліки – це високе енергоспоживання та необхідність у дорогому обладнанні для майнінгу, що робить процес менш екологічним [6].

PoS - це альтернативний алгоритм консенсусу, який був створений для вирішення проблем, пов'язаних з високим енергоспоживанням PoW. Замість обчислювальної потужності, PoS використовує кількість криптовалют, яку користувач готовий поставити як заставу (stake). Чим більше токенів у користувача, тим більша ймовірність, що він буде вибраний для додавання нового блоку до блокчейну. Основна відмінність від PoW полягає у тому, що PoS не потребує значних обчислювальних ресурсів, що робить його більш екологічно дружнім і енергоефективним. Це дозволяє зменшити витрати на підтримку мережі та збільшити швидкість підтвердження транзакцій.

В алгоритмі PoW вузли змагаються в розв'язуванні криптографічної задачі [7]; перший, хто розв'язав, додає блок до блокчейну, а в алгоритмі PoS вузли обираються на основі їх частки володіння (стейк) криптовалюти [8].

Delegated Proof of Stake (DPoS) – це вдосконалення PoS, де учасники мережі голосують за делегатів, які будуть відповідальні за додавання нових блоків. Це підвищує ефективність роботи мережі і дозволяє швидше досягати

консенсусу, але може призвести до певної централізації, оскільки вибір делегатів може зосередитися в руках невеликої групи учасників. [9].

В таблиці 1.1 наведено порівняння ключових особливостей алгоритмів консенсу.

Таблиця 1.1

Порівняння алгоритмів консенсу

Алгоритм	Переваги	Недоліки
PBFT	Висока швидкість і надійність у невеликих мережах.	Зменшення ефективності зі збільшенням кількості вузлів.
PoW	Високий рівень безпеки і стійкість до атак.	Високе споживання енергії і ресурсоемність.
PoS	Енергоефективніший порівняно з PoW, менше потреб у ресурсах.	Вразливість до центральної концентрації стейків.
DPoS	Висока швидкість транзакцій, більш демократичний процес.	Можливість змови серед делегатів, вразливість до централізації.

Отже, консенсус є ключовим елементом у функціонуванні мереж блокчейн і забезпечує узгодженість, безпеку та децентралізацію даних. Різні алгоритми консенсусу дозволяють технології блокчейн досягати різних цілей, таких як безпека, ефективність, масштабованість і стійкість. Кожен алгоритм має свої переваги та недоліки, і вибір відповідного алгоритму залежить від потреб конкретної мережі блокчейн. Важливість консенсусу полягає в його здатності забезпечити надійність і гнучкість системи за відсутності централізованого органу. Це робить технологію блокчейн унікальною, і очікується, що вона отримає подальший розвиток у децентралізованих фінансах, управлінні даними та інших сферах.

1.2 Проблема візантійських генералів

Як і у випадку з більшістю розподілених обчислювальних систем, учасникам криптовалютних мереж необхідно регулярно коригувати поточний стан блокчейна. Це те, що називається формуванням консенсусу [10]. Однак досягнення консенсусу в розподілених мережах безпечним та ефективним способом є непростим завданням.

Отже, якщо розподілена мережа комп'ютерних вузлів може вийти з ладу або діяти зловмисно, як ми можемо домовитись про рішення? Це фундаментальне питання так званої проблеми візантійських генералів, яка породила концепцію візантійської відмовостійкості [11].

Проблема візантійських генералів - це аналогія в комп'ютерній науці, яка використовується для опису проблеми встановлення і підтримки безпеки в розподіленій мережі. Для вирішення цієї проблеми чесні вузли (наприклад, комп'ютери або інші фізичні пристрої) повинні мати можливість досягти консенсусу, незважаючи на наявність нечесних вузлів. Це означає, що більшість вузлів повинні встановити набір правил і домовитися про те, як забезпечити дотримання цих правил в мережі [12].

У 1982 році Леслі Лампорт, Роберт Шостак і Маршалл Піз опублікували дослідницьку роботу під назвою "Проблема візантійських генералів" (The Byzantine Generals Problem) [13].

Аналогія, використана в питанні про візантійських генералів, в основному виглядає так: деякі підрозділи візантійської армії стоять поблизу ворожих міст і готуються до битви. Різні генерали можуть спілкуватися один з одним тільки через гінців. Їм необхідно узгодити план спільних дій [14].

Розвиток подій наступний:

1. Якщо усі легіони атакують - вони перемагають;
2. Якщо усі легіони відступають - вони зберігають людей (теж вдалий результат);
3. Якщо частина атакує, частина відступає - армія зазнає поразки.

Однак потрібно припустити, що деякі генерали є зрадниками, які хочуть перешкодити вірним генералам домовитись про спільний план дій. Тому потрібен алгоритм, який гарантує, що невелика група зрадників не зможе порушити зв'язок. Отже, щоб вирішити проблему візантійські генерали повинні обмінятися один з одним інформацією, виключивши помилкові дані. Точніше, вони повинні обмінятися інформацією про чисельність легіонів, вірних Візантії, і зробити висновки про чисельність легіону зрадників. Завдання передбачає, що при N кількості генералів зрадниками можуть виявитися $N-1$.

Проблема розглядається в контексті усних повідомлень і письмових підписаних повідомлень, яким приписуються різні властивості [15]. Усні повідомлення характеризуються наступним чином:

A1. Кожне повідомлення, яке надсилається, доставляється правильно (повідомлення не губляться).

A2. Одержувач повідомлення знає, хто його відправив.

A3. Відсутність повідомлення можна виявити.

Для повідомлень з такими властивостями було доведено, що консенсус не може бути досягнутий з трьома генералами, якщо передбачається, що один з них є зрадником [16].

Письмові, підписані повідомлення мають всі властивості (A1-A3) усних повідомлень і додатково характеризуються властивостями, наведеними нижче:

A4. Підпис лояльного генерала неможливо підробити.

A5. Будь-хто може перевірити справжність підпису.

Припускаючи наведені вище властивості підписаних повідомлень, показано, що консенсус можливий лише з трьома генералами, використовуючи просту функцію більшості голосів. Розв'язок далі узагальнено для розгляду множинних сценаріїв помилок і зроблено висновок: для того, щоб терпіти m генералів-зрадників, потрібно $3m + 1$ лояльний генерал і $m + 1$ раунд обміну інформацією. На рисунках 1.1 та 1.2 зображено

роботу даного алгоритму коли зрадник генерал та командуючий відповідно, при $m = 1$ та $n = 4$ [17].

лейтенанта 2— Нехай C — командир, а $L[i]$ — лейтенант i :

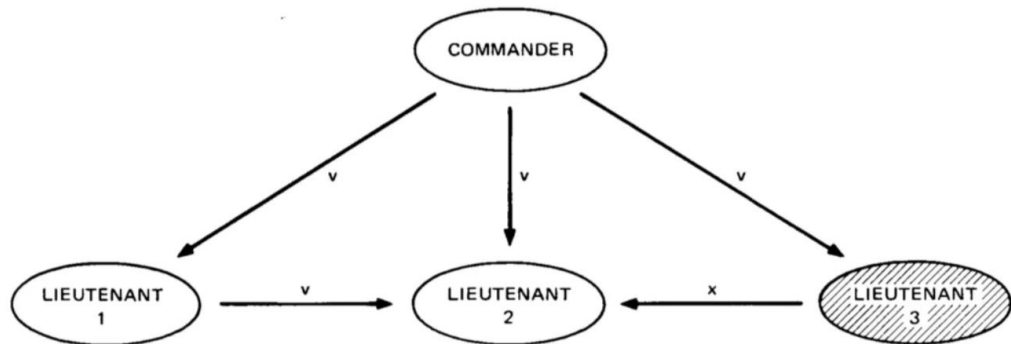


Рис. 1.1 Алгоритм коли зрадник генерал

Кроки:

1. Командир посилає v до всіх лейтенантів
2. $L1$ надсилає v до $L2$ | $L3$ надсилає x до $L2$
3. $L2 \leftarrow \text{більшість}(v, v, x) == v$

Остаточне рішення приймається більшістю голосів від $L1$, $L2$, $L3$, і в результаті досягнуто консенсусу.

Розглянемо випадок командира як зрадника:

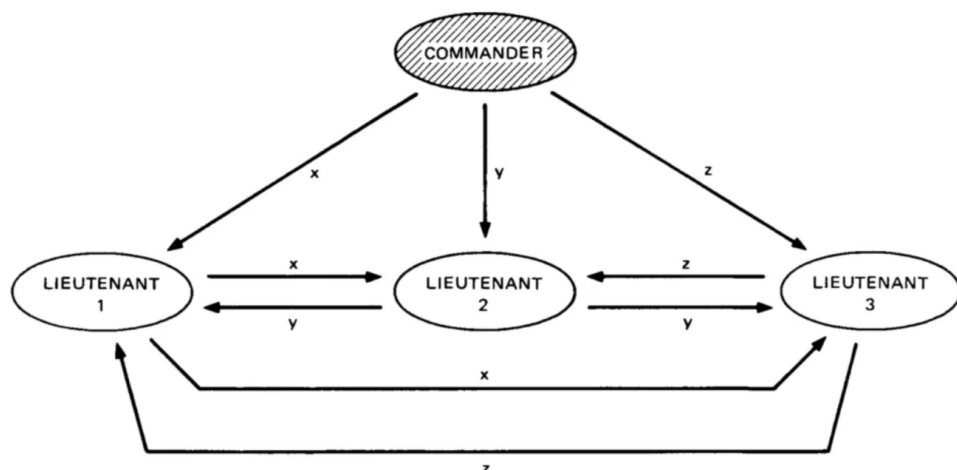


Рис. 1.2 Алгоритм коли зрадник командир

Кроки:

1. Командир надсилає x , y , z на $L1$, $L2$, $L3$ відповідно

2. $L1$ надсилає x до $L2, L3$ | $L2$ надсилає y до $L1, L3$ | $L3$ надсилає z до $L1, L2$
3. $L1 \leftarrow \text{більшість}(x,y,z)$ | $L2 \leftarrow \text{більшість}(x,y,z)$ | $L3 \leftarrow \text{більшість}(x,y,z)$

Усі вони мають однакову цінність, тому досягається консенсус. Зверніть увагу на те, що навіть якщо x, y, z різні, значення більшості (x, y, z) однакове для всіх 3 лейтенантів. У випадку, коли x, y, z є абсолютно різними командами, ми можемо припустити, що вони діють за параметром за замовчуванням `retreat`.

Загальна проблема візантійський генералів важлива для криптовалют, так як, показує, як децентралізована мережа може досягти консенсусу перед обличчям потенційно зловмисних або дисфункціональних вузлів. Вирішення цієї проблеми гарантуватиме надійність і безпеку блокчейну, дозволяючи учасникам довіряти системі без необхідності довіряти один одному.

Також завдання візантійських генералів дає основу для розуміння того, як децентралізовані криптовалютні мережі можуть досягати консенсусу в середовищі, де деякі учасники можуть обманювати або порушувати правила. Різні протоколи консенсусу, такі як Proof-of-Work, Proof-of-Stake і PBFT, пропонують різні підходи до вирішення цієї проблеми [18].

1.3 Practical Byzantine Fault Tolerance (PBFT)

Practical Byzantine Fault Tolerance (PBFT) — це форма алгоритму Byzantine Fault Tolerant (BFT), який використовується консенсусним алгоритмом у деяких блокчейнах.

BFT - це алгоритм, розроблений для вирішення загальної проблеми візантійських генералів. Він дозволяє лояльним генералам йти на компроміс навіть за наявності зрадників; основне припущення BFT полягає в тому, що певний відсоток генералів є чесними [19].

PBFT — це оптимізація алгоритму BFT, розроблена для того, щоб зробити BFT практичним для великих мереж. Одним із основних способів цього досягти є усунення зв'язку між кожним вузлом у мережі блокчейн.

Доказ візантійської відмовостійкості (PBFT) - це послідовний алгоритм, який використовується в розподілених системах для досягнення

консенсусу в мережі вузлів, навіть за наявності несправних або шкідливих вузлів [20]. Він спеціально розроблений для обробки помилок, пов'язаних із довільною та шкідливою поведінкою вузлів у розподіленій мережі. На рисунку 1. показано роботу алгоритму у звичайному випадку без первинних несправностей. Репліка 0 є первинною, репліка 3 несправна, а С – клієнт.

Ось основні етапи, через які працює PBFT (рис. 1.3) [21]:

Запит: клієнт надсилає запит на виконання операції до головного вузла.

Перед підготовка: всі інші частини системи отримують повідомлення перед підготовки від головного вузла.

Підготовка (Prepare): Щоб підтвердити отримання повідомлення pre-prepare, всі вузли обмінюються повідомленнями prepare.

Commit: Кожен вузол отримує повідомлення, що підтверджує готовність виконати запит.

Відповідь (Reply): Клієнту надсилається відповідь про виконання операції через Вузли.

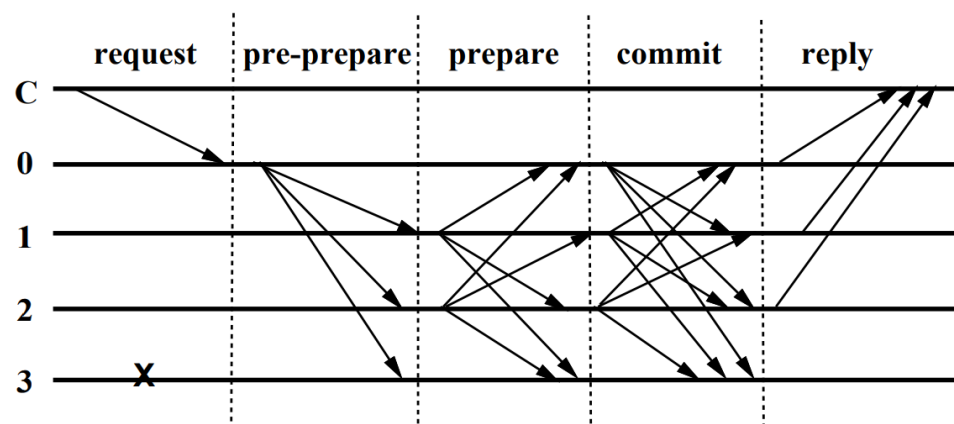


Рис. 1.3 Основні етапи PBFT

У PBFT мережа вузлів утворює послідовну групу, де кожен вузол, у свою чергу, виконує роль лідера. Процес узгодження включає серію круглих столів, в яких лідер пропонує Значення або серію транзакцій, а інші вузли мережі голосують за запропоноване значення, спілкуючись один з одним для досягнення згоди щодо узгодженого значення.

Для досягнення візантійської відмовостійкості PBFT вимагає, щоб в мережі була мінімальна кількість вузлів, які функціонували б коректно. Дві

третини вузлів працюють чесно і дотримуються протоколу, але PBFT не дозволяє мережі узгоджувати послідовні значення, навіть якщо деякі вузли несправні або шкідливі.

Практичний візантійський алгоритм відмовостійкості (PBFT) широко використовується в блокчейн-системах з консенсусним алгоритмом. Його мета полягає в тому, щоб дозволити мережі комп'ютерів колективно домовлятися про порядок і дійсність транзакцій, навіть коли має справу з несправними або зловмисними вузлами. Для досягнення консенсусу PBFT покладається на обміні повідомленнями та раундах. До тих пір, поки більше дві третини вузлів працюють чесно, PBFT гарантує домовленість про стан блокчейну. Цей алгоритм знаходить широке застосування в дозволенних блокчейн-мережах, де учасники відомі і довіряють один одному. Завдяки використанню PBFT, ці мережі мають швидке завершення транзакцій та підвищену пропускну здатність. На рис. 1.4 зображено блок-схему PBFT [22].

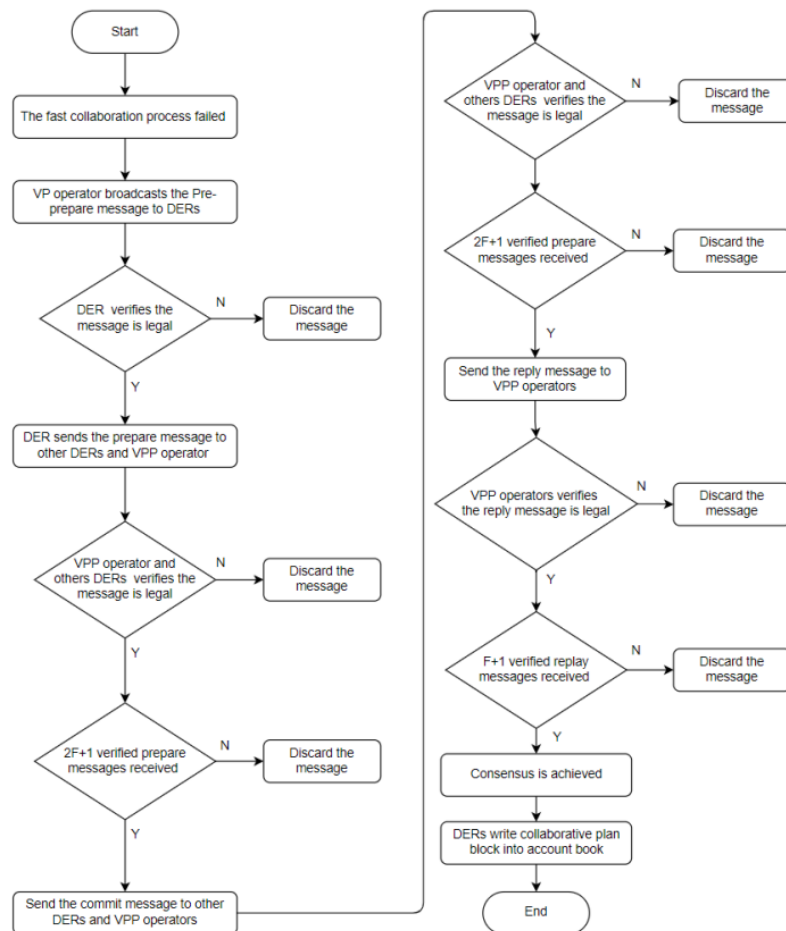


Рис. 1.4 Блок-схема PBFT

Деякі з переваг PBFT в консенсусі блокчейну є наступними:

- PBFT спеціально розроблений як алгоритм відмовостійкості і може реагувати на збої в роботі вузлів. Це важливо в додатках, де вузли можуть вийти з ладу без попередження, наприклад, в консенсусі блокчейну.
- багато алгоритмів консенсусу в блокчейні, такі як Proof of Work і Proof of Stake, мають лише імовірнісну визначеність. Це означає, що підтверджені блоки можуть бути видалені з розподіленого реєстру після реорганізації.
- PBFT - це візантійський алгоритм відмовостійкості. Це означає, що він може впоратися з наявністю певної кількості зловмисних вузлів в мережі.

На додачу до цих переваг, PBFT має деякі недоліки. Деякі з них є наступними:

- в PBFT головний вузол приймає рішення про вміст наступного блоку в блокчейні і керує процесом затвердження. Це призводить до централізації, що суперечить основним принципам блокчейну.
- PBFT вимагає повідомлень від певного відсотка вузлів у мережі для завершення вмісту блоку. Це означає, що кількість повідомлень залежить від розміру мережі, що обмежує масштабованість.
- з тієї ж причини, що і масштабованість, використання пропускну здатності мережі є проблемою в PBFT. Чим більше вузлів взаємодіє, тим більше споживається пропускну здатність.
- PBFT приймає рішення на основі отримання повідомлень від певної кількості вузлів у мережі. Якщо зловмисник має контроль над різними вузлами, він має право більшості голосів і може затверджувати шкідливі блоки [23].

Висновки до першого розділу

У першому розділі було розглянуто що таке алгоритми консенсу, здійснено ґрунтовний аналіз існуючих алгоритмів консенсусу, що використовуються в блокчейн-системах. Розписано проблему візантійських генералів та розглянуто алгоритм її вирішення. Також було розглянуто алгоритм PBFT в основі якого лежить візантійська відмовостійкість. У

результаті аналізу було встановлено, що PBFT є ефективним рішенням для блокчейн-систем з обмеженою кількістю учасників завдяки високій швидкості консенсусу та відмовостійкості. Проте він не масштабується при зростанні кількості вузлів.

РОЗДІЛ 2. МЕТОДИ ВИРІШЕННЯ ПРОБЛЕМИ МАСШТАБОВАНOSTІ ПРОТОКОЛІВ КОНСЕНСУСУ

2.1 Аналіз алгоритмів які вирішили проблему масштабованості

Solana - це передовий блокчейн-проект з відкритим вихідним кодом, який має на меті використовувати різні проривні технології для наступного покоління DApps (рис. 2.1).



Рис.2.1 Solana

Він має максимальну пропускну здатність понад 50 000 TPS і час блоку всього 400 мс. Проект спрямований на створення масштабованої, безпечної та високо децентралізованої платформи, яка потенційно може підтримувати тисячі вузлів без шкоди для пропускну здатності, допомагаючи уникнути деяких проблем, з якими стикаються конкуруючі системи [24].

Токен SOL заснований на унікальній комбінації технологій, які гарантують високу продуктивність і масштабованість. Основною технологією є Proof of History (PoH), винайдена Анатолієм Яковенко, яка дозволяє створювати мітку часу для кожної транзакції, гарантуючи її узгодженість і порядок в часі без необхідності участі всіх вузлів мережі. PoH дозволяє створювати мітку часу для кожної транзакції, гарантуючи її узгодженість і порядок в часі без необхідності участі всіх вузлів мережі. Це значно прискорює процес транзакцій, оскільки вузли не витрачають час на узгодження. Ще одна ключова технологія - Tower BFT (візантійська відмовостійкість). Tower BFT використовує історичний запис, створений PoH,

щоб узгодити стан блокчейну, дозволяючи мережі досягати консенсусу швидше та ефективніше.

Протокол Tower BFT в мережі Solana базується на концепції Recoverable Persistence Interval, яка адаптивно враховує фактичний розподіл часток між верифікаторами [25]. У цьому протоколі кожен валідатор має частку, що відображає його вплив на процес консенсусу, а загальна вага всіх валідаторів розраховується як сума кожної частки. Для того, щоб блок був затверджений, загальна вага W повинна досягти певного порогу T , де N - загальна кількість валідаторів (вузлів) в мережі Solana.

Кожен валідатор має певну вагу w_i , яка визначає його вплив на консенсус. Загальна вага всіх валідаторів розв'язується за формулою:

$$W = \sum_{i=1}^N w_i. \quad (2.1)$$

Для того, щоб даний блок був схвалений і доданий до ланцюга (тобто був досягнутий консенсус), має бути досягнутий певний поріг T від загальної ваги W в голосах. Тобто має виконуватися наступна умова, де V - множина валідаторів, що проголосували за даний блок.

$$\sum_{i \in V} w_i \geq T \quad (2.2)$$

Традиційно в системах BFT, $T = \frac{2W}{3} + \epsilon$,

де ϵ - невеликий зсув, щоб уникнути еквівалентності. Це забезпечує стійкість до $\frac{1}{3}$ візантійських (зловмисних) вузлів.

Максимальна кількість візантійських вузлів (зловмисників), яку може витримати система, нехай буде f . Це значення залежить від загального розподілу ставок між валідаторами. Визначимо множину всіх верифікаторів у порядку спадання ставок: $\{v_1, v_2, \dots, v_N\}$, де $w_{v_1} \geq w_{v_2} \geq \dots \geq w_{v_N}$.

Максимальну кількість візантійських вузлів f , що можуть бути в системі без загрози досягнення консенсусу визначається за формулою:

$$\sum_{j=1}^f w_{v_j} \leq \frac{W}{3} \quad (2.3)$$

Іншими словами, f - це максимальна кількість аутентифікаторів, сумарна частка яких не перевищує $\frac{1}{3}$ загального стейку в мережі.

Поріг авторизації блоку T визначається наступним чином.

$$T = \frac{2W}{3} + \sum_{j=f+1}^N w_{v_j} \quad (2.4)$$

Іншими словами, T - це $\frac{2}{3}$ сумарного стейку плюс частки всіх валідаторів, крім найбільшого j .

Цей підхід, який називається відновлюваним інтервалом, гарантує, що консенсус буде досягнутий шляхом підтвердження блоків і жертвування рештою вузлів, навіть якщо всі f найбільших валідаторів є візантійськими.

Протокол візантійського BFT Tower Solana, який ітеративно працює над досягненням цього порогу T для кожного нового блоку. Валідатори обмінюються повідомленнями Vote і квитанціями про готовність Commit. Кожна нова ітерація починається з оновлення стану валідатора на основі попередніх даних [26].

Tower BFT може більш гнучко адаптуватися до фактичного розподілу ставок між валідаторами, використовуючи концепцію поновлюваного діапазону T замість фіксованих $\frac{2W}{3}$. Це робить мережу Solana більш стійкою до атак візантійського вузла і консенсусу, також збільшує безпеку і продуктивність мережі, забезпечуючи надійність навіть за наявності недобросовісних вузлів.

Проте як і кожна блокчейн платформа Solana має свої недоліки.

Незважаючи на заяви про децентралізацію, Solana критикують за її надмірно централізовану структуру зі значною кількістю токенів, зосереджених у невеликій кількості інвесторів і валідаторів.

Solana неодноразово стикалася з проблемами масштабування і перебоями в роботі мережі, що робило платформу тимчасово непридатною для використання і ставило під сумнів її надійність.

Валідатори в мережі Solana потребують високопродуктивного обладнання, що може створювати бар'єри для входу нових учасників і заохочувати централізацію.

Solana стикається з жорсткою конкуренцією з боку інших високопродуктивних блокчейн-платформ, таких як Ethereum 2.0, що ускладнює залучення нових користувачів і розробників.

Як і всі криптовалюти, Solana схильна до регуляторних ризиків, включаючи потенційні зміни в законодавстві та правовому статусі криптовалют, які можуть вплинути на її розвиток і використання.

2.2 Проблеми масштабованості протоколів консенсусу, що базуються на візантійській угоді

Традиційно слабкою стороною блокчейнів є проблема гармонізації безпеки, децентралізації та масштабованості – здатності обробляти велику кількість транзакцій. А чим популярніша мережа, тим більше в ній проводиться операцій.

Протоколи консенсусу, які базуються на візантійській угоді (BFT, Byzantine Fault Tolerance), є ефективними для забезпечення безпеки та стійкості блокчейн-систем, навіть у випадках, коли частина учасників діє зловмисно. Однак, при масштабуванні таких протоколів виникає низка проблем.

Найчастіше проблеми з масштабованістю виникають через те, що візантійська угода часто вимагає компромісів між учасниками, що може затримувати прийняття рішень; велика кількість учасників може ускладнити ефективну комунікацію та взаємодію; високі вимоги до обчислювальних ресурсів для обробки великої кількості даних та взаємодії; збільшення кількості учасників може зробити систему більш уразливою до атак та зламів.

Як правило, масштабування протоколів для BFT фокусується або на зменшенні кількості вузлів, необхідних для перенесення візантійських несправностей, або на зменшенні складності зв'язку протоколу, щоб дозволити більші розміри мережі [27].

У протоколах BFT кожна репліка зазвичай зв'язується безпосередньо з кожною іншою реплікою під час кожного раунду консенсусу, що призводить до складності комунікації. Це вимагає обміну великою кількістю повідомлень, що збільшується експоненційно із зростанням кількості нод. Наприклад, у класичних BFT протоколах кількість повідомлень пропорційна $O(n^2)$ де n це кількість учасників. Це прийнятно, коли їх зазвичай 4 або не набагато більше, але стає непрактичним, якщо вони можуть представляти сотні або тисячі біткоїн-вузлів [28]. Проте ByzCoin за допомогою колективного підписання (CoSi) зміг зменшити складність зв'язку, тим, що замінює прямий зв'язок із автентифікацією MAC цифровими підписами, щоб зробити автентифікацію доступною для передачі й таким чином уможливити розріджені шаблони зв'язку, які можуть зменшити звичайну затримку зв'язку з $O(n^2)$ до $O(n)$ [29].

PBFT забезпечує як безпеку, так і живучість за умови, що дотримується межа, де відбувається не більше f помилок серед $3f+1$ учасників. Кожен раунд PBFT складається з 3 окремих фаз [30]. Першою є фаза попередньої підготовки, на якій первинний перегляд оголошує наступний запис, з яким система повинна погодитися. Отримавши попередню підготовку, кожен вузол перевіряє правильність пропозиції і розсилає групі повідомлення про підготовку. Ноди чекають, поки не зберуть кворум $(2f+1)$ підготовчих повідомлень, і публікують це спостереження з повідомленням про фіксацію. Потім вони чекають кворуму $(2f+1)$ повідомлень про фіксацію, щоб переконатися, що достатня кількість вузлів готова застосувати будь-яке рішення. Чим менше n , тим менші витрати на зв'язок при тій самій кількості відмов. Однакова кількість вузлів n також означає, що мережа може витримати більшу кількість вузлів, що вийшли з ладу. Одним із способів зменшити кількість вузлів є випадковий вибір меншого набору консенсусних

вузлів в якості комітету, який ініціює процес консенсусу. Менші комітети консенсусу забезпечують кращу пропускну здатність, оскільки вони досягають більшої пропускну здатності з меншими витратами на зв'язок. Одним з методів зменшення кількості вузлів є шардинг. Шардинг (sharding) – це рішення для масштабування блокчейну: це просто метод, який дозволяє мережі обслуговувати більшу кількість користувачів за допомогою швидших транзакцій [31]. Зменшення кількості вузлів - це використання технік, щоб знизити n з $3f + 1$ до $2f + 1$.

Також для зменшення навантаження можна використати зовнішні компоненти (наприклад, надійне обладнання) або зменшити модель системи.

2.3 Метод вирішення проблеми масштабованості

Спроби збільшити масштабованість блокчейну призвели до двох різних напрямків: рішення Level 1 і рішення Level 2.

Level 1 відноситься до базового рівня архітектури блокчейну. Це базова структура мережі блокчейн. Біткоїн, Ефіріум і ланцюжок BNB є прикладами блокчейнів першого рівня. Level 2 відноситься до мереж, побудованих поверх інших блокчейнів. Іншими словами, якщо Біткоїн - це Level 1, то мережа Lightning, яка працює поверх нього, є прикладом Level 2 [32].

Рішення рівня 1 вимагають внесення змін до базового програмного забезпечення блокчейну. Ці рішення також відомі як рішення для масштабування мережі. Їх основна мета - поліпшити фундаментальні властивості мережі блокчейн, такі як максимальний розмір блоку і швидкість перевірки блоків. Шардінг, Standalone Witness (SEGWIT) і хардфоркінг є поширеними варіантами масштабування блокчейнів першого рівня [33].

Зазвичай, кожен вузол блокчейну відповідає за обробку всього обсягу транзакцій в мережі. Такий тип обробки даних зазвичай називають послідовною обробкою. Це означає, що кожен вузол повинен підтримувати і зберігати всю важливу інформацію, таку як залишки на рахунках і історію транзакцій. По суті, кожен вузол повинен обробляти всі мережеві операції,

дані і транзакції. У цій моделі запис усіх транзакцій на кожному вузлі підвищує безпеку блокчейну, але також значно сповільнює обробку даних. Саме тут на допомогу приходить паралельна обробка даних, яка може виконувати велику кількість операцій одночасно. Шардинг може бути ефективним рішенням цієї проблеми, оскільки він розділяє або розподіляє навантаження транзакцій по всій мережі блокчейн. Це означає, що не всім вузлам потрібно управляти і обробляти все навантаження блокчейну.

Шардинг являє собою технологію масштабування бази даних, основна суть якої в тому, щоб розбити її на окремі сегменти, кожен з яких можна було б винести на окремий сервер. Простими словами, шардинг дозволяє одному блокчейну розділити себе на кілька менших і більш керованих блокчейнів, які називаються шард-ланцюгами, також відомими як рівні даних. Коли всі шарди зібрані разом, вони утворюють всю базу даних [34]. Однак замість того, щоб кожна нода мережі зберігала повну історію транзакцій блокчейну, кожна нода закріплена лише за декількома з цих менших шардів. Це значно зменшує навантаження даних на кожную окрему ноду, збільшуючи швидкість транзакцій. Крім того, це дозволяє мережі обробляти кілька потоків транзакцій паралельно, що значно покращує її масштабованість.

Хоча шардинг пропонує кілька потенційних переваг для підвищення ефективності мереж блокчейн, цей процес також створює певні проблеми. До потенційних вразливостей, які створює шардинг, відносяться наступні: зменшення децентралізованості мережі, транзакції між шардами є складними і можуть призвести до подвійних витрат, якщо ними не керувати належним чином, якщо певний шард недоступний, коли він потрібен (через те, що вузол, який підтримує цей шард, перебуває в автономному режимі), це може призвести до проблем з доступністю даних і порушити роботу всієї мережі, синхронізація нод може спричинити затримки в мережі через час, необхідний для обміну й оновлення інформації між різними нодами, також може призвести до проблем для централізованих валідаторів, таких як змова та викрадення 51% [35]. Іншими словами, фрагментація невеликих мереж може

дозволити хакерам атакувати валідаторів. Чим менше валідаторів, тим легше зловмисникам атакувати систему. Однак фрагментація досить щільної мережі не повинна становити проблему.

Одним з найпопулярнішим рішенням який використовує шардинг є Zilliqa та Harmony.

Zilliqa — це блокчейн-платформа, яка може бути масштабована за допомогою технології шардингу. Вона стала першою публічною платформою блокчейну, яка впровадила шардинг на своїй основній мережі, коли була запущена в 2017 році [36].

Zilliqa може обробляти велику кількість транзакцій одночасно завдяки розділенню своєї мережі на кілька шардів. Також вона створює безпечні смарт-контракти за допомогою мови Scilla [37]. Завдяки шардингу Zilliqa може розвиватися лінійно, розширюючи свою мережу, також шардинг гарантує високу безпеку.

Harmony — це блокчейн-платформа, яка використовує технологію шардингу для збільшення масштабованості та швидкості [38].

Harmony ділить свою мережу на кілька шардів, що дозволяє швидко обробляти транзакції. Також вона використовує алгоритм EPoS, щоб зменшити централізацію та підтримувати делегування стейкінгу. Harmony використовує шардинг для розділення мережі блокчейну на паралельні мережі, які називаються сегментами, для підвищення швидкості та ефективності мережі за нижчою ціною. Завдяки цій стратегії транзакції розраховуються на платформі менш як за три секунди, а вартість — у 100 разів менша за інші блокчейн-транзакції [39].

Основним обмеженням шардингу є те, що він є оптимальною лише тоді, коли операції залишаються на одній гілці. Наприклад, в Harmony координація між шардами в міжшардових транзакціях є складною через необхідність забезпечення атомарності та консистентності даних. Harmony використовує механізм асинхронного підтвердження міжшардових транзакцій, що може призводити до затримок. Так само Zilliqa стикається з

складнощами масштабування для транзакцій, які включають кілька шардів, оскільки шарди вимагають додаткової координації, що спричиняє затримки під час перевірки міжшардових транзакцій.

Загалом, шардинг є важливим кроком у вирішенні трилеми блокчейну. Незважаючи на нові складнощі та потенційні недоліки, його потенціал для покращення масштабованості без шкоди для децентралізації є дуже перспективним для майбутнього блокчейн-мереж [40].

2.4 Порівняння Cross Sharding Model та Non-cross Sharding Model

Кількість вузлів у шардингу менша, ніж у всій мережі, що неминуче несе ризики для безпеки. Деякі дослідники вивчали розподіл вузлів у шардингах за певними правилами, щоб уникнути надмірної кількості зловмисних вузлів, розподілених у шардингу, наприклад, теорія ігор. Однак у більшості досліджень вузли розподіляються на шардинги випадковим чином.

На рисунку 2.2 показано основні етапи транзакції без перехресного шардингу [41].

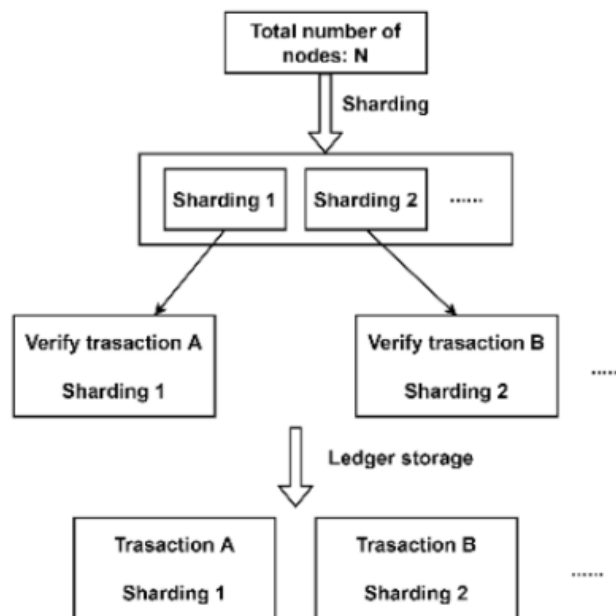


Рис. 2.2 Без перехресний шардинг

По-перше, всі вузли мережі випадковим чином розподіляються на кожен шардінг. На етапі консенсусу вузли не можуть змінити шард, до якого вони належать. Другий - етап верифікації транзакцій, де транзакції можуть

відбуватися і перевірятися тільки між вузлами в межах одного шардингу. Нарешті, є етап зберігання реєстру, де вузли в одному шардингу зберігають тільки ті транзакції, які обробляються в цьому конкретному шардингу. У традиційній блокчейн-системі ноди повинні зберігати всі транзакції в мережі. Тому ресурси зберігання вузлів можна заощадити за рахунок шардингу.

Тепер розглянемо блок-схему транзакції перехресного шардингу (рис.2.3).

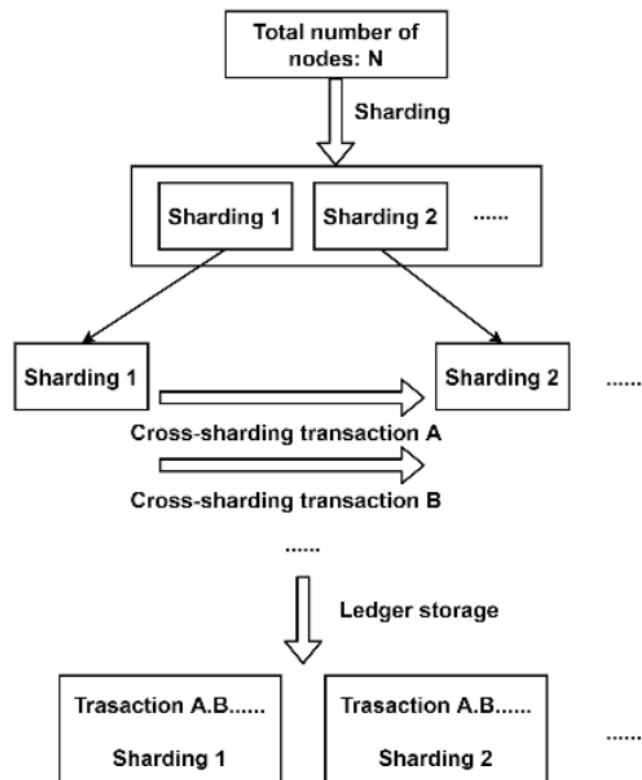


Рис. 2.3 Перехресний шардінг

Перші кроки в перехресному шардингу такі самі як і в неперехресному шардингу вузли мережі випадково розподіляються на кожен шардінг, та відбувається верифікація транзакцій.

Найочевидніша відмінність перехресного шардингу від неперехресного полягає в тому, що в перехресному шардингу транзакції можуть бути згенеровані з часткових вузлів різних шардів [42]. Ці транзакції вимагають координації між різними шардінгами для забезпечення консистентності та

атомарності даних. В таблиці 2.1 також наведені інші відмінності між двома моделями.

Таблиця 2.1

Порівняння Cross Sharding та Non-cross Sharding Model

Характеристика	Cross Sharding Model	Non-cross Sharding Model
Визначення	Транзакції можуть охоплювати кілька шардів.	Транзакції виконуються лише всередині одного шарду.
Масштабованість	Обмежена масштабованість через складність координації між шардами.	Вища масштабованість завдяки ізольованій обробці транзакцій.
Процес обробки транзакцій	Потребує координації між шардами для виконання транзакцій.	Транзакції обробляються локально в одному шарді.
Продуктивність	Залежить від ефективності координації між шардами.	Вища продуктивність завдяки ізольованій обробці транзакцій.
Витрати на комунікацію	Високі через міжшардну комунікацію.	Низькі, оскільки немає потреби у міжшардній взаємодії.
Стійкість до атак	Вразливіший через можливі помилки в координації.	Менш вразливий через відсутність міжшардної взаємодії.

Для більшої наочності було змодельовано поведінку обох підходів шардингу.

Програма складається з трьох основних функцій, кожна з яких виконує специфічні завдання для симуляції та порівняння моделей шардування.

Функція `cross_sharding_simulation(nodes, shards, delay)` призначена для симуляції моделі перехресного шардування, де транзакції можуть відбуватися між різними шардами. Основною метою є моделювання обробки транзакцій,

які потребують взаємодії між різними частинами системи, включаючи комунікаційні затримки.

Функція починається з фіксації поточного часу (`start_time`) для подальшого вимірювання тривалості виконання симуляції. Потім створюється словник `shard_map`, який використовується для зберігання розподілу вузлів по шардам. Ключами цього словника є назви шардів (наприклад, "Shard_1", "Shard_2" тощо), а значеннями — списки вузлів, що належать до відповідного шарду.

Далі відбувається розподіл вузлів по шардам. Для кожного вузла генерується випадковий номер шарду за допомогою функції `random.randint(1, shards)`, що забезпечує випадковий та потенційно нерівномірний розподіл ресурсів між шардами. Вузол додається до відповідного списку в `shard_map`.

Після розподілу вузлів функція симулює транзакції між шардами за допомогою двох вкладених циклів. Зовнішній цикл перебирає всі шардами, а внутрішній цикл перебирає всі шардами, що йдуть після поточного шарду зовнішнього циклу, тим самим уникнувши повторного оброблення однакових пар. Для кожної пари шардів виконується пауза (`time.sleep(delay)`), яка моделює затримку, пов'язану з обміном транзакціями між цими шардами.

Наприкінці симуляції фіксується час завершення (`end_time`), а різниця між `end_time` та `start_time` повертається як результат виконання функції, що представляє загальний час виконання симуляції перехресного шардування.

```
def cross_sharding_simulation(nodes, shards, delay):
```

```
    start_time = time.time()
```

```
    shard_map = {}
```

```
    for i in range(shards):
```

```
        shard_map[f"Shard_{i+1}"] = []
```

```
    # Розподіл вузлів по шардах
```

```
    for node in range(1, nodes + 1):
```

```
        shard_id = random.randint(1, shards)
```

```
        shard_map[f"Shard_{shard_id}"].append(node)
```

```
    # Симуляція транзакцій між шардом
```

```
    for i in range(shards):
```

```

        for j in range(i + 1, shards):
            time.sleep(delay)

    end_time = time.time()
    return end_time - start_time

```

Функція `non_cross_sharding_simulation(nodes, shards)` призначена для симуляції моделі неперехресного шардування, де транзакції обробляються локально в межах кожного шарду без необхідності взаємодії з іншими шардами. Мета полягає в моделюванні обробки транзакцій, які не вимагають комунікаційних затримок між шардами.

Подібно до попередньої функції, `non_cross_sharding_simulation` починається з фіксації поточного часу (`start_time`). Далі створюється словник `shard_map`, який використовується для зберігання розподілу вузлів по шардах. Відмінність полягає в тому, що вузли розподіляються рівномірно серед шардів. Це досягається шляхом обчислення кількості вузлів на кожен шард (`nodes_per_shard`) через цілочисельне ділення загальної кількості вузлів на кількість шардів.

Вузли призначаються до шардів за допомогою циклу, де кожному шарду присвоюється відповідний діапазон вузлів без випадковості. Це забезпечує рівномірний та передбачуваний розподіл ресурсів між шардами, що є характерним для моделі неперехресного шардування.

Після розподілу вузлів функція симулює транзакції, які обробляються локально в межах кожного шарду. У даній реалізації симуляція представлена порожнім циклом, що символізує відсутність необхідності у взаємодії між шардами та, відповідно, відсутність комунікаційних затримок.

Наприкінці симуляції фіксується час завершення (`end_time`), а різниця між `end_time` та `start_time` повертається як результат виконання функції, що представляє загальний час виконання симуляції неперехресного шардування.

```

def non_cross_sharding_simulation(nodes, shards):

    start_time = time.time()

    shard_map = {}
    nodes_per_shard = nodes // shards

```

```

    for i in range(shards):
        start_index = i * nodes_per_shard
        end_index = start_index + nodes_per_shard
        shard_map[f"Shard_{i+1}"] = list(range(start_index + 1,
end_index + 1))

    # Симуляція локальних транзакцій
    for shard in shard_map.values():
        pass

    end_time = time.time()
    return end_time - start_time

```

Функція `compare_sharding_models(nodes, shards, delay)` служить для порівняння двох моделей шардування — перехресного та неперехресного. Вона викликає обидві симуляційні функції з переданими параметрами та виводить результати їх виконання, що дозволяє аналізувати різницю у продуктивності між цими моделями.

Функція спочатку викликає `cross_sharding_simulation` з переданими параметрами (`nodes, shards, delay`) та зберігає отриманий час виконання у змінній `cross_time`. Потім викликається `non_cross_sharding_simulation` з параметрами (`nodes, shards`), і отриманий час виконання зберігається у змінній `non_cross_time`.

Після отримання обох результатів функція виводить їх на екран у форматі, що забезпечує високу точність — до чотирьох десяткових знаків. Це дозволяє чітко бачити різницю у часі виконання між двома моделями шардування (див. рис. 2.4).

```

def compare_sharding_models(nodes, shards, delay):

    cross_time = cross_sharding_simulation(nodes, shards, delay)
    non_cross_time = non_cross_sharding_simulation(nodes, shards)

    print(f"Cross Sharding Time: {cross_time:.4f} seconds")
    print(f"Non-Cross Sharding Time: {non_cross_time:.4f} seconds")

```

```

Cross Sharding Time: 3.2142 seconds
Non-Cross Sharding Time: 0.0330 seconds

```

Рис. 2.4 Симуляція Non-Cross Sharding та Cross Sharding

Порівняння двох моделей шардування базується на вимірюванні часу виконання симуляцій, що відображає ефективність обробки транзакцій у кожній з моделей. Відмінність між цими моделями полягає у наявності або відсутності комунікаційних затримок, що суттєво впливає на загальний час виконання транзакцій.

У моделі перехресного шардування затримки між шардами відображають реальні умови взаємодії між частинами системи, що може призвести до збільшення загального часу обробки транзакцій. У моделі неперехресного шардування відсутність таких затримок дозволяє швидше обробляти транзакції, однак може обмежувати гнучкість та масштабованість системи у випадку складних взаємодій між даними.

Отже, обидві моделі мають свої переваги та недоліки, проте як видно з таблиці, та моделі симуляції Non-cross Sharding Model має декілька переваг у порівнянні з Cross Sharding Model, особливо коли йдеться про простоту, швидкість обробки транзакцій, затримку, надійність, вартість впровадження та продуктивність [43].

Тепер потрібно проаналізувати розподіл вузлів і зловмисних вузлів в рамках без перехресної моделі. Оскільки розподіл вузлів є випадковим, ймовірність того, що n вузлів будуть розподілені в одному шардінгу задовольняє біноміальний розподіл [44] (це розподіл дискретної випадкової величини, який описує кількість успішних результатів у серії незалежних експериментів, кожен з яких має два можливих виходи: успіх або невдача), як показано в формулі 2.5. В формулі наведено позначення які означають:

N - загальна кількість вузлів у системі блокчейн,

n - кількість вузлів, розподілених у шардінгу,

S – кількість шардингів

$$P(n) = C_N^n \left(\frac{1}{S}\right)^n \left(\frac{S-1}{S}\right)^{N-n} \quad (2.5)$$

Для кращої візуалізації створимо графік. Для початку, задаємо загальну кількість вузлів у системі блокчейн $N = 150$, а кількість фрагментів $Sas = 10$. Потім підставимо ці дані в формулу ймовірності розподілу, і отримуємо криву ймовірності $P(n)$ щодо n , як показано на рисунку 2.8. Результати показують, що $P(n)$ має максимальну ймовірність при $n = N/S$. Це означає, що коли вузли випадковим чином розподілені на шарди, кількість вузлів у кожному шарді є сталою.

Отже, графік залежності $P(n)$ від n (див. рис. 2.5) підтвердив, що $n=N/S$ є найбільш імовірним розподілом для вузлів у шардах.

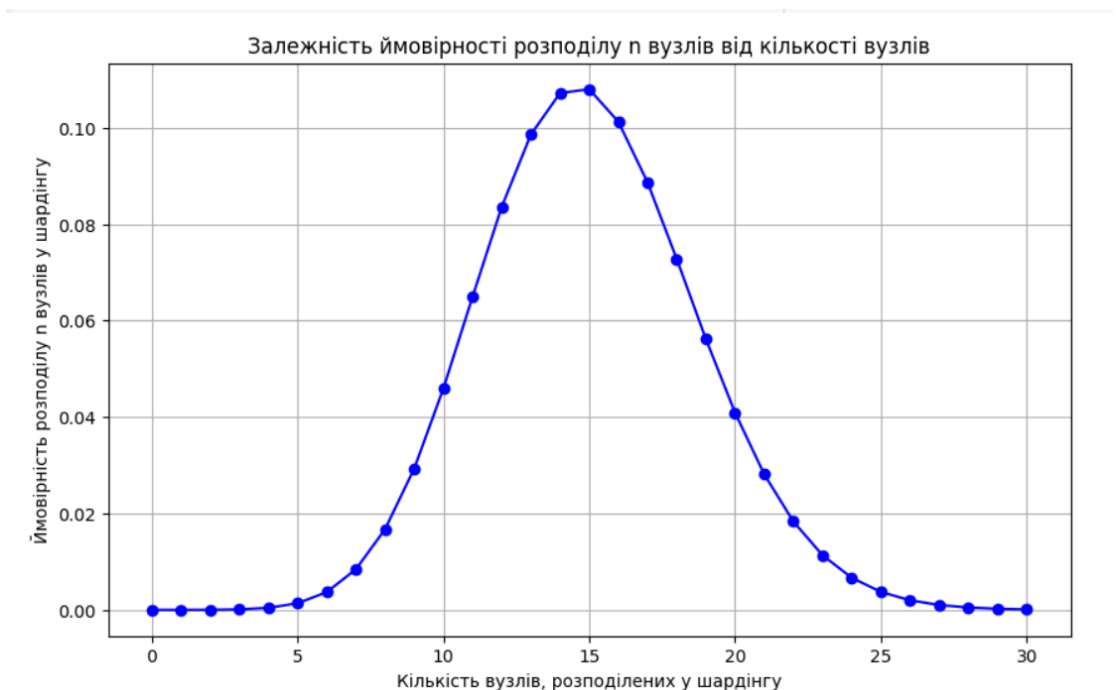


Рис. 2.5 Графік залежності ймовірності розподілу n вузлів у шардингу та кількості вузлів в шардингу

Для зловмисних вузлів, у шардингу, який може вмістити n вузлів, ймовірність випадкового розподілу m шкідливих вузлів можна розглядати як задачу вибірки продуктів, тобто є M (загальна кількість шкідливих вузлів у системі блокчейн) серед N (загальна кількість вузлів у системі блокчейн), і ймовірність вилучення n (кількість вузлів, розподілених у шардингу) продуктів з m (кількість шкідливих вузлів, розподілених при шардингу) є гіпергеометричним розподілом [45] (це статистичний розподіл дискретних випадкових величин, який використовується для визначення ймовірності

певної кількості успішних результатів у вибірці без повернення з кінцевої сукупності).

$$P_1(m) = \frac{C_M^m C_{N-M}^{n-m}}{C_N^n} \quad (2.6)$$

Далі можемо вивести ймовірність успішної транзакції в шардінгу, який може вмістити n вузлів. Згідно з консенсусом PBFT, максимум можна дозволити $1/3$ зловмисних вузлів у шарді, тому ймовірність успішної транзакції зростає від $m = 0$ до $1/3$ від n , як показано на рис 2.10.

$$P_s(m) = \sum_{m=0}^{\frac{n}{3}} P_1(m) = \sum_{m=0}^{\frac{n}{3}} \frac{C_M^m C_{N-M}^{n-m}}{C_N^n} \quad (2.7)$$

На рисунку 2.6, $P_s(m)$ зменшується зі збільшенням частки зловмисних вузлів, що вказує на те, що зловмисні вузли впливатимуть на досягнення консенсусу транзакцій PBFT. Згідно з правилом спадної кривої, $P_s(m)$ повільно зменшується поблизу $R = 0$ і $R = 0,7$, і дуже швидко падає між $R = 0,2$ і $R = 0,5$, де R – це частка шкідливих вузлів у загальній кількості вузлів.

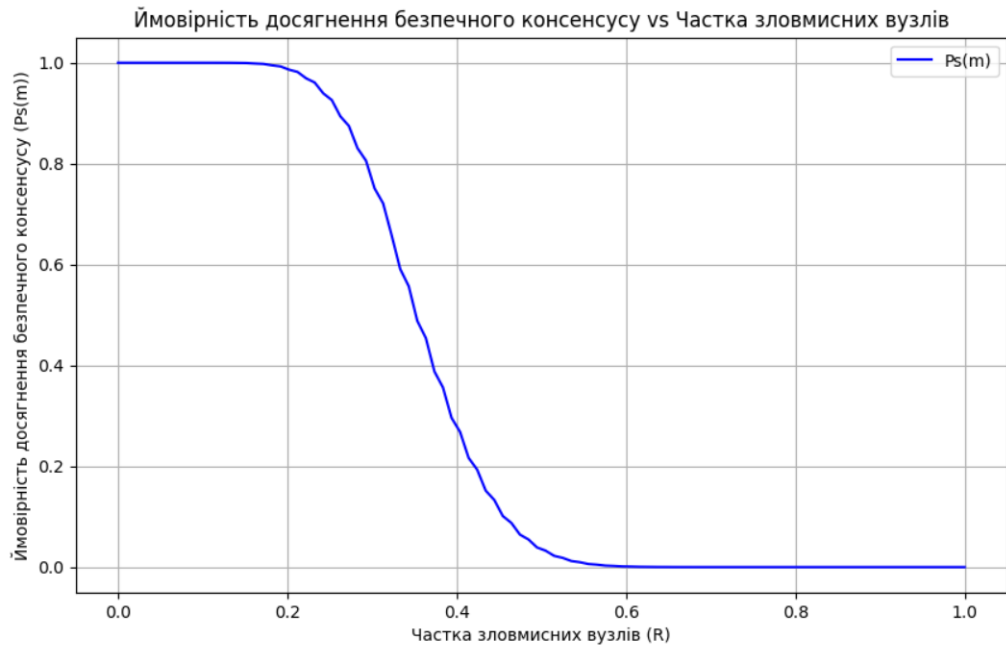


Рис. 2.6 Залежність ймовірності досягнення безпечного консенсусу від частки зловмисних вузлів

Крім того, на практиці часто потрібно спочатку визначити кількість розподілених вузлів у шардінгу. Тому, коли визначено n вузлів у шардінгу, ймовірність наявності m зловмисних вузлів у цьому шардінгу є добутком біноміального та гіпергеометричного розподілу, оскільки ці два рівняння є незалежними.

$$P(m|n) = P(n)P_1(m) = C_N^n \left(\frac{1}{S}\right)^n \left(\frac{S-1}{S}\right)^{N-n} \frac{C_M^m C_{N-M}^{n-m}}{C_N^n} \quad (2.8)$$

На рисунку 2.7 значення $P(m|n)$ залежить не лише від кількості шкідливих вузлів m у кожному шарді, але й від значення R . Зі збільшенням R пікове значення $P(m|n)$ зміщується вправо, а пікові значення $P(m|n)$ з обох боків трохи більші, ніж пікові значення $P(m|n)$ зсередини. Крім того, пік $P(m|n)$ приблизно відповідає значенню $m = Rn$.

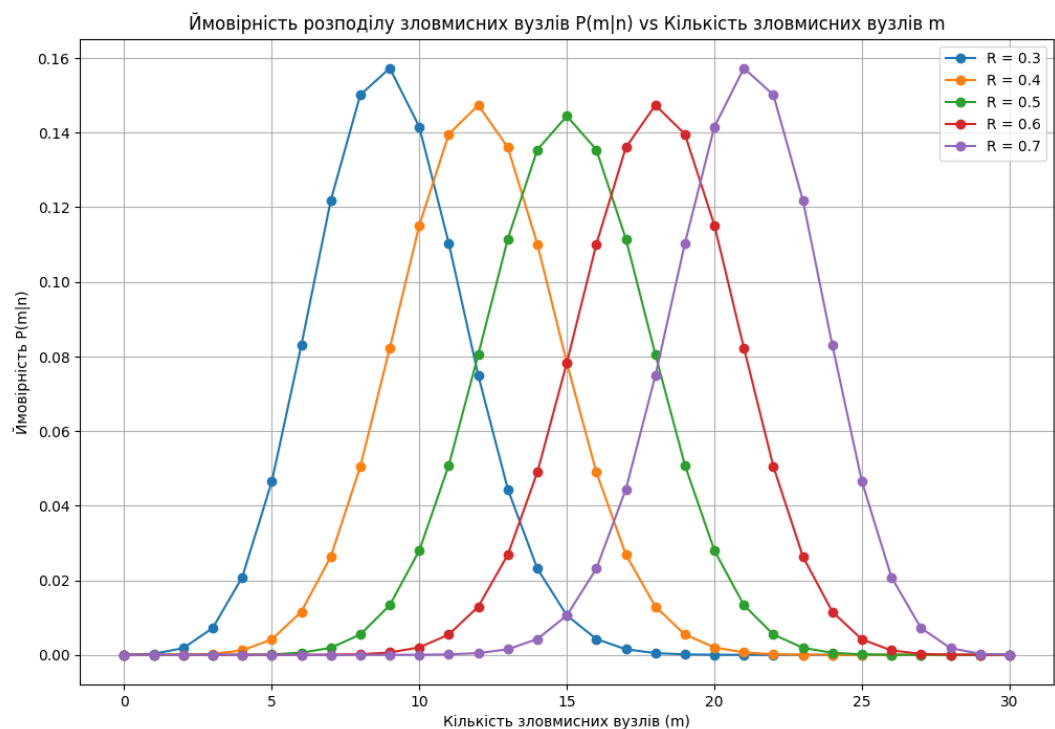


Рис. 2.7 Залежність ймовірності розподілу зловмисних вузлів від кількості зловмисних вузлів

Подібно до ймовірності наявності зловмисних вузлів в шардінгу, ймовірність досягнення консенсусу PBFT транзакції обчислюється формулою.

$$P_s(m|n) = \sum_{m=0}^{\frac{n}{3}} P(m|n) = \sum_{m=0}^{\frac{n}{3}} C_N^n \left(\frac{1}{S}\right)^n \left(\frac{S-1}{S}\right)^{N-n} \frac{C_M^m C_{N-M}^{n-m}}{C_N^n} \quad (2.9)$$

Правило на рисунку 2.15 приблизно таке ж, як на рисунку 2.11, за винятком того, що ймовірність успішних консенсусних транзакцій PBFT набагато нижча, тому що ми визначили n вузлів у шардингу, і ймовірність того що ці n вузлів будуть віднесені до одного і того ж шардингу дуже мала, що показано на рисунку 2.8.

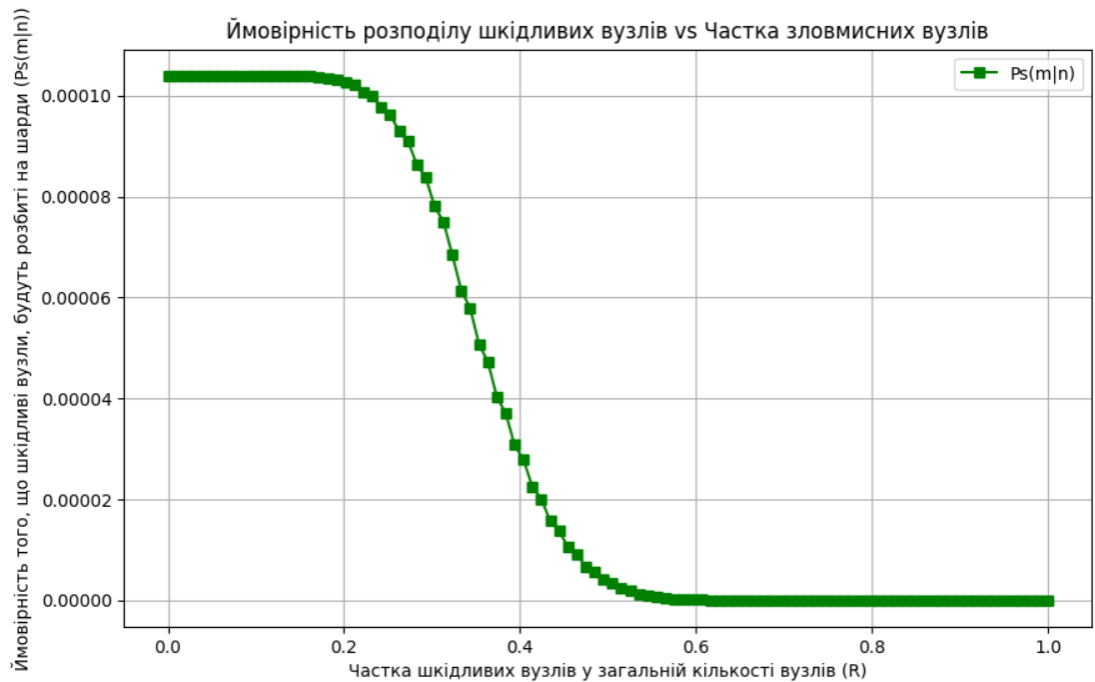


Рис. 2.8 Залежність ймовірності досягнення безпечного консенсусу від частки зловмисних вузлів

Отже, моделювання показало, що кількість зловмисних вузлів впливає на ймовірність консенсусних транзакцій PBFT в цих двох моделях, що є основою для розгортання алгоритму консенсусу PBFT в ситуаціях шардингу.

Висновки до другого розділу

У другому розділі було розглянуто проблему масштабованості у протоколів консенсусу, що базуються на візантійській угоді. Також було проаналізовано алгоритми які вирішили проблему та на основі цих алгоритмів, шардинг було запропоновано як один з методів вирішення

проблеми, який дозволяє розбити мережу на менші частини (шарди), що паралельно обробляють транзакції, підвищуючи продуктивність.

Також порівнювали моделі cross sharding та non-cross sharding, визначили, що Non-cross Sharding Model має деякі переваги такі як: простота, масштабованість, затримка, надійність, вартість впровадження та продуктивність.

Далі за допомогою математичної моделі було розглянуто як кількість зловмисних вузлів впливає на ймовірності відмови комітетів.

РОЗДІЛ 3. МЕТОДИ ВИРІШЕННЯ ПРОБЛЕМИ МАСШТАБОВАНOSTI

3.1 Вирішення проблеми за допомогою шардінгу

Завдання створення шардів у блокчейн-системі складніше, ніж у розподіленій базі даних. По-перше, вузли повинні бути випадковим чином розподілені на комітети без упереджень. По-друге, розмір кожного комітету повинен бути ретельно підібраний для досягнення балансу між ефективністю та безпекою. Нарешті, призначення комітетів має відбуватися з певною затримкою, щоб запобігти адаптивним атакам, які фокусуються на компрометації більшості вузлів у комітеті.

Щоб безпечно створити шард, потрібно незміщене випадкове число rnd для розподілу вузлів на комітети, використовуючи rnd для обчислення випадкової π перестановки всіх можливих перестановок $[1:N]$, згенерованих rnd . Потім π розбивається на групи приблизно однакового розміру, кожна з яких представляє всі вузли комітету.

Далі TEE [46] використовується для ефективного пошуку rnd у розподіленому візантійському просторі шляхом додавання анклавів, які повертають випадкове значення – *RandomnesBeacon* - до кожного вузла. Передбачається, що синхронні мережі мають постійну затримку Δ під час процедури генерації випадкових значень.

Приклад реалізації шардингової блокчейн-системи, яка працює в епохах. Кожна нова епоха відповідає призначенню нового вузла в комітет. На початку кожної епохи кожен вузол викликає значення *RandomnesBeacon* анклавів у номері епохи. Анклав створює 2 випадкові значення q та rnd , за допомогою двох незалежних функцій $sgx_{read_{rand}}$. Enclave повертає підписаний сертифікат, що містить (e, rnd) , але тільки при значенні $q = 0$. Цей сертифікат надсилається всім вузлам мережі. Після Δ таймауту вузли фіксуються на найнижчому rnd , отриманому в епоху e , і це значення використовується для обчислення комітету, до якого призначено вузол.

Для того, щоб зловмисник не міг вибірково відкидати вихідні дані анклаву, потрібно змінити остаточне випадкове значення rnd , щоб анклав

викликався лише один раз за епоху. Якщо вузол не отримує повідомлення від анклаву (e, rnd) після затримки Δ , яка виникає при його отриманні, інкрементальний процес повторюється.

Ймовірність повторення цього процесу: $P_{repeat} = (1 - 2^{-l})^N$, де l - кількість бітів q . Це значення можна змінювати для досягнення бажаного компромісу між P_{repeat} та надлишковістю повідомлень між вузлами: $O(2^{-l}N^2)$. Наприклад, коли $l = \log(z)$ для деякої константи z , $P_{repeat} \approx e^{-1}$, а кількість обмінів повідомленнями дорівнює $O(N)$.

Оскільки призначення комітету визначається випадковою перестановкою $\pi \in [1:N]$ серед усіх можливих, заданих значенням rnd , призначення комітету можна розглядати як випадкову вибірку без заміни. Отже, гіпергеометричний розподіл можна використовувати для обчислення ймовірності відмови комітету (коли комітет містить більше ніж f візантійських вузлів). Коли загальний розмір мережі становить N вузлів, з яких не більше $F = sN$ вузлів є візантійськими, нехай X - змінна ймовірності, що представляє кількість візантійських вузлів, призначених комітету розміром n . Тоді ймовірність відмови комітету (ймовірність того, що безпека буде скомпрометована) становить:

$$Pr [X \geq f] = \sum_{x=f}^n \frac{\binom{F}{x} \binom{N-F}{n-x}}{\binom{N}{n}} \quad (3.1)$$

Ймовірність відмови комітету може бути зведена до незначної, якщо підібрати розмір комітету відповідно до рівняння, яке наведено вище. $f \leq \frac{n-1}{2}$ (у випадку PBFT), при потужності протидії 25% кожен комітет повинен містити більше 600 вузлів, що мінімізує ймовірність відмови комітету; кожен комітет може витримати до $f = \frac{n-1}{2}$ відмов, тому групи можуть бути значно меншими можна зробити меншими: $n = 80$ для $Pr [X \geq \frac{n-1}{2}] \leq 2^{-20}$. Є дві причини, чому менші комітети працюють швидше: по-перше, зменшуючи надлишковий обмін повідомленнями між вузлами, група може досягти вищої

пропускної здатності. Друга причина полягає в тому, що решта комітетів у мережі можуть досягти більшої пропускної здатності завдяки зменшенню робочого навантаження.

Програма складається з трьох основних функцій, кожна з яких виконує специфічні завдання, необхідні для моделювання та оцінки ефективності шардінгової системи у присутності візантійських вузлів.

Функція `generate_random_shard_assignment(N)` призначена для генерації випадкової перестановки вузлів, що є першочерговим кроком у процесі формування комітетів. Випадковий розподіл вузлів забезпечує рівномірне й непередбачуване призначення вузлів до комітетів, що є критично важливим для забезпечення безпеки та надійності системи.

Використовуючи клас `SystemRandom` з модуля `random`, функція генерує випадкове ціле число (`rnd`). `SystemRandom` використовує системні джерела випадковості, що забезпечує більш високий рівень ентропії у порівнянні з стандартним генератором випадкових чисел. Функція встановлює насіння генератора випадкових чисел за допомогою методу `random.seed(rnd)`. Це забезпечує відтворюваність випадкових перестановок при повторному запуску симуляції з тим самим насінням. Генерується список `nodes`, що містить всі вузли від 1 до N (включно). Використовуючи метод `random.shuffle(nodes)`, список вузлів перемішується у випадковому порядку. Функція повертає перемішаний список вузлів, що буде використовуватися для подальшого розподілу на комітети.

```
def generate_random_shard_assignment(N):  
    rnd = random.SystemRandom().randint(1, 1_000_000)  
    random.seed(rnd)  
    nodes = list(range(1, N + 1))  
    random.shuffle(nodes)  
    return nodes
```

Функція `divide_into_committees(permutation, shard_size)` відповідає за розподіл перемішаного списку вузлів на комітети певного розміру. Це ключовий етап у процесі шардінгу, оскільки від правильного розподілу залежить баланс навантаження та безпека системи. Використовуючи спискове

зрізання, функція розбиває список `permutation` на підписки (комітети) розміром `shard_size`. Цей вираз перебирає індекси i від 0 до довжини списку `permutation` з кроком `shard_size` і створює підписки, що містять `shard_size` вузлів кожен.

```
def divide_into_committees(permutation, shard_size):
    committees = [permutation[i:i + shard_size] for i in range(0,
len(permutation), shard_size)]
    return committees
```

Функція `calculate_success_probability(N, n, F)` обчислює ймовірність того, що у комітеті з n вузлів буде більше чесних вузлів ніж візантійських (F), що дозволить успішно досягти консенсусу. Це важливий показник для оцінки безпеки та надійності шардінгової системи у присутності зловмисних вузлів. Використовується функція `comb(a, b)` з модуля `math` для обчислення кількості комбінацій, що дозволяє визначити кількість способів вибрати x чесних вузлів з `honest_nodes` та $n - x$ візантійських вузлів з F . Для кожного можливого числа чесних вузлів x (від $n // 2 + 1$ до n) обчислюється ймовірність того, що саме x чесних вузлів та $n - x$ візантійських вузлів потраплять у комітет. Ці ймовірності сумуються для отримання загальної ймовірності успішного консенсусу.

```
def calculate_success_probability(N, n, F):
    honest_nodes = N - F
    probability = sum(
        comb(honest_nodes, x) * comb(F, n - x) / comb(N, n)
        for x in range(n // 2 + 1, n + 1)
    )
    return probability
```

На рис. 3.1 програма дозволяє користувачу ввести параметри системи та одразу отримати оцінку її безпеки через ймовірність успішного консенсусу. Це корисно для тестування різних конфігурацій системи та визначення оптимальних параметрів для забезпечення високого рівня надійності та ефективності.

```

Введіть загальну кількість вузлів (N): 100
Введіть розмір комітету (shard_size): 10
Введіть загальну кількість візантійських вузлів (F): 27

Сформовані комітети:
Комітет 1: [45, 78, 29, 10, 95, 57, 18, 5, 100, 30]
Комітет 2: [84, 86, 82, 20, 85, 93, 27, 34, 49, 81]
Комітет 3: [75, 91, 89, 48, 1, 13, 8, 44, 11, 15]
Комітет 4: [55, 79, 74, 21, 51, 94, 67, 14, 52, 7]
Комітет 5: [69, 38, 58, 73, 62, 31, 4, 60, 25, 41]
Комітет 6: [68, 33, 43, 36, 98, 6, 46, 76, 50, 3]
Комітет 7: [9, 83, 63, 90, 23, 16, 88, 35, 92, 37]
Комітет 8: [65, 77, 66, 64, 32, 17, 72, 53, 47, 22]
Комітет 9: [56, 12, 87, 19, 24, 2, 39, 28, 97, 61]
Комітет 10: [59, 96, 71, 99, 40, 42, 26, 80, 70, 54]

Ймовірність успішного консенсусу: 0.907789

```

Рис. 3.1 Моделювання процесу розподілу вузлів у шардінговій системі

Використання симуляційних моделей, таких як ця програма, є важливим інструментом для дослідження та оптимізації розподілених систем. Вони дозволяють передбачати поведінку системи під різними умовами, аналізувати потенційні вразливості та приймати обґрунтовані рішення щодо архітектури та параметрів шардінгу. Завдяки цьому, можна забезпечити високу продуктивність, масштабованість та надійність сучасних інформаційних систем, що є особливо актуальним у світі, де обсяги даних постійно зростають, а вимоги до швидкості та ефективності обробки транзакцій стають дедалі вищими.

Адаптивні атаки можуть скомпрометувати групи або чесні вузли, які не вийшли з ладу, але компрометація таких вузлів є дуже трудомісткою. Отже, справедливим є наступне твердження: регулярне перепризначення або реконфігурація груп, які перерозподіляють вузли між комітетами, є достатньою умовою для захисту від цього типу атак. Реконфігурація груп відбувається кожну епоху e . Після закінчення епохи $e - 1$, вузол отримує випадково згенероване значення rnd . Вузол обчислює новий розподіл комітетів на e епоху відповідно до значення rnd . Вузол, який перепризначається до комітету, називається перехідним вузлом. Період,

протягом якого перехідна вершина призначається до нового комітету, називається періодом зміни епохи.

Протягом періоду зміни епохи перехідна вершина спочатку припиняє обробляти запити від старого комітету, а потім починає отримувати стан нового комітету від поточних членів кожного комітету. Після завершення отримання стану, вузол формально приєднується до нового комітету і починає обробляти транзакції. Під час зміни епохи перехідний вузол не бере участі в протоколі консенсусу ні старого, ні нового комітету. З цієї причини реконфігурації, коли всі вузли мігрують відразу, є небажаними, оскільки це призводить до непрацездатності системи в період зміни епохи для всіх вузлів.

Оптимальним є підхід, коли всі вузли мігрують до інших комітетів колективно. Зокрема, в кожному комітеті кількість вузлів, що переходять до нового комітету за епоху e , має бути не більше B . Порядок переміщення вершин визначається випадковою величиною rnd .

3.2 Розподіл вузлів в шардінгу за допомогою репутації

Репутація також може бути використана для покращення розповсюдження вузлів на шарди. Переваги використання репутації в тому, що кожна група матиме загальний рейтинг репутації, тобто пропорції активних і неактивних вузлів, чесних та зловмисних валідаторів. Позитивна репутація означає те, що вузол коректно виконує свої обов'язки, наприклад такі як: швидке підтвердження транзакцій, участь у консенсусі та точність даних. В свою чергу негативна репутація знижує рейтинг вузла за такі дії як: бездіяльність, пропуск голосування, шахрайські дії [47].

Також ймовірність того, що валідатор буде обраний лідером групи, залежить від його репутації. В теорії однакові пропорції активних і неактивних вузлів гарантують високу швидкість транзакцій у кожній групі, тому сукупні показники репутації повинні перешкоджати зловмисній поведінці вузлів.

Кожен вузол оцінюється та ранжується відповідно до спільної поведінки в початковому стані. Вузли мережі blockchain поділяються на консенсусні

вузли, вузли-кандидати та альтернативні вузли. Вузли, які ранжуються, групуються за формулою $2f + 1, f/2$. Вузли консенсусу представляють всі вузли і завершують процес консенсусу в системі. Вузли-кандидати не беруть участі в процесі консенсусу системи, а зберігають результати консенсусу. Вузли консенсусу мають пріоритет, оскільки вони є основними компонентами. Вузли-кандидати та альтернативні вузли лише синхронізують оновлення інформації, перевірені вузлами консенсусу, з обліковою книгою блокчейну, яку вони ведуть. Вони не беруть участі в консенсусі. На практиці системні вимоги регулюють співвідношення між кандидатами та альтернативними вузлами, але не менше $2f + 1$, де f є максимальна кількість толерантних вузлів.

Нові вузли не можуть бути консенсусними мережами, доки вони не будуть авторизовані та оцінені консорціумом. Після цього новим мережам буде надано відповідний рівень. Консенсусні вузли повинні мати оцінку більшу за 9. Кандидати мають вузли з рейтингом від 8 до 9, а вузли з рейтингом менше або дорівнює 8 є альтернативними вузлами. У вузлів є три різні стани, і вузли в різних станах можуть взаємозамінні, якщо дотримуються наступних правил:

До кожного консенсусу, досягнутого вузлом, додається 0,02 до балів вузлів з більш ніж 9 балами, 0,1 до балів вузлів з 8 до 9 балів і 0,2 до балів вузлів з менше 8 балів. Консенсусні вершини мають максимальну оцінку 10 балів, і оцінка не збільшується після отримання цієї оцінки. S_p це інтеграл до консенсусу, а S_q інтеграл після консенсусу.

Якщо альтернативна вершина помиляється, її рейтинг знижується на 0,5. Якщо консенсусна вершина помиляється, помилкова вершина перетворюється на вершину кандидата, а кількість балів вершини безпосередньо збільшується на 8. Консенсусна вершина перетворюється на кандидата з найвищою кількістю балів; її бали змінюються на 9.

$$S_p = \begin{cases} S_p + 0.02, & S_p > 9 \\ S_p + 0.1, & 8 < S_p \leq 9 \\ S_p + 0.2, & S_p \leq 8 \end{cases} \quad (3.2)$$

На рисунку 3.2 показано механізм балів який використовується для сортування вузлів; вузли, які є дуже надійними, мають $2f + 1$, розташовані в порядку $\{0, \dots, 2f\}$ від більшого до меншого відповідно до номера консенсусного вузла. Головний вузол з високим ступенем довіри визначається першим N, а бали служать еталоном для забезпечення стабільності головного вузла.

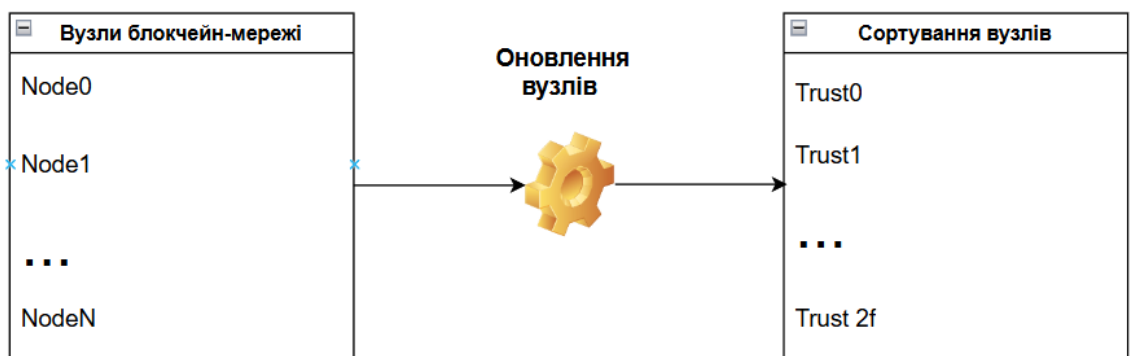


Рис. 3.2 Механізм балів який використовується для сортування вузлів

Проте також є деякі недоліки використання репутації, вона викликає додаткову складність тим, що вимагає зберігання та оновлень реплік для кожного вузла, вузли намагатимуться обійти систему та отримати репутацію неправомірним шляхом, потрібно врахувати, що швидко підвищити репутацію неможливо або навпаки навмисно знижувати репутацію інших вузлів. Для того, щоб запобігти атаці можна запропонувати декілька методів, таких як використання алгоритмів, що враховують історію поведінки вузлів для виявлення аномалій, використання мультисигнатур або інших методів для підтвердження достовірності відгуків, обмеження впливу одного вузла на репутацію інших вузлів, щоб зменшити ризик атак.

3.3 Залежність ймовірності відмови при зміні даних

На рис. 3.3 показано, що, ймовірність відмови комітету значно зменшується зі збільшенням кількості вузлів у мережі. Це свідчить про те, що наявність більшої кількості вузлів підвищує стійкість системи до відмов. На початку ймовірність відмови комітету значно знижується при кількості вузлів від 10 до 30. Це може бути пов'язано з тим, що ймовірність того, що буде достатньо чесних вузлів для досягнення консенсусу, збільшується зі збільшенням кількості вузлів.

Також графік стає більш плавним, коли кількість вузлів збільшується до 40 або більше. Що означає, що при збільшенні кількості вузлів продовжує зменшуватися ймовірність відмови, але вже не так стрімко, як на початку, а при кількості вузлів більше 80, ймовірність відмови комітету наближається до нуля, що означає високу стійкість системи. Отже, можна припустити, що при збільшенні вузлів, ймовірність відмови комітету наближається до мінімуму, що означає, що додавання нових вузлів неефективне.

Також, графік може служити основою для аналізу вартості та вигоди від збільшення кількості вузлів у системі, враховуючи додаткові витрати, пов'язані з збільшенням кількості вузлів (наприклад, витрати на апаратне забезпечення, енергоспоживання тощо).



Рис. 3.3 Залежність ймовірності відмови від кількості вузлів

Згідно з графіком, який показаний на рис. 3.4, ймовірність відмови комітету значно зростає зі збільшенням кількості візантійських вузлів у мережі. Ймовірність відмови залишається дуже низькою, майже нульовою при низькій частці візантійських вузлів.

Коли частка візантійських вузлів перевищує 0,3, ймовірність відмови значно збільшується. Це означає, що система стає більш чутливою до відмов, якщо кількість візантійських вузлів перевищує тридцять відсотків. Для створення надійної шардінгової системи важливо враховувати, що частка шкідливих вузлів повинна бути значно нижчою за тридцять відсотків. Це можна досягти за допомогою різних засобів захисту, наприклад, перевірки автентичності вузлів, додаткове шифрування даних і постійний моніторинг мережі.

Чим менша ця частка, тим надійніше працює система шардингу. Якщо значення буде близьким до 50%, система може виявитися нестабільною або повністю відмовити, це пов'язано з тим, що консенсус стає неможливим, якщо більшість вузлів некоректно поведуться.

Також, на практиці результати цього аналізу можуть вказувати на те, що для того, щоб забезпечити максимальну можливу стійкість системи до візантійських атак, необхідно збалансувати кількість шардінгів і загальної кількості вузлів.

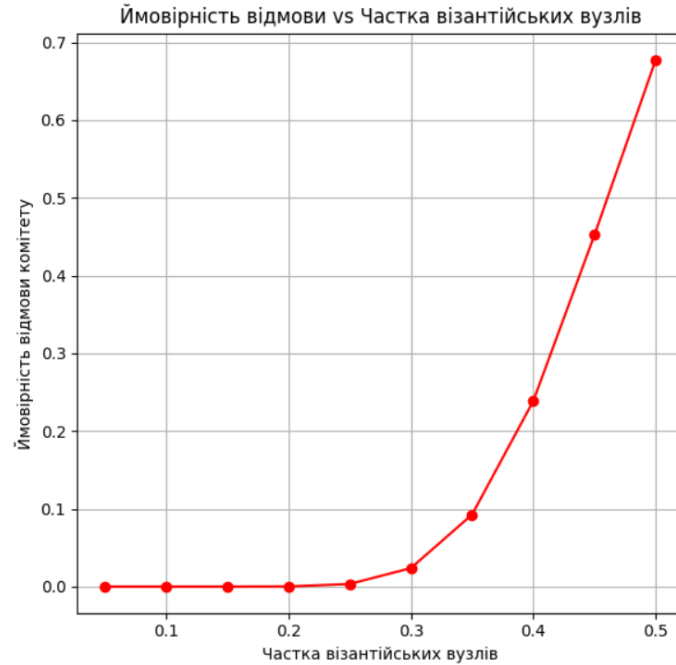


Рис. 3.4 Залежність ймовірності відмови від частки вузлів

На рис. 3.5 представлено графік, що показує, як змінюється ймовірність відмови комітету залежно від його розміру.

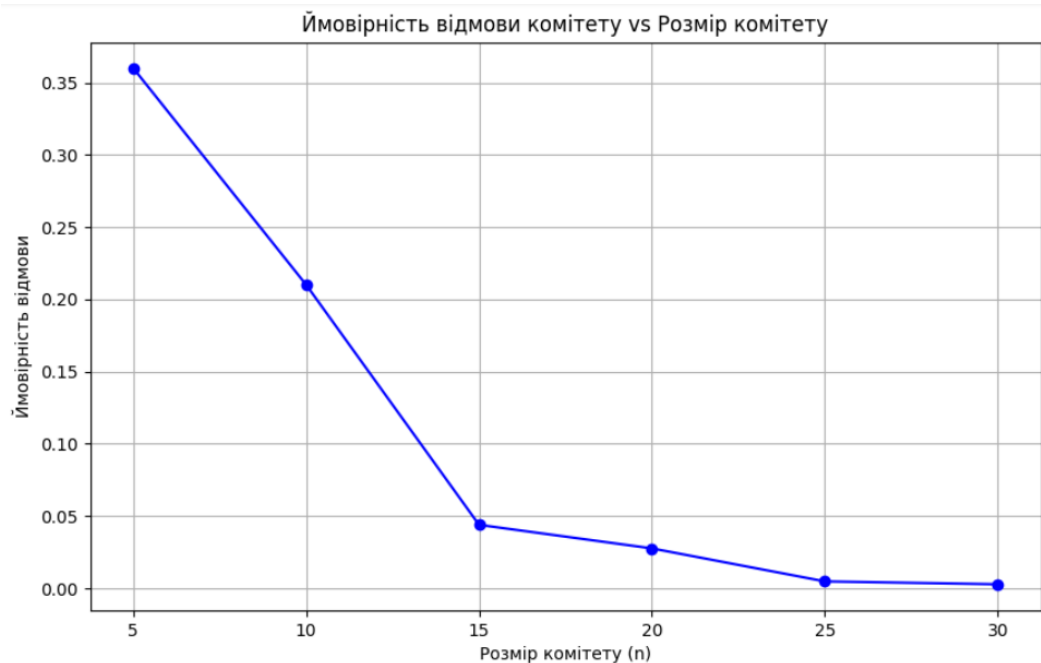


Рис. 3.5 Залежність ймовірності відмови від частки вузлів

При збільшенні розміру комітету ймовірність відмови комітету зменшується. На початку, коли розмір комітету складає 5, ймовірність відмови становить приблизно 0.35 (35%). При розмірі комітету 30, ймовірність відмови комітету наближається до 0. Найбільше зниження ймовірності відмови спостерігається на ранніх етапах збільшення розміру комітету. Зокрема, між розмірами комітету 5 і 10 видно найбільший спад у ймовірності відмови. Зі збільшенням розміру комітету до 30, ймовірність відмови продовжує зменшуватися, але вже не так різко. Це свідчить про те, що більший розмір комітету забезпечує меншу ймовірність відмови, але кожне наступне збільшення розміру має все менший вплив.

Отже, збільшення розміру комітету суттєво зменшує ймовірність відмови, особливо на ранніх етапах. Найбільший вплив на зменшення ймовірності відмови досягається при збільшенні розміру комітету з 5 до 15 вузлів. Також, після досягнення певного розміру комітету (близько 20-30 вузлів) подальше збільшення незначно впливає на ймовірність відмови, що може бути важливим при оптимізації використання ресурсів.

3.4 Аналіз безпеки шардінгового блокчейну

Нехай k кількість груп, які представляють частину глобального стану блокчейну. Реконфігурація групи суттєво змінює набір вузлів, які обробляють запити для кожної k групи. Розглянемо групу sh , яка представляє комітет, що обробляє шард sh в епоху $e - 1$ часом C_{e-1} та епоху e часом C_e . Вузли V змінюють комітети за час C_{e-1} і існує $\frac{n}{k}$ вузлів, які протягом цього часу очікують залишитися з часом C_e то $\frac{n(k-1)}{k*B}$ проміжних комітетів обробляє шард sh під час зміни епохи.

Міграція вузла V не порушує безпеку групи sh , так як кількість візантійських вузлів не збільшується. З іншого боку, якщо новий вузол V мігрує до нового комітету, кількість візантійських вузлів у тимчасовому комітеті може збільшити розмір комітету до рівня, коли ймовірність відмови комітету є високою, а ймовірність відмови є основним фактором, що визначає

рівень безпеки комітету. Оскільки вузли переходу вибираються випадковим чином відповідно до значення rnd , ймовірність відмови проміжного комітету відповідає рівнянню ймовірності відмови комітету. В інтервалі часу від C_{e-1} до C_e очікується існування $\frac{n(k-1)}{k*B}$ проміжних комітетів. Використовуючи булеву нерівність, оцінюємо ймовірність того, що шард має відповідний рівень безпеки протягом перехідної епохи:

$$Pr_{faulty} \leq \sum_{i=1}^{\frac{n(k-1)}{k*B}} \sum_{x=f}^n \frac{\binom{F}{x} \binom{N-F}{n-x}}{\binom{N}{n}} \quad (3.3)$$

Наприклад, $n = 80$, $f = \frac{n-1}{2}$, $k = 10$ груп і вузлів $B = \log(n) = 6$, $Pr_{faulty} \approx 10^{-5}$. Виходячи з вищенаведеного рівняння, можна мати B перехідні вузли, щоб знайти компроміс між життєздатністю та безпекою системи під час переходу епох.

Під час перехідного періоду кожен комітет має B вузлів, які не обробляють запити. Якщо $B > f$, то решта вузлів не можуть сформувати кворум і шард не буде функціонувати. Тому, чим вище значення B , тим більший ризик втрати життєздатності під час перехідного періоду епохи.

Висновки до третього розділу

У третьому розділі розглядаються різні підходи до реалізації шардингу та безпеки системи. Для створення шардів використовувалася випадкова перестановка π , створена значенням rnd , що дозволяє вузлам випадково розподілятися на комітети.

Розглянуто репутацію як метод розподілу вузлів на шарди.

Також оцінюється ймовірність відмови комітету залежно від кількості вузлів і частки візантійських вузлів. Встановлено, що система стає нестабільною при значній частці візантійських вузлів (більше 30%).

Крім того, було проведено симуляцію, щоб показати, як ймовірність відмови залежить від кількості вузлів у комітеті. Щоб уникнути зниження

ефективності системи, міграція вузлів у перехідний період потребує ретельного контролю.

ВИСНОВКИ

У першому розділі було розглянуто що таке алгоритми консенсу, здійснено ґрунтовний аналіз існуючих алгоритмів консенсусу, що використовуються в блокчейн-системах. Розписано проблему візантійських генералів та розглянуто алгоритм її вирішення. Також було розглянуто алгоритм PBFT в основі якого лежить візантійська відмовостійкість. У результаті аналізу було встановлено, що PBFT є ефективним рішенням для блокчейн-систем з обмеженою кількістю учасників завдяки високій швидкості консенсусу та відмовостійкості. Проте він не масштабується при зростанні кількості вузлів.

У другому розділі було розглянуто проблему масштабованості у протоколів консенсусу, що базуються на візантійській угоді. Також було проаналізовано алгоритми які вирішили проблему та на основі цих алгоритмів, було запропоновано шардинг як один з методів вирішення проблеми, який дозволяє розбити мережу на менші частини (шарди), що паралельно обробляють транзакції, підвищуючи продуктивність.

Також порівнювали моделі cross sharding та non-cross sharding, визначили, що Non-cross Sharding Model краще підходить для підвищення продуктивності за умови мінімізації взаємодії між групами.

Далі було розписано математичну модель ймовірності відмови комітетів.

У третьому розділі розглядаються різні підходи до реалізації шардингу та безпеки системи. Для створення шардів використовувалася випадкова перестановка π , створена значенням rnd , що дозволяє вузлам випадково розподілятися на комітети.

Також оцінюється ймовірність відмови комітету залежно від кількості вузлів і частки візантійських вузлів. Встановлено, що система стає нестабільною при значній частці візантійських вузлів (більше 30%).

Крім того, було проведено симуляцію, щоб показати, як ймовірність відмови залежить від кількості вузлів у комітеті. Щоб уникнути зниження

ефективності системи, міграція вузлів у перехідний період потребує ретельного контролю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гальчин А. В. Роль блокчейн-технології у сучасному світі. Збірник матеріалів XI Всеукраїнської науково-практичної конференції молодих вчених «Наукова молодь-2023» (Київ, 21 листопада 2023 р.). / упоряд.: А. Яцишин. К.: ЦП «КОМПРИНТ», 2023. – С. 40. URL: http://www.uintei.kiev.ua/sites/default/files/zbirnik_konferenciyi_naukova_molod_2023.pdf.
2. Гальчин А. В., Чижмотря О. Г. Аналіз існуючих алгоритмів в блокчейні. Технічна творчість: Збірник наукових праць./ Укл.: Скиба М. Є., Поліщук О. С., Романець Т. П. – Хмельницький: ХНУ, 2023. – № 7 – С. 138-139. URL: https://maees.khmnu.edu.ua/wp-content/uploads/2024/02/Технічна-творчість_Збірник_№7_2023.pdf.
3. Гальчин А. В., Чижмотря О. Г. Приватність та безпека в блокчейні. Технічна творчість: Збірник наукових праць./ Укл.: Скиба М. Є., Поліщук О. С., Романець Т. П. – Хмельницький: ХНУ, 2023. – № 7 – С. 141-142. URL: https://maees.khmnu.edu.ua/wp-content/uploads/2024/02/Технічна-творчість_Збірник_№7_2023.pdf.
4. Academy В. Що таке алгоритм консенсусу на блокчейні? | Binance Academy. *Binance Academy*. URL: <https://academy.binance.com/uk/articles/what-is-a-blockchain-consensus-algorithm>.
5. A secure and highly efficient blockchain PBFT consensus algorithm for microgrid power trading - Scientific Reports. *Nature*. URL: <https://www.nature.com/articles/s41598-024-58505-w>.
6. Консенсус, що це таке в блокчейні, які бувають види та для чого він. *web3.0 – нове покоління інтернету вже настало!*. URL: <https://www.web3.org.ua/shcho-take-konsensus-v-blokchejni>.
7. Proof-of-Stake (PoS) - як працює алгоритм? | EXBASE. *EXBASE - create crypto wallet online | Cryptocurrency Wallet*. URL: <https://exbase.io/uk/wiki/dokaz-chastki>.

8. Пояснення: Що таке підтвердження роботи (PoW) у блокчейні?. *Bybit*. URL: <https://learn.bybit.com/uk/blockchain/what-is-proof-of-work-in-blockchain/>.
9. Майбутнє технології блокчейн: 10 ключових розділів про нові тенденції, масштабовані рішення та нормативні зміни. *Julien Florkin*. URL: <https://julienflorkin.com/uk/technology/blockchain/blockchain-technology/>.
10. Блокчейн: основні принципи та сфери застосування. *Вгору | Новини*. URL: <https://vgoru.org/pererva-na-kavu/blokchejn-osnovni-principi-ta-sferi-zastosuvannya>.
11. Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982), 382–401. <https://doi.org/10.1145/357172.357176>.
12. Що таке проблема візантійських генералів?. *Plisio*. URL: <https://plisio.net/uk/blog/what-is-the-byzantine-generals-problem>.
13. Academy B. Пояснення візантійської проблеми | Binance Academy. *Binance Academy*. URL: <https://academy.binance.com/uk/articles/byzantine-fault-tolerance-explained>.
14. Махым Zosym. Проблема візантійських генералів (The Byzantine Generals Problem). *Maxym Zosym*. URL: <https://www.maxzosim.com/the-byzantine-generals-problem/>.
15. GeeksforGeeks. Consensus Problem of Distributed Systems - GeeksforGeeks. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/consensus-problem-of-distributed-systems/>.
16. Driscoll, K., Hall, B., Sivencrona, H., Zumsteg, P. (2003). Byzantine Fault Tolerance, from Theory to Reality. In: Anderson, S., Felici, M., Littlewood, B. (eds) *Computer Safety, Reliability, and Security. SAFECOMP 2003. Lecture Notes in Computer Science*, vol 2788. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39878-3_19.

17. Naomiii. Byzantine Fault Tolerance. *Medium*. URL: <https://medium.com/fmfw-io/byzantine-fault-tolerance-dffd691f1cc5>.
18. Як різні протоколи розв'язують завдання візантійських генералів? | GNCrypto | GNCrypto News. *Crypto Market News, Alerts, Today, Live Coin Update* ▷ GNCrypto. URL: <https://www.gncrypto.news/ua/news/the-byzantine-generals-problem-how-do-protocols-solve-it>.
19. Як різні протоколи розв'язують завдання візантійських генералів? | GNCrypto | GNCrypto News. *Crypto Market News, Alerts, Today, Live Coin Update* ▷ GNCrypto. URL: <https://www.gncrypto.news/ua/news/the-byzantine-generals-problem-how-do-protocols-solve-it>.
20. Devs T. C. An introduction to PBFT consensus algorithm. *Medium*. URL: <https://medium.com/tronnetwork/an-introduction-to-pbft-consensus-algorithm-11cbd90aaec>.
21. Gibala S. Implementing practical byzantine fault tolerance - part 1. URL: <https://sgibala.com/02-01-implementing-pbft/>.
22. Як різні протоколи розв'язують завдання візантійських генералів? | GNCrypto | GNCrypto News. *Crypto Market News, Alerts, Today, Live Coin Update* ▷ GNCrypto. URL: <https://www.gncrypto.news/ua/news/the-byzantine-generals-problem-how-do-protocols-solve-it>.
23. Wang D., Wang Z., Lian X. Research on Distributed Energy Consensus Mechanism Based on Blockchain in Virtual Power Plant. *Sensors*. 2022. Vol. 22, no. 5. P. 1783. URL: <https://doi.org/10.3390/s22051783>.
24. Academy B. Що таке solana (SOL)? | binance academy. *Binance Academy*. URL: <https://academy.binance.com/uk/articles/what-is-solana-sol>.
25. Огляд проєкту Solana | CRYPTOLOGY. Школа трейдингу CRYPTOLOGY.KEY | Навчання трейдингу. URL: <https://cryptology.school/blog/ogliad-projektu-solana>
26. Розвиток екосистеми Solana: від «вбивці Ethereum» до інноваційного піонера. *Gate.io*. URL: <https://www.gate.io/uk/learn/articles/-evolution-of-the-solana-ecosystem-from-ethereum-killer-to-innovation-pioneer/4659>.

27. Fan S. Переосмислення масштабованості. *Medium*.
URL: <https://medium.com/@StarknetFan/переосмислення-масштабованості-6b1a1f1f2d03>.
28. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. *arXiv.org*. URL: <https://arxiv.org/abs/1602.06997>.
29. Alqahtani S. Paper review of byzcoin. *Medium*.
URL: <https://salemal.medium.com/paper-review-of-byzcoin-9ce7676d1c32>.
30. PBFT, IBFT and QBFT consensus algorithms explained. *The Web3 Labs Blog*.
URL: <https://blog.web3labs.com/web3development/comparing-byzantine-fault-tolerance-consensus-algorithms>.
31. Що таке шардинг (sharding) в блокчейн мережах | web3.org.ua. *web3.0 – нове покоління інтернету вже настало!*.
URL: <https://www.web3.org.ua/shcho-take-shardynh/>.
32. Academy B. Blockchain layer 1 vs. layer 2 scaling solutions | binance academy. *Binance Academy*.
URL: <https://academy.binance.com/en/articles/blockchain-layer-1-vs-layer-2-scaling-solutions>.
33. Що таке шардинг в блокчейні? Як це працює? – ITstatti.in.ua. *IT статті - інформаційний блог | Заробіток в інтернеті, інвестиції, бізнес, фінанси, реклама та інтернет – ITstatti.in.ua*. URL: <https://itstatti.in.ua/crypto/1075-shcho-take-sharding.html>.
34. Rafalskyi M. OFFENSES IN DECENTRALIZED SYSTEMS. 34. 2022. No. 34. P. 26–36. URL: <https://doi.org/10.26565/2075-1834-2022-34-03>.
35. Academy B. Що таке zilliqa (ZIL)? | binance academy. *Binance Academy*.
URL: <https://academy.binance.com/uk/articles/what-is-zilliqa-zil>.
36. Kraken Learn team. Що таке Zilliqa? (ZIL). *Kraken*.
URL: <https://www.kraken.com/uk-ua/learn/what-is-zilliqa-zil>.
37. Що таке криптовалюта Harmony (ONE) і чому вам слід бути обережним. *Bybit*. URL: <https://learn.bybit.com/uk/altcoins/harmony-crypto/>.

38. Academy B. Що таке шардинг і як він працює? | Binance Academy. *Binance Academy*. URL: <https://academy.binance.com/uk/articles/what-is-sharding-and-how-does-it-work>.
39. Блокчейни рівня 1. Ознайомлення з основною інфраструктурою криптовалют. *Bybit Learn*. URL: <https://learn.bybit.com/uk/blockchain/what-is-a-layer-1-blockchain/>.
40. Посібник для початківців з алгоритму Proof-of-authority (PoA) – ITstatti.in.ua. *IT статті - інформаційний блог | Заробіток в інтернеті, інвестиції, бізнес, фінанси, реклама та інтернет – ITstatti.in.ua*. URL: <https://itstatti.in.ua/crypto/1068-alhorytm-proof-of-authority-poa.html>
41. Що таке гармонія? Усе, що вам потрібно знати про ONE. *Gate Learn*. URL: <https://www.gate.io/uk/learn/articles/what-is-harmony/283>.
42. Sharding for Scalable Blockchain Networks - SN Computer Science. *SpringerLink*. URL: <https://link.springer.com/article/10.1007/s42979-022-01435-z>.
43. Stringhandler. Comparing Sharding Approaches in Blockchain. *Tari*. URL: <https://www.tari.com/updates/2023-10-20-update-120>.
44. Ткаченко Максим Біноміальний розподіл. *BUE*. URL: https://vue.gov.ua/Біноміальний_розподіл.
45. Що таке гіпергеометричний розподіл імовірностей?. *House of Math*. URL: <https://www.houseofmath.com/uk/encyclopedia/statystyka-ta-ymovirnist/ymovirnist-i-kombinatoryka/rozpodily-ymovirnostey/shcho-take-hiperheometrychnyy-rozpodil-ymovirnostey>.
46. Trusted execution environments (TEE) – intel SGX | secret network. *Secret Network Introduction | Secret Network*. URL: <https://docs.scrt.network/secret-network-documentation/introduction/secret-network-techstack/privacy-technology/intel-sgx>.
47. J. Li and Y. Ning, "Blockchain Transaction Sharding Algorithm Based on Account-Weighted Graph," in *IEEE Access*, vol. 12, pp. 24672-24684, 2024, doi: 10.1109/ACCESS.2024.3365510. keywords:

{ Sharding;Blockchains;Delays;Throughput;Scalability;Resource
management;Heuristic algorithms;Sharding;Transaction
databases;Scalability;transaction sharding;sharding granularity;cross-shard
transaction proportion;transaction delay},

Додатки

Симуляція моделей Cross Sharding і Non-Cross Sharding

```
import random
import time

def cross_sharding_simulation(nodes, shards, delay):

    start_time = time.time()

    shard_map = {}
    for i in range(shards):
        shard_map[f"Shard_{i+1}"] = []

    # Розподіл вузлів по шардах
    for node in range(1, nodes + 1):
        shard_id = random.randint(1, shards)
        shard_map[f"Shard_{shard_id}"].append(node)

    # Симуляція транзакцій між шардом
    for i in range(shards):
        for j in range(i + 1, shards):
            time.sleep(delay)

    end_time = time.time()
    return end_time - start_time

def non_cross_sharding_simulation(nodes, shards):

    start_time = time.time()

    shard_map = {}
    nodes_per_shard = nodes // shards

    for i in range(shards):
        start_index = i * nodes_per_shard
        end_index = start_index + nodes_per_shard
        shard_map[f"Shard_{i+1}"] = list(range(start_index + 1, end_index + 1))

    # Симуляція локальних транзакцій
    for shard in shard_map.values():
        pass

    end_time = time.time()
    return end_time - start_time

def compare_sharding_models(nodes, shards, delay):

    cross_time = cross_sharding_simulation(nodes, shards, delay)
```

```
non_cross_time = non_cross_sharding_simulation(nodes, shards)

print(f"Cross Sharding Time: {cross_time:.4f} seconds")
print(f"Non-Cross Sharding Time: {non_cross_time:.4f} seconds")

if __name__ == "__main__":
    # Параметри системи
    total_nodes = 1000000 # Загальна кількість вузлів
    total_shards = 5      # Кількість шардів
    delay_between_shards = 0.1 # Затримка між шардом у секундах

    # Порівняння моделей
    compare_sharding_models(total_nodes, total_shards,
delay_between_shards)
```

Моделювання шардингової системи та оцінка надійності консенсусу

```
import random
from math import comb

def generate_random_shard_assignment(N):

    rnd = random.SystemRandom().randint(1, 1_000_000)
    random.seed(rnd)
    nodes = list(range(1, N + 1))
    random.shuffle(nodes)
    return nodes

def divide_into_committees(permutation, shard_size):
    committees = [permutation[i:i + shard_size] for i in range(0,
len(permutation), shard_size)]
    return committees

def calculate_success_probability(N, n, F):

    honest_nodes = N - F
    probability = sum(
        comb(honest_nodes, x) * comb(F, n - x) / comb(N, n)
        for x in range(n // 2 + 1, n + 1)
    )
    return probability

# Основна програма
if __name__ == "__main__":

    N = int(input("Введіть загальну кількість вузлів (N): "))
    shard_size = int(input("Введіть розмір комітету (shard_size): "))
    F = int(input("Введіть загальну кількість візантійських вузлів (F): "))

    if F > N // 3:
        print("\nЙмовірність відмови надто висока. Шардингова система стає
ненадійною.")
    else:
        permutation = generate_random_shard_assignment(N)

        committees = divide_into_committees(permutation, shard_size)
        print("\nCформовані комітети:")
        for idx, committee in enumerate(committees):
            print(f"Комітет {idx + 1}: {committee}")

        success_probability = calculate_success_probability(N, shard_size,
F)
        print(f"\nЙмовірність успішного консенсусу:
{success_probability:.6f}")
```

Код для побудови ймовірність розподілу n вузлів у шардінгу

```
import matplotlib.pyplot as plt
import numpy as np
from math import comb

# Встановлюємо загальну кількість вузлів та кількість фрагментів
N = 150
S = 10
n_values = np.arange(0, 31, 1) # Значення n від 0 до 30 з кроком 1

# Формула для розрахунку P(n)
P_n = [comb(N, n) * (1 / S)**n * ((S - 1) / S)**(N - n) for n in n_values]

# Побудова графіку
plt.figure(figsize=(10, 6))
plt.plot(n_values, P_n, marker='o', linestyle='-', color='b')
plt.xlabel('Value of n')
plt.ylabel('Value of P(n)')
plt.title('Probability P(n) vs. Value of n')
plt.grid(True)
plt.show()
```

Код для побудови залежності ймовірності досягнення безпечного консенсусу від частки зловмисних вузлів

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb

def Ps(N, M, n):
    """Обчислює ймовірність успішної транзакції."""
    probability = 0
    for m in range(int(n / 3) + 1):
        probability += (comb(M, m) * comb(N - M, n - m)) / comb(N, n)
    return probability

# Параметри
R_values = np.linspace(0, 1, 100)
N = 150 # Загальна кількість вузлів
n = 30 # Розмір комітету
Ps_values = [Ps(N, int(R * N), n) for R in R_values]

# Побудова графіку
plt.figure(figsize=(10, 6))
plt.plot(R_values, Ps_values, label="Ps(m)", color="b")
plt.title("Ймовірність досягнення безпечного консенсусу vs Частка зловмисних вузлів")
plt.xlabel("Частка зловмисних вузлів (R)")
plt.ylabel("Ймовірність досягнення безпечного консенсусу (Ps(m))")
plt.grid(True)
plt.legend()
plt.show()
```

Код для побудови залежності розподілу зловмисних вузлів від кількості зловмисних вузлів

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom

# Параметри
n = 30 # Розмір шардінгу (комітету)
R_values = [0.3, 0.4, 0.5, 0.6, 0.7] # Різні значення частки зловмисних вузлів
m_values = np.arange(0, n + 1) # Кількість зловмисних вузлів від 0 до n

# Функція для обчислення  $P(m|n)$ 
def P_m_given_n(m, n, R):
    p = R # Ймовірність зловмисного вузла
    return binom.pmf(m, n, p)

# Побудова графіку
plt.figure(figsize=(12, 8))

for R in R_values:
    P_m_n = [P_m_given_n(m, n, R) for m in m_values]
    plt.plot(m_values, P_m_n, marker='o', label=f'R = {R}')

plt.title("Ймовірність розподілу зловмисних вузлів  $P(m|n)$  vs Кількість зловмисних вузлів  $m$ ")
plt.xlabel("Кількість зловмисних вузлів ( $m$ )")
plt.ylabel("Ймовірність  $P(m|n)$ ")
plt.legend()
plt.grid(True)
plt.show()
```

Код для побудови залежності ймовірності безпечного консенсусу з m зловмисними вузлів, розподілених у шардингу з n вузлів

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb

# Функція для обчислення ймовірності досягнення безпечного консенсусу
def Ps(N, S, n):
    P_s = []
    for R in np.linspace(0, 1, 100):
        M = int(R * N)
        probability = 0
        for m in range(int(n / 3) + 1):
            P_m_n = (comb(N, n) * (1 / S)**n * ((S - 1) / S)**(N - n) *
                      (comb(M, m) * comb(N - M, n - m)) / comb(N, n))
            probability += P_m_n
        P_s.append(probability)
    return P_s

# Параметри
N = 150 # Загальна кількість вузлів
S = 10  # Кількість шардингу
n = 30  # Розмір комітету

# Обчислення значень ймовірності для різних R
R_values = np.linspace(0, 1, 100)
Ps_values = Ps(N, S, n)

# Побудова графіку
plt.figure(figsize=(10, 6))
plt.plot(R_values, Ps_values, label="Ps(m|n)", color="g", marker='s')
plt.title("Ймовірність досягнення безпечного консенсусу vs Частка зловмисних вузлів")
plt.xlabel("Частка зловмисних вузлів (R)")
plt.ylabel("Ймовірність досягнення безпечного консенсусу (Ps(m|n))")
plt.grid(True)
plt.legend()
plt.show()
```

Код для побудови залежності ймовірності відмови комітету від його розміру

```
import matplotlib.pyplot as plt
import numpy as np
from math import comb

def failure_probability(N, n, f):
    """
    Обчислює ймовірність відмови комітету.
    :param N: Загальна кількість вузлів.
    :param n: Розмір комітету.
    :param f: Максимально допустима кількість візантійських вузлів.
    :return: Ймовірність відмови.
    """
    F = int(0.25 * N) # 25% візантійських вузлів
    probability = 0
    for x in range(f, n + 1):
        probability += (comb(F, x) * comb(N - F, n - x)) / comb(N, n)
    return probability

# Параметри
N = 150 # Загальна кількість вузлів
committee_sizes = range(5, 31, 5) # Розмір комітету від 5 до 30
failure_probs = [failure_probability(N, n, (n - 1) // 2) for n in
committee_sizes]

# Побудова графіка
plt.figure(figsize=(10, 6))
plt.plot(committee_sizes, failure_probs, marker='o', color='b')
plt.title("Ймовірність відмови комітету vs Розмір комітету")
plt.xlabel("Розмір комітету (n)")
plt.ylabel("Ймовірність відмови")
plt.grid()
plt.show()
```

Код для побудови залежності ймовірності відмови комітету від кількості та частки вузлів

```
import random
from math import comb
import matplotlib.pyplot as plt

class Sharding:
    def __init__(self, num_nodes, num_shards, max_byzantine_fraction):
        self.num_nodes = num_nodes
        self.num_shards = num_shards
        self.max_byzantine_fraction = max_byzantine_fraction

    def compute_failure_probability(self, committee_size,
max_byzantine_nodes):
        """Обчислення ймовірності відмови комітету."""
        F = int(self.max_byzantine_fraction * self.num_nodes) # Кількість
візантійських вузлів
        N = self.num_nodes
        n = committee_size
        f = max_byzantine_nodes

        failure_probability = 0
        for x in range(f, n + 1):
            failure_probability += (comb(F, x) * comb(N - F, n - x)) /
comb(N, n)
        return failure_probability

    def simulate_committee_failure(self):
        """Симуляція для кожного шарду."""
        committee_size = self.num_nodes // self.num_shards
        max_byzantine_nodes = (committee_size - 1) // 2
        return self.compute_failure_probability(committee_size,
max_byzantine_nodes)

# Тестування для різної кількості вузлів
num_shards = 4
max_byzantine_fraction = 0.3
nodes_range = range(10, 101, 10) # Кількість вузлів від 10 до 100 з кроком
10
failure_probabilities_nodes = []

for num_nodes in nodes_range:
    sharding = Sharding(num_nodes, num_shards, max_byzantine_fraction)
    failure_probability = sharding.simulate_committee_failure()
    failure_probabilities_nodes.append(failure_probability)

# Тестування для різної частки візантійських вузлів
num_nodes = 100
```

```

fractions_range = [i / 100 for i in range(5, 51, 5)] # Від 0.05 до 0.5 з
кроком 0.05
failure_probabilities_fractions = []

for fraction in fractions_range:
    sharding = Sharding(num_nodes, num_shards, fraction)
    failure_probability = sharding.simulate_committee_failure()
    failure_probabilities_fractions.append(failure_probability)

# Побудова графіків
plt.figure(figsize=(12, 6))

# Графік для показу залежності від кількості вузлів
plt.subplot(1, 2, 1)
plt.plot(nodes_range, failure_probabilities_nodes, marker='o', color='b')
plt.title("Ймовірність відмови vs Кількість вузлів")
plt.xlabel("Кількість вузлів")
plt.ylabel("Ймовірність відмови комітету")
plt.grid()

# Графік для показу залежності від частки візантійських вузлів
plt.subplot(1, 2, 2)
plt.plot(fractions_range, failure_probabilities_fractions, marker='o',
color='r')
plt.title("Ймовірність відмови vs Частка візантійських вузлів")
plt.xlabel("Частка візантійських вузлів")
plt.ylabel("Ймовірність відмови комітету")
plt.grid()

plt.tight_layout()
plt.show()

```