Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол №__ від ____ червня 2026 р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)
«___» червня 2026 р.

Кваліфікаційна робота
здобувача першого (бакалаврського) рівня вищої освіти

«МЕТОД ВЕРИФІКАЦІЇ ЦІЛІСНОСТІ ДАНИХ У ВИСОКОДОСТУПНИХ РОЗПОДІЛЕНИХ БАЗАХ ДАНИХ (NoSQL) З ВИКОРИСТАННЯМ БЛОКЧЕЙН-ТЕХНОЛОГІЙ»

Спеціальність ***123 – Комп'ютерна інженерія***

Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Нікіта ХРИСТЕНКО**
(підпис)

Науковий керівник _____ **Ольга МОРОЗ**
(підпис)

Харків – 2026

АНОАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 83 сторінки, із яких 50 сторінок основної частини, 7 рисунків, 18 таблиць, 42 найменуваня у списку використаних джерел та 4 додатків.

Метою кваліфікаційної роботи є забезпечення контролю узгодженості стану вузлів та виявлення несанкціонованих змін даних у високодоступних розподілених NoSQL-системах на основі блокчейн-технологій шляхом розробки методу верифікації цілісності даних.

Об'єкт дослідження – процеси забезпечення цілісності та узгодженості даних у високодоступних розподілених NoSQL-системах.

Предмет дослідження – методи та алгоритми блокчейн-верифікації цілісності даних і контролю узгодженості станів вузлів у розподілених NoSQL-системах.

У роботі розглянуто проблему контролю цілісності даних у розподілених NoSQL-системах і запропоновано її вирішення через блокчейн-верифікацію хешів подій та станів вузлів. Реалізовано прототип на Python і MongoDB з журналюванням, перевіркою та відновленням даних. Експерименти підтвердили виявлення несанкціонованих змін, видалень і розсинхронізації, а також відновлення узгодженого стану.

Область застосування – розподілені інформаційні системи, центри обробки даних, корпоративні NoSQL-сховища, системи аудиту подій, критичні інформаційні інфраструктури, хмарні платформи, а також освітній процес за спеціальністю 123 «Комп'ютерна інженерія».

Ключові слова: комп'ютерна інженерія, розподілені системи, NoSQL, MongoDB, блокчейн, верифікація даних, контроль цілісності, журнал подій, хешування, Merkle tree, відмовостійкість, реплікація даних, високодоступні системи, децентралізовані системи, Python.

ABSTRACT

The explanatory note to the bachelor's qualification work consists of an introduction, four chapters, conclusions, a list of references and appendices. The total volume of the work is 83 pages, including 50 pages of the main content, 7 figures, 18 tables, 42 references and 4 appendices.

The purpose of the qualification work is to improve reliability, fault tolerance and trustworthiness of data in distributed computer systems through the development of a method and architectural solutions for verifying the integrity of data and the state of NoSQL storage systems using decentralized blockchain technologies.

The object of research is the architecture and operational processes of highly available distributed computer systems based on NoSQL storage systems under conditions of decentralized data integrity control.

The subject of research is models, methods and software-hardware tools for blockchain-based verification of data integrity, event logs and node states in highly available distributed NoSQL systems.

The work addresses the problem of data integrity control in distributed NoSQL systems and proposes a solution based on blockchain verification of event and node state hashes. A prototype system is implemented using Python and MongoDB, including event logging, data verification, and recovery mechanisms. Experimental results confirm the system's ability to detect unauthorized modifications, data deletion, and node desynchronization, as well as to restore a consistent system state.

The scope of application includes distributed information systems, data centers, corporate NoSQL storage systems, event auditing systems, critical information infrastructures, cloud platforms, and the educational process in the field of specialty 123 "Computer Engineering".

Keywords: computer engineering, distributed systems, NoSQL, MongoDB, blockchain, data verification, integrity control, event log, hashing, Merkle tree, fault tolerance, data replication, highly available systems, decentralized systems, Python.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 7
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСТІ ДАНИХ У РОЗПОДІЛЕНИХ СИСТЕМАХ 10
1.1. Високодоступні розподілені бази даних та їх особливості 10
1.2. NoSQL-бази даних: типи, архітектура, сфери застосування 14
1.3. Проблеми цілісності та довіри до даних у розподілених середовищах 15
1.4. Основи блокчейн-технологій та криптографічних методів 17
1.5. Аналіз сучасних методів контролю цілісності даних у розподілених NoSQL-системах 19
Висновок до розділу 1 22
РОЗДІЛ 2. РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ВЕРИФІКАЦІЇ ЦІЛІСНОСТІ ДАНИХ У РОЗПОДІЛЕНОМУ NoSQL-КЛАСТЕРІ 23
2.1. Вимоги до системи та обмеження 23
2.2. Загальна архітектура запропонованого рішення 24
2.3. Модель інформаційних потоків між компонентами системи 25
2.4. Метод верифікації цілісності даних у розподіленому NoSQL-кластері 27
2.5. Структура блоку та механізм хешування 29
2.6. Сценарії роботи системи та обробка збоїв 31
Висновок до розділу 2 32
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ВЕРИФІКАЦІЇ 34
3.1. Вибір інструментів та технологій розробки 34
3.2. Реалізація NoSQL-сховища та журналу подій 36
3.3. Реалізація власного блокчейн-модуля 38
3.4. Алгоритми контролю цілісності даних 39
3.5. Демонстрація роботи програмного прототипу 42

Висновок до розділу 3 43
РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА ФУНКЦІОНУВАННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ ВЕРИФІКАЦІЇ 45
4.1. Сценарії функціонування системи та методика проведення експериментів 45
4.2. Експерименти зі збоями та порушенням цілісності 46
4.3. Аналіз результатів верифікації 49
4.4. Оцінка продуктивності та накладних витрат 49
4.5. Порівняння з традиційними підходами 50
Висновок до розділу 4 51
ВИСНОВКИ 52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 54
ДОДАТКИ 58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД	–	база даних
СУБД	–	система управління базами даних
NoSQL	–	Not Only SQL нереляційна база даних
SQL	–	Structured Query Language
CRDT	–	Conflict-free Replicated Data Type тип даних із автоматичним усуненням конфліктів реплікації
CAP	–	Consistency, Availability, Partition Tolerance теорема CAP
JSON	–	JavaScript Object Notation
CSV	–	Comma-Separated Values
SHA-256	–	Secure Hash Algorithm 256-bit
IoT	–	Internet of Things
Merkle Root	–	кореневий хеш дерева Меркла
Snapshot	–	контрольний знімок стану даних
Audit Log	–	журнал аудиту подій

ВСТУП

Стрімкий розвиток інформаційних технологій, хмарних обчислень, систем обробки великих даних та розподілених сервісів призвів до широкого використання високодоступних комп'ютерних систем, побудованих на базі NoSQL-сховищ даних. Такі системи активно застосовуються в корпоративних інформаційних платформах, центрах обробки даних, хмарних сервісах, фінансових системах, мережевій інфраструктурі та інших критично важливих сферах, де необхідно забезпечити безперервність функціонування та високу швидкодію при роботі з великими обсягами інформації.

Однією з ключових особливостей NoSQL-систем є використання механізмів реплікації та розподіленого зберігання даних, що дозволяє забезпечити високу доступність сервісів навіть у випадку відмов окремих вузлів. Проте поряд із перевагами таких систем виникають нові проблеми, пов'язані з контролем цілісності даних, достовірністю журналів подій та виявленням несанкціонованих змін у розподіленому середовищі. У традиційних рішеннях контроль цілісності часто покладається на внутрішні механізми системи управління базами даних або централізовані журнали аудиту, які можуть бути модифіковані разом із даними або втрачати актуальність у випадку збоїв.

Особливо актуальною ця проблема стає для високодоступних розподілених систем, де одночасно функціонують декілька вузлів із реплікованими даними. У таких умовах виникає необхідність забезпечення механізму незалежної перевірки стану даних, який дозволяв би виявляти несанкціоновані зміни, незаконне видалення записів, розсинхронізацію реплік та інші порушення цілісності незалежно від внутрішнього стану окремого вузла.

Одним із перспективних напрямів вирішення зазначеної проблеми є використання блокчейн-технологій. Завдяки властивостям незмінності записів, криптографічному захисту та децентралізованому характеру зберігання

інформації блокчейн може виступати додатковим рівнем контролю цілісності даних у розподілених NoSQL-системах. Інтеграція блокчейн-модуля дозволяє фіксувати контрольні хеші подій та станів системи, забезпечуючи можливість подальшої верифікації незалежно від змін, які могли відбутися в основному сховищі даних.

Актуальність роботи обумовлена необхідністю підвищення рівня довіри до даних у високодоступних розподілених системах, забезпечення контролю цілісності інформації та підвищення відмовостійкості сучасних NoSQL-сховищ в умовах децентралізованого середовища функціонування.

Метою кваліфікаційної роботи є забезпечення контролю узгодженості стану вузлів та виявлення несанкціонованих змін даних у високодоступних розподілених NoSQL-системах на основі блокчейн-технологій шляхом розробки методу верифікації цілісності даних.

Об'єкт дослідження – процеси забезпечення цілісності та узгодженості даних у високодоступних розподілених NoSQL-системах.

Предмет дослідження – методи та алгоритми блокчейн-верифікації цілісності даних і контролю узгодженості станів вузлів у розподілених NoSQL-системах.

Завдання дослідження:

1. Провести аналіз архітектур високодоступних розподілених NoSQL-систем та дослідити проблеми забезпечення цілісності даних.
2. Дослідити сучасні методи контролю цілісності інформації, механізми журналювання подій та можливості застосування блокчейн-технологій у розподілених системах.
3. Розробити архітектуру системи блокчейн-верифікації цілісності даних для розподіленого NoSQL-кластера.
4. Розробити метод та алгоритми контролю цілісності даних і перевірки узгодженості станів вузлів.
5. Реалізувати програмний прототип системи на базі MongoDB та Python.

6. Провести моделювання порушень цілісності даних, відмов вузлів і розсинхронізації реплік.

7. Виконати експериментальне дослідження та оцінити ефективність запропонованого методу за показниками виявлення порушень цілісності та накладних витрат.

Методи дослідження: теорія розподілених систем, методи системного аналізу, криптографічні методи хешування даних, методи контролю цілісності інформації, технології блокчейн, методи моделювання відмовостійких систем, засоби програмної реалізації на базі Python та MongoDB.

Практичне значення одержаних результатів полягає у можливості використання запропонованого методу для підвищення рівня довіри до даних у розподілених NoSQL-системах, виявлення несанкціонованих змін та забезпечення контролю цілісності інформації в умовах відмов окремих вузлів і децентралізованого середовища функціонування.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСТІ ДАНИХ У РОЗПОДІЛЕНИХ СИСТЕМАХ

1.1. Високодоступні розподілені бази даних та їх особливості

Розподілені високодоступні бази даних забезпечують зберігання та реплікацію даних на кількох вузлах, що дає змогу витримувати відмови окремих компонентів і масштабувати систему горизонтально. Для цього зазвичай застосовують реплікаційні схеми, зокрема моделі з виділеним провідним вузлом, рівноправні багатовузлові схеми та системи, що використовують кворум. Окремо важливим механізмом є шардінг, тобто горизонтальне розподілення даних між кількома вузлами або групами вузлів .

З теореми CAP випливає, що у розподіленому середовищі неможливо одночасно гарантувати повну узгодженість, високу доступність і стійкість до розділення мережі [38]. Тому практичні системи завжди обирають певний компроміс між цими властивостями. До поширених моделей консистентності належать сильна, кінцева, каузальна та інші варіації, які визначають, наскільки швидко й узгоджено різні вузли відображають оновлення даних.

Основними загрозами для таких систем є відмова вузлів, розділення кластера на ізольовані частини, компрометація окремих серверів, а також несанкціонована зміна журналів операцій. Для протидії цим загрозам застосовуються криптографічні механізми: хеш-функції, Merkle-дерева, цифрові підписи, коди автентифікації повідомлень і захищені канали передавання даних. У сукупності ці засоби дають змогу виявляти несанкціоновані зміни, підтверджувати авторство операцій і захищати обмін між вузлами.

Теорема CAP (теорема Брюера) – це фундаментальний принцип у розподілених системах, який стверджує, що у розподіленій системі неможливо одночасно гарантувати всі три властивості: консистентність, доступність і стійкість до розділення мережі.

– ***C*** (Consistency – узгодженість), усі вузли бачать однакові, найактуальніші дані в один і той самий момент.

– ***A*** (Availability – доступність), кожен запит отримує відповідь, навіть якщо вона може бути не найсвіжішою.

– ***P*** (Partition tolerance – стійкість до розділення), система продовжує працювати навіть при розриві мережі між вузлами.

1.1.1. Визначення та ключові поняття

Розподілена база даних – це сукупність логічно пов'язаних даних, фізично розміщених на різних вузлах мережі та керованих як єдине ціле. Висока доступність означає, що система спроектована так, аби мінімізувати час простою навіть за умови відмови частини інфраструктури. Для цього використовуються реплікація, моніторинг стану вузлів і механізми автоматичного перемикання на резервний вузол.

Типовим прикладом є кластерна організація на основі провідного вузла та реплік, де один вузол приймає записи, а інші синхронізують зміни. У разі відмови провідного вузла один із резервних вузлів може бути обраний новим активним елементом кластера, що дозволяє зберегти безперервність роботи. Подібний підхід використовується в багатьох сучасних системах. Водночас існують і рівноправні архітектури, у яких усі вузли мають однаковий статус і можуть обробляти операції без виділеного майстра.

Теорема CAP стверджує, що розподілена система не може одночасно забезпечити повну узгодженість і доступність під час мережевого розділення. На практиці це означає, що різні системи віддають пріоритет або консистентності, або доступності. Одні бази даних орієнтовані на суворішу узгодженість, інші – на вищу доступність і масштабованість, допускаючи тимчасові розбіжності між копіями.

До основних моделей консистентності належать:

– *Сильна консистентність*, за якої будь-яке читання повертає найсвіжіше зафіксоване значення.

- *Кінцева консистентність*, коли дані можуть тимчасово відрізнятися на різних вузлах, але з часом синхронізуються.
- *Каузальна консистентність*, яка зберігає порядок операцій, пов'язаних причинно-наслідковими зв'язками.
- *Read-your-writes*, що гарантує клієнту видимість власних останніх записів.

Вибір моделі залежить від вимог до продуктивності, затримок і точності даних.

1.1.2. Архітектурні схеми та моделі

У високодоступних системах найчастіше застосовуються дві базові архітектури, а саме провідний–резервний підхід і рівноправна багатовузлова модель. У першому випадку запис виконується через провідний вузол, у другому – усі вузли є рівноправними. Для забезпечення узгодженості часто застосовуються кворумні механізми. Масштабування реалізується шляхом шардінгу, що дозволяє розподіляти дані між вузлами кластера [28,35].

Окреме місце займає кворумний підхід, за якого операція вважається успішною лише після підтвердження більшістю реплік. Такий механізм дозволяє підтримувати узгодженість навіть у разі відмови частини вузлів. Саме тому кворумні стратегії широко застосовуються у розподілених сховищах, де важливо зберегти баланс між доступністю та точністю [28, 38].

Ще одним важливим механізмом є шардінг – горизонтальне розподілення даних на частини, які зберігаються на різних вузлах. Це дає змогу збільшувати продуктивність і масштабувати систему без повної перебудови сховища. Однак шардінг ускладнює виконання складних міжшардових операцій і вимагає додаткової координації [35].

1.1.3. Типові загрози та вектори атак

У розподілених високодоступних базах даних існує кілька характерних класів загроз. Одні з них пов'язані з технічними збоями, інші – з навмисними діями зловмисника.

До типових ризиків належать:

- Розділення мережі, коли кластер ділиться на ізольовані частини, що може призвести до розбіжностей між копіями даних.
- Компрометація вузла, унаслідок якої зловмисник здобуває можливість змінювати дані або журнали.
- Несанкціонована модифікація журналів, що порушує цілісність реплік і ускладнює відновлення коректного стану системи.
- Відмови візантійського типу, коли окремі вузли поводяться непередбачувано або навмисно передають хибну інформацію.
- Перехоплення мережевого трафіку, якщо канали між вузлами не захищені.
- Недостатньо продумана логіка перемикання, що може призводити до помилкових виборів активного вузла або до тривалої недоступності сервісу.

Отже, захист у таких системах має враховувати як надійність інфраструктури, так і криптографічну перевірку цілісності та автентичності даних.

1.1.4. Криптографічні та алгоритмічні механізми захисту

Для підвищення цілісності та довіри до даних застосовують низку криптографічних і алгоритмічних засобів.

Хеш-функції використовуються для отримання короткого цифрового відбитка даних. Будь-яка зміна запису призводить до зміни його хеша, що дає змогу виявляти підробку або спотворення [39].

Merkle-дерева являють собою ієрархічну структуру, де нижній рівень містить хеші окремих записів, а верхній рівень формує узагальнений відбиток усього набору даних. Такий підхід дозволяє ефективно перевіряти цілісність великих журналів і сховищ [40].

Цифрові підписи забезпечують не лише цілісність, а й автентичність змін. Якщо критичні записи підписуються приватним ключем вузла, інші учасники можуть перевірити, що зміни справді походять від авторизованого джерела.

Коди автентифікації повідомлень забезпечують контроль цілісності даних під час передавання, а захищені транспортні канали гарантують конфіденційність і захист від підміни трафіку. Криптографічно захищені журнали подій, побудовані на основі хеш-ланцюгів або цифрових підписів, дозволяють виявляти будь-які спроби несанкціонованого втручання [33].

Додатково мітки часу забезпечують коректне впорядкування подій і запобігають повторному використанню повідомлень. Таким чином, поєднання реплікації, механізмів перевірки цілісності, цифрових підписів і захищених каналів формує надійну основу для функціонування розподілених баз даних.

1.2. NoSQL-бази даних: типи, архітектура, сфери застосування

NoSQL-бази даних – це клас нереляційних систем керування даними, що призначені для роботи з великими обсягами різнорідної інформації та забезпечення високої масштабованості. На відміну від традиційних реляційних систем, вони не обмежуються жорстко визначеними схемами та дозволяють зберігати дані у більш гнучких структурах. Порівняльний аналіз найбільш поширених типів NoSQL-баз даних наведено в табл. 1.1 [28, 29, 34–37].

Таблиця 1.1.

Порівняльна характеристика основних типів NoSQL-баз даних.

Тип	Особливість	Приклад
Key-Value	зберігання пар ключ-значення	Redis
Document	документи JSON/BSON	MongoDB
Wide-column	великі масиви даних	Cassandra
Graph	зв'язки між об'єктами	Neo4j
Time-series	часові ряди	InfluxDB

Як видно з табл. 1.1, документно-орієнтовані бази даних забезпечують оптимальне поєднання гнучкості структури даних, масштабованості та підтримки реплікації, що обумовило вибір MongoDB для реалізації програмного прототипу системи [35].

Таким чином, вибір конкретного типу NoSQL-бази визначається характером задачі, вимогами до продуктивності та необхідним рівнем узгодженості даних.

1.3. Проблеми цілісності та довіри до даних у розподілених середовищах

Питання цілісності та довіри до даних у розподілених середовищах є одним із ключових для сучасних інформаційних систем. На відміну від централізованих баз даних, у яких контроль над станом інформації зосереджений в одному обчислювальному контурі, у розподілених системах дані проходять через кілька вузлів, каналів передавання, механізмів реплікації та етапів синхронізації. У таких умовах виникає не лише технічна складність підтримання однакового стану даних, а й проблема перевірки того, що ці дані залишаються достовірними протягом усього життєвого циклу.

1.3.1. Цілісність даних як багатовимірна характеристика

У розподілених системах довіра до даних ускладнюється наявністю декількох вузлів, мережевих затримок і можливих відмов обладнання. Навіть за наявності механізмів реплікації не завжди можливо однозначно підтвердити актуальність і достовірність інформації. Тому контроль цілісності є одним із ключових завдань сучасних розподілених систем [1], [2], [38]

1.3.2. Природа довіри до даних у розподіленому середовищі

Довіра до даних у розподіленій системі не виникає автоматично. Вона формується на основі припущень про надійність вузлів, коректність програмного забезпечення, безпеку каналів передавання, контроль доступу та здатність системи виявляти аномальні ситуації. Якщо в централізованій системі джерелом правди зазвичай виступає один сервер або одна СУБД, то у розподіленому середовищі ця роль розподіляється між механізмами реплікації, журналювання, узгодження та відновлення [16], [18].

Довіру до даних доцільно розглядати на кількох рівнях. На нижчому рівні вона пов’язана з апаратним забезпеченням та операційною системою. На середньому – з вузлами, які зберігають і передають інформацію. На вищому – з політиками доступу, процедурами аудиту та механізмами підтвердження змін.

Якщо хоча б один із цих рівнів працює некоректно, виникає ризик втрати цілісності або появи суперечностей між копіями даних.

Особливо складною є ситуація в середовищах, де частина вузлів належить різним адміністраторам або навіть різним організаціям. У такому випадку довіра не може спиратися лише на припущення про чесність усіх учасників. Потрібні технічні засоби перевірки, які дозволяють відстежити історію змін і підтвердити її незмінність незалежно від внутрішнього стану окремого вузла [39,40].

1.3.3. Основні джерела порушення цілісності

Порушення цілісності в розподілених системах може виникати як з технічних, так і з організаційних причин. До найпоширеніших належать мережеві збої, затримки передавання, часткова втрата доступності вузлів, помилки конфігурації, некоректна робота механізмів реплікації та незаплановані зміни даних на різних копіях. Однією з найбільш типових проблем є ситуація, коли зміна зафіксована лише на одному вузлі, але ще не поширена на інші. У цей проміжок часу різні клієнти можуть бачити різні версії одного й того самого запису. З погляду архітектури це не завжди є помилкою, однак у практичних системах така розбіжність може призвести до неправильних рішень, якщо прикладна логіка очікує більш жорсткої узгодженості [28], [35].

Іншим джерелом проблем є паралельні оновлення одного й того самого об’єкта. Якщо система не має достатньо надійного механізму розв’язання конфліктів, результат може бути неоднозначним: частина змін буде втрачена, а частина – застосована у непередбачуваній послідовності. Схожа ситуація виникає і під час відновлення після відмови, коли частина журналів уже втрачена або резервна копія не відповідає актуальному стану кластера [16,24].

1.3.4. Узгодженість, автентичність та трасованість даних

Реплікація даних у розподілених системах може призводити до виникнення різних версій одного запису на окремих вузлах. Для визначення актуального стану використовуються механізми консистентності та синхронізації реплік.

Важливими вимогами також є автентичність даних і можливість простежити історію їх змін. Для цього застосовуються журнали аудиту, криптографічні хеш-функції та цифрові підписи. Поєднання зазначених механізмів дозволяє підвищити рівень довіри до інформації та забезпечити контроль її походження [16,24,39].

1.4. Основи блокчейн-технологій та криптографічних методів

Блокчейн є технологією розподіленого зберігання даних, що забезпечує незмінність записів за рахунок криптографічного зв'язування блоків. У роботі блокчейн розглядається як додатковий механізм контролю цілісності та аудиту даних у NoSQL-середовищі [26], [27]

1.4.1. Концепція блокчейну як механізму довіри

Блокчейн можна розглядати як спеціалізовану форму розподіленого журналу, в якому записи організовані у вигляді послідовності блоків, пов’язаних між собою криптографічними залежностями. Кожен новий блок містить посилання на попередній, що створює незмінний ланцюг даних. З огляду на проблеми, описані у підрозділі 1.3, така структура дозволяє вирішити одну з ключових задач, а саме забезпечення незмінності історії змін. Якщо будь-який запис у ланцюгу змінюється, це призводить до порушення зв’язку між блоками, що легко виявляється під час перевірки. У контексті високодоступних розподілених баз даних блокчейн не обов’язково використовується як основне сховище даних. Натомість він може виконувати роль додаткового рівня верифікації, де зберігаються контрольні відбитки стану системи або журналів змін. Такий підхід дозволяє поєднати високу продуктивність NoSQL-систем із надійністю криптографічного підтвердження цілісності [32].

1.4.2. Криптографічні основи перевірки цілісності

Як було зазначено у підрозділі 1.1, криптографічні методи відіграють ключову роль у забезпеченні цілісності даних. У поєднанні з блокчейн-технологіями вони утворюють комплексний механізм, що дозволяє не лише виявляти зміни, а й доводити їхню правомірність.

Основою таких підходів є використання хеш-функцій, які перетворюють довільний обсяг даних у фіксований за розміром відбиток. Важливою властивістю є те, що навіть незначна зміна вхідних даних призводить до суттєвої зміни результату [39]. Це дозволяє використовувати хеші як індикатори цілісності.

Подальший розвиток цієї ідеї реалізується через структури типу Merkle-дерев, які дозволяють ефективно перевіряти великі обсяги даних [33]. У таких структурах перевірка цілісності окремого елемента не потребує аналізу всього набору, що є критично важливим для масштабованих систем. Крім того, цифрові підписи забезпечують автентичність джерела даних. Якщо запис підписаний приватним ключем, будь-який інший вузол може перевірити його достовірність за допомогою відповідного публічного ключа. Це безпосередньо пов'язано з проблемою довіри, розглянутою раніше, оскільки дозволяє встановити, хто саме ініціював зміну.

Таким чином, криптографія забезпечує три ключові властивості, а саме цілісність, автентичність і незаперечність, які є основою для побудови довірених розподілених систем.

1.4.3. Інтеграція блокчейну з NoSQL-системами

У даній роботі блокчейн використовується не як основне сховище даних, а як незалежний журнал аудиту та механізм перевірки цілісності стану розподіленого NoSQL-кластера [26,33].*1.4.4. Консенсус як основа узгодженої довіри*

Консенсус забезпечує узгодження стану між учасниками блокчейн-мережі та визначає коректну версію даних. У контексті розподілених NoSQL-систем механізми консенсусу можуть використовуватись для підтвердження достовірності контрольних записів і результатів верифікації [16], [25]

1.4.5. Забезпечення незмінності та перевірюваності даних

Незмінність даних досягається шляхом криптографічного зв'язування блоків. Будь-яка модифікація раніше збереженої інформації призводить до зміни контрольних хешів і легко виявляється під час перевірки. Це дозволяє

використовувати блокчейн як незалежний механізм підтвердження цілісності та історії змін. [16]–[25]

1.4.6. Обмеження та виклики використання блокчейну

Попри значні переваги, інтеграція блокчейн-технологій із розподіленими базами даних має певні обмеження. Зокрема, це стосується продуктивності, затримок при досягненні консенсусу та обмеженої пропускної здатності системи.

Крім того, зберігання великого обсягу даних безпосередньо у блокчейні є неефективним, тому на практиці використовується гібридний підхід, де основні дані залишаються у NoSQL-сховищі, а блокчейн зберігає лише їхні контрольні відбитки.

Ще одним викликом є складність інтеграції та необхідність узгодження між різними рівнями системи. Неправильна реалізація може призвести до ситуацій, коли блокчейн не відображає реальний стан даних або не використовується ефективно.

1.5. Аналіз сучасних методів контролю цілісності даних у розподілених NoSQL-системах

Забезпечення цілісності даних у розподілених системах є складною багаторівневою задачею, що поєднує архітектурні підходи, моделі консистентності, які були розглянуті у підрозділах 1.1–1.2 та криптографічні механізми захисту. На практиці сучасні системи не покладаються на один метод, а використовують комбінацію вбудованих механізмів контролю цілісності, що реалізуються на різних рівнях, від зберігання даних до мережевої взаємодії [16]–[35]

1.5.1. Підходи до забезпечення цілісності в сучасних СУБД

Більшість сучасних розподілених баз даних реалізують контроль цілісності через поєднання реплікації, журналювання операцій та механізмів узгодження стану між вузлами. У документно-орієнтованих системах, таких як MongoDB, цілісність підтримується через журнали змін і механізми реплікації. Дані записуються послідовно, а зміни поширюються на інші вузли кластера.

Узгодженість досягається через підтвердження записів більшістю вузлів, що дозволяє уникнути втрати даних при відмовах. У стовпчикових розподілених системах, таких як Apache Cassandra, контроль цілісності базується на поєднанні кворумних операцій та механізмів відновлення, таких як фонова синхронізація реплік. Оскільки такі системи часто орієнтовані на модель кінцевої консистентності, вони використовують додаткові процедури узгодження для виправлення розбіжностей між вузлами. Системи типу ключ-значення, наприклад Redis, забезпечують цілісність через механізми збереження стану, типу журнали операцій або знімки пам'яті, та реплікацію. Однак у таких системах, орієнтованих на швидкодію, контроль цілісності часто має спрощений характер і може потребувати додаткових зовнішніх механізмів.

1.5.2 Порівняльний аналіз існуючих підходів

Для визначення переваг і недоліків існуючих рішень було проведено порівняльний аналіз найбільш поширених підходів до контролю цілісності даних у розподілених системах. Результати аналізу наведено в табл. 1.2.

Таблиця 1.2.

Порівняльний аналіз підходів контролю цілісності даних.

Метод	Виявлення модифікацій	Відновлення	Децентралізація
Реплікація даних	Частково	Так	Так
Журналювання подій (Audit Log)	Так	Частково	Ні
Кворумні механізми	Частково	Так	Так
Хешування	Так	Ні	Ні
Merkle Tree	Так	Ні	Частково
Центральний аудит	Так	Так	Ні
Blockchain-підхід	Так	Так	Так

Як видно з табл. 1.2, традиційні механізми реплікації та резервного копіювання забезпечують базовий рівень захисту даних, однак не дозволяють гарантовано виявляти приховані зміни або підтверджувати достовірність історії модифікацій. Натомість використання криптографічних механізмів і блокчейн-технологій забезпечує можливість незалежної перевірки цілісності даних та аудиту виконаних операцій. Саме тому для розв'язання поставленої задачі

доцільним є застосування підходу, що поєднує можливості NoSQL-систем із механізмами блокчейн-верифікації [16–25, 28, 33, 35].

1.5.3. Криптографічні рішення контролю цілісності

Як було зазначено у підрозділі 1.1, криптографічні методи є ключовим інструментом для виявлення несанкціонованих змін даних. У сучасних рішеннях вони реалізуються у кількох формах. Хешування використовується для контролю цілісності окремих записів або блоків даних. У більш складних системах застосовуються ієрархічні структури, такі як Merkle-дерева, що дозволяють ефективно перевіряти великі обсяги інформації без необхідності повного порівняння. Цифрові підписи забезпечують підтвердження походження даних. Вони особливо важливі у середовищах із частково недовіреними вузлами, де необхідно гарантувати, що зміни були внесені авторизованими учасниками. Окремий клас рішень базується на концепції незмінних журналів. Наприклад, система Amazon QLDB реалізує модель журналу, де всі зміни зберігаються у вигляді криптографічно пов'язаного ланцюжка, що унеможливлює непомітну модифікацію історії даних.

1.5.4. Алгоритмічні механізми узгодження та перевірки

Окрім криптографії, значну роль відіграють алгоритмічні підходи до забезпечення цілісності. Як зазначалося раніше, у розподілених системах широко застосовуються кворумні механізми, що дозволяють гарантувати узгодженість за рахунок підтвердження більшістю вузлів.

Додатково використовуються механізми:

- Анти-ентропії – періодичне порівняння та синхронізація даних між вузлами;
- Read-repair – виправлення неузгоджених даних під час операцій читання;
- Write-repair – забезпечення узгодженості під час запису;
- Конфлікт-резолюції – автоматичне або напівавтоматичне вирішення суперечностей між версіями даних.

У деяких системах застосовуються структури типу CRDT, які дозволяють автоматично об'єднувати зміни без втрати інформації навіть у разі паралельних оновлень.

1.5.5. Порівняльний аналіз підходів

Аналіз існуючих рішень показує, що кожен підхід має власні переваги та обмеження. Реплікація і кворумні механізми забезпечують доступність даних, криптографічні методи підвищують рівень довіри та можливість аудиту, а алгоритми узгодження підтримують консистентність у розподіленому середовищі. Тому сучасні системи контролю цілісності використовують комплексний підхід, поєднуючи архітектурні, алгоритмічні та криптографічні механізми. З урахуванням обмежень теореми CAP перспективним напрямом є інтеграція криптографічних засобів у процеси реплікації та журналювання для підвищення достовірності даних без суттєвого впливу на продуктивність системи.

Висновок до розділу 1

У першому розділі досліджено особливості високодоступних розподілених NoSQL-систем, механізми реплікації та забезпечення узгодженості даних. Проведено аналіз проблем контролю цілісності у розподіленому середовищі та розглянуто сучасні криптографічні методи захисту інформації. Встановлено доцільність використання блокчейн-технологій як незалежного механізму підтвердження цілісності та аудиту даних. Результати аналізу стали основою для розробки методу верифікації цілісності даних у NoSQL-кластері..

РОЗДІЛ 2
РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ВЕРИФІКАЦІЇ ЦІЛІСНОСТІ ДАНИХ У РОЗПОДІЛЕНОМУ NoSQL-КЛАСТЕРІ

2.1. Вимоги до системи та обмеження

Запропонована система призначена для верифікації цілісності даних у розподіленому NoSQL-середовищі з використанням блокчейна як незмінного журналу подій. Її мета – виявлення несанкціонованих змін, видалення чи розсинхронізації даних між вузлами без значного впливу на продуктивність.

Функціональні вимоги передбачають підтримку операцій додавання, оновлення та видалення записів, автоматичне формування хешів подій, накопичення їх у батчі, створення блоків із прив'язкою до попереднього стану, перевірку цілісності даних через порівняння з еталонними хешами у блокчейні, а також моделювання збоїв вузлів і відновлення їх стану (див. табл. 2.1).

Таблиця 2.1

Вимоги до системи та обмеження

Категорія	Вимога / обмеження	Призначення
Функціональна	Підтримка операцій insert/update/delete	Робота з NoSQL-даними
Функціональна	Формування журналу подій	Фіксація змін
Функціональна	Хешування подій	Контроль цілісності
Функціональна	Формування блоків	Збереження audit trail
Функціональна	Перевірка snapshot hash	Виявлення модифікацій
Нефункціональна	Відмовостійкість	Робота при збоях вузлів
Нефункціональна	Масштабованість	Додавання нових вузлів
Нефункціональна	Низький overhead	Мінімальний вплив на продуктивність
Обмеження	Спрощений blockchain	Навчально-дослідницький характер
Обмеження	Локальна мережа	Відсутність реального WAN-кластера

Нефункціональні вимоги включають відмовостійкість, модульність архітектури, відтворюваність експериментів, простоту інтеграції з NoSQL-сховищами та можливість оцінки накладних витрат криптографічної верифікації.

Через навчально-дослідницький характер роботи система використовує спрощену модель блокчейна без повноцінного децентралізованого консенсусу.

Вона забезпечує незмінність послідовності блоків і перевірку історії змін, працюючи лише з тестовими наборами даних, достатніми для демонстрації принципів роботи.

2.2. Загальна архітектура запропонованого рішення

Архітектура запропонованого рішення (див. рис. 2.1) побудована за модульним принципом і містить кілька логічних відокремлених компонентів, а саме підсистему керування NoSQL-даними, модуль журналювання подій, блокчейн-модуль фіксації хешів, модуль верифікації та блок експериментальної симуляції збоїв.

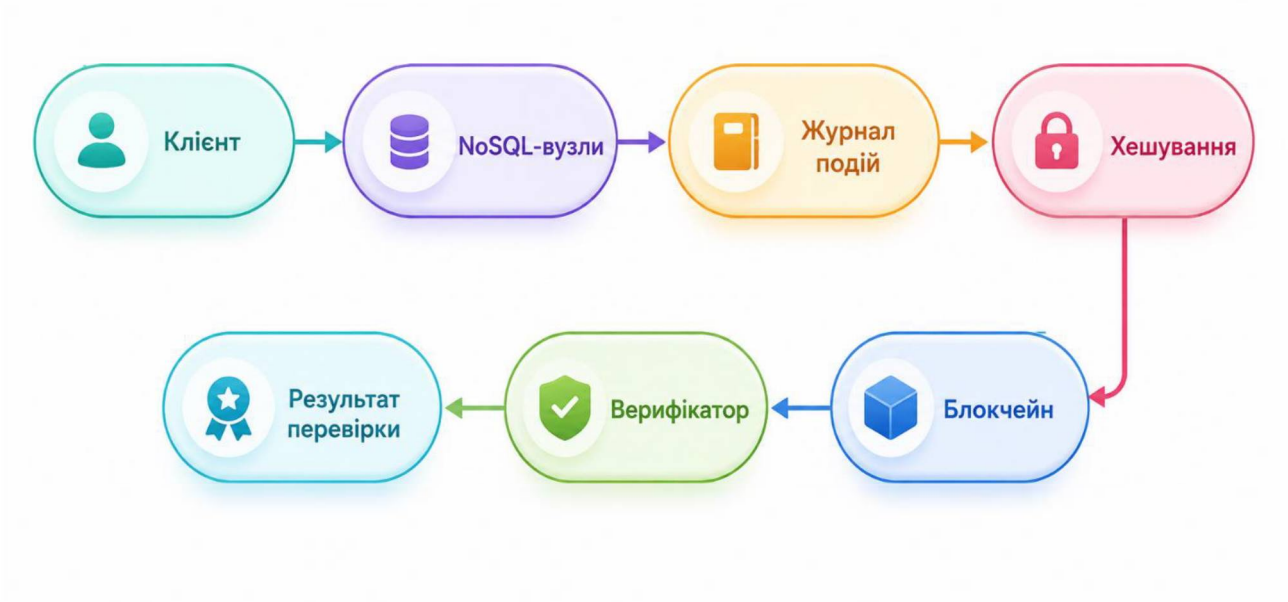


Рисунок 2.1 – Загальна архітектура системи.

На верхньому рівні система взаємодіє з клієнтським джерелом даних, яке ініціює операції над записами, далі зміни надходять до розподіленого NoSQL-кластера, де кожен логічний вузол зберігає власну копію даних, це дозволяє моделювати реплікацію, відмову вузлів і розходження станів. Кожна операція над даними супроводжується формуванням події, до якої входить тип операції, ідентифікатор запису, старий і новий стани, часовий мітка та список вузлів, на яких ця операція була застосована. Подія отримує криптографічний хеш і передається до блокчейн-модуля.

Блокчейн-модуль не виконує роль платіжної системи або універсальної децентралізованої платформи, а саме в межах цієї роботи – створення послідовного ланцюга блоків, кожен з яких містить набір хешів подій, Merkle root або агрегований хеш батчу, а також хеш попереднього блоку.

Таким чином, будь-яка зміна в історії подій порушує цілісність ланцюга. Модуль верифікації виконує порівняння поточного стану даних у вузлах кластера з еталонним станом, зафіксованим у блокчейні. Якщо хеші не збігаються, система реєструє порушення цілісності. Якщо виявлено розбіжність між вузлами, це вказує на розсинхронізацію або часткову втрату консистентності.

2.3 Модель інформаційних потоків між компонентами системи

Для забезпечення контролю цілісності даних у розподіленому NoSQL-кластері запропоновано модель взаємодії компонентів системи, яка поєднує механізми збору даних, криптографічної верифікації та зберігання контрольної інформації у блокчейн-середовищі. Модель визначає послідовність обробки даних та інформаційні потоки між основними компонентами системи (див. рис. 2.2).

Вузли MongoDB

Вузли MongoDB забезпечують зберігання основних даних користувачів та підтримують механізми реплікації. Дані можуть бути розподілені між декількома вузлами кластера для підвищення доступності та відмовостійкості.

Модуль збору даних (Verification Agent)

Модуль збору даних періодично отримує інформацію про стан колекцій, документи та службові метадані. Основним завданням модуля є формування набору даних для подальшої перевірки цілісності.

Модуль хешування

Після отримання даних система обчислює криптографічні хеші документів за допомогою алгоритму SHA-256. Отримані значення використовуються для виявлення будь-яких змін у структурі або вмісті документів.

Модуль побудови Merkle-дерева

На основі обчислених хешів формується Merkle-дерево, кореневий хеш якого є узагальненим цифровим відбитком поточного стану даних у кластері.

Блокчейн-модуль

Кореневий хеш Merkle-дерева передається до блокчейн-модуля, де зберігається у вигляді незмінного запису. Це дозволяє забезпечити незалежну перевірку цілісності даних та неможливість прихованої модифікації контрольної інформації.

Модуль перевірки

Під час виконання перевірки система повторно формує Merkle-дерево та порівнює отриманий кореневий хеш із значенням, збереженим у блокчейні. У випадку розбіжності запускається процедура локалізації змінених документів.

Адміністратор системи

Адміністратор отримує результати перевірки, звіти про порушення цілісності та інформацію про стан розподіленого сховища.

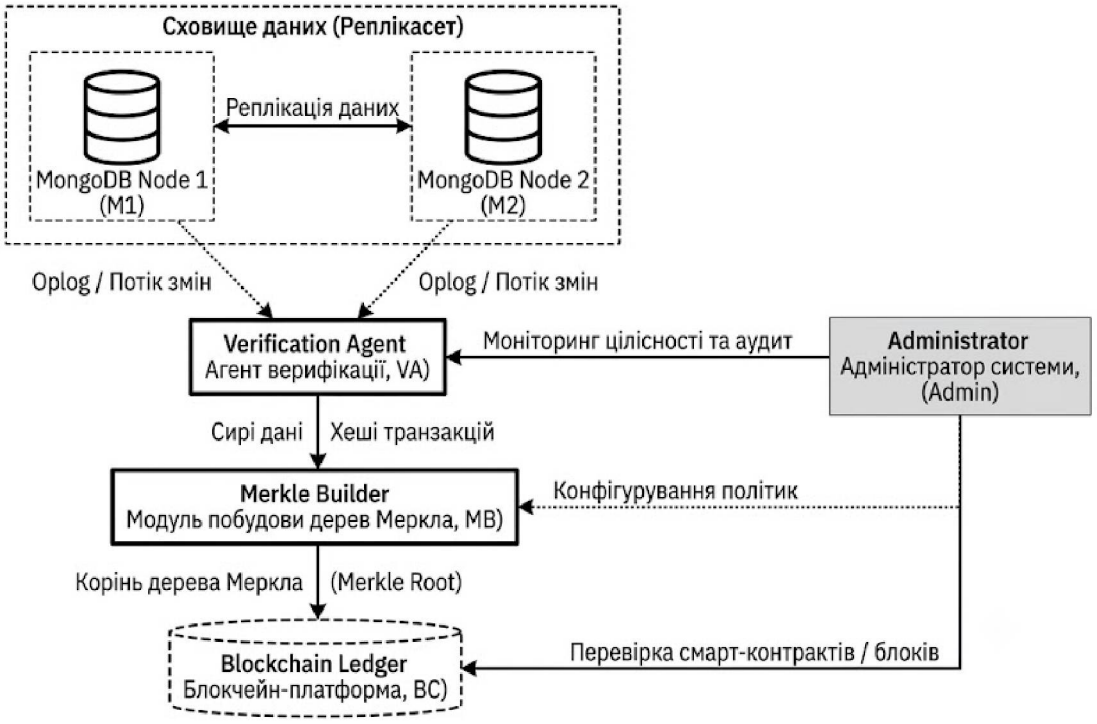


Рисунок 2.2 – Модель взаємодії компонентів системи.

2.4. Метод верифікації цілісності даних у розподіленому NoSQL-кластері

Запропонований метод верифікації цілісності даних у розподіленому NoSQL-кластері реалізується як послідовність взаємопов'язаних етапів:

– *Етап 1. Зчитування поточного стану даних.*

Виконується отримання документів із усіх вузлів розподіленого NoSQL-кластера та формування їх канонічного подання для подальшої обробки.

– *Етап 2. Формування контрольних хешів записів.*

Для кожного документа обчислюється криптографічний хеш за алгоритмом SHA-256, який виступає цифровим відбитком його поточного стану.

– *Етап 3. Журналювання подій.*

Для кожної операції додавання, зміни або видалення запису формується службова подія, що містить інформацію про виконану операцію, час її здійснення та змінені дані.

– *Етап 4. Побудова Merkle-дерева.*

Отримані хеші подій групуються у структуру Merkle Tree для компактного представлення великої кількості змін.

– *Етап 5. Обчислення Merkle Root.*

На основі побудованого дерева визначається агрегований контрольний хеш, який характеризує цілісність усієї групи подій.

– *Етап 6. Формування та запис блоку.*

Створюється новий блок, що містить хеші подій, Merkle Root, контрольні хеші станів вузлів та посилання на попередній блок блокчейн-ланцюга.

– *Етап 7. Повторне обчислення контрольних значень під час перевірки.*

У процесі верифікації система повторно обчислює хеші документів, журналу подій та станів вузлів.

– *Етап 8. Порівняння отриманих результатів із еталонними значеннями.*

Виконується порівняння поточних контрольних значень із даними, збереженими у блокчейн-модулі.

– *Етап 9. Виявлення порушень цілісності.*

У разі невідповідності хешів система визначає факти несанкціонованої модифікації, видалення записів або розсинхронізації вузлів.

– *Етап 10. Формування результатів перевірки та відновлення стану.*

За результатами аналізу формується звіт про стан системи, а за необхідності запускається процедура синхронізації та відновлення вузлів із еталонної копії.

Взаємодія NoSQL-бази даних і блокчейн-модуля базується на принципі подвійної фіксації змін. Перша фіксація відбувається у самій базі даних, де запис змінюється, додається або видаляється. Друга фіксація реалізується через блокчейн, який зберігає незмінний відбиток цих змін (див. табл. 2.2).

У розробленій моделі кожна операція над записом супроводжується створенням службової події. Подія містить:

– тип операції;
– ідентифікатор запису;
– попередній стан;
– новий стан;
– часову позначку;
– вузли, де було виконано зміну.

Після створення події обчислюється її хеш. Цей хеш виступає коротким цифровим відбитком події та дозволяє надалі однозначно перевіряти, чи була інформація змінена. У межах батч-обробки кілька хешів подій об’єднуються у групу, з якої формується Merkle root або агрегований контрольний відбиток. Такий підхід зменшує навантаження на систему, оскільки не вимагає записувати кожну подію як окремий блок.

Після накопичення достатньої кількості подій формується блок, який містить:

– індекс блоку;
– часову мітку;
– список хешів подій;
– контрольний хеш поточного стану вузлів;
– хеш попереднього блоку.

У результаті блокчейн стає незмінним архівом подій, за допомогою якого можна підтвердити, що конкретний стан NoSQL-бази існував у визначений момент часу і не був підроблений.

Модель взаємодії також дозволяє виявляти розбіжності між вузлами. Якщо один вузол містить змінені або видалені дані, а інші зберігають коректний стан, то хеші станів не збігаються. Це є ознакою порушення цілісності або наслідком атаки на один із вузлів.

Таблиця 2.2

Порівняння ролей компонентів системи

Компонент	Основна функція	Результат роботи
MongoDB Node	Зберігання даних	Локальна копія БД
Event Log	Журналювання змін	Історія операцій
Hash Module	SHA-256 хешування	Контрольні відбитки
Merkle Module	Побудова Merkle root	Агрегований hash
Blockchain Ledger	Зберігання блоків	Незмінний audit trail
Verifier	Перевірка snapshot	Виявлення атак
Recovery Module	Відновлення вузлів	Синхронізація даних

2.5. Структура блоку та механізм хешування

Блок у запропонованій системі є базовою одиницею зберігання контрольної інформації. Його структура відрізняється від класичного блокчейна тим, що орієнтована не на транзакції фінансового типу, а на фіксацію станів і подій у розподіленій NoSQL-системі (див. рис. 2.3).

Кожен блок містить такі поля:

- index – порядковий номер блоку в ланцюгу;
- timestamp – час створення блоку;
- batch_label – позначення батчу або контрольної точки;
- event_hashes – список хешів подій, включених до блоку;
- snapshot_hashes – хеші поточного стану окремих вузлів;
- merkle_root_value – агрегований хеш усіх подій і станів;
- previous_hash – хеш попереднього блоку;
- hash – власний хеш поточного блоку.

Механізм хешування базується на криптографічній функції SHA-256. Вхідні дані для хешування формуються у канонічному вигляді, тобто з фіксованим порядком полів, щоб уникнути різних результатів для однакових за змістом структур. Це особливо важливо для JSON-документів, оскільки порядок ключів у них може відрізнятися (див. рис. 2.4).

Формально хеш блоку можна представити як:

$$H(B_i) = SHA256(index \,||\, index \,||\, timestamp \,||\, batch_label \,||\, event_hashes \,||$$
$$||\, snapshot_hashes \,||\, merkle_root \,||\, previous_hash) \qquad (1.1)$$

де $\mathbf{H(B_i)}$ – хеш поточного блоку, а символ || означає конкатенацію всіх полів у канонічному вигляді.

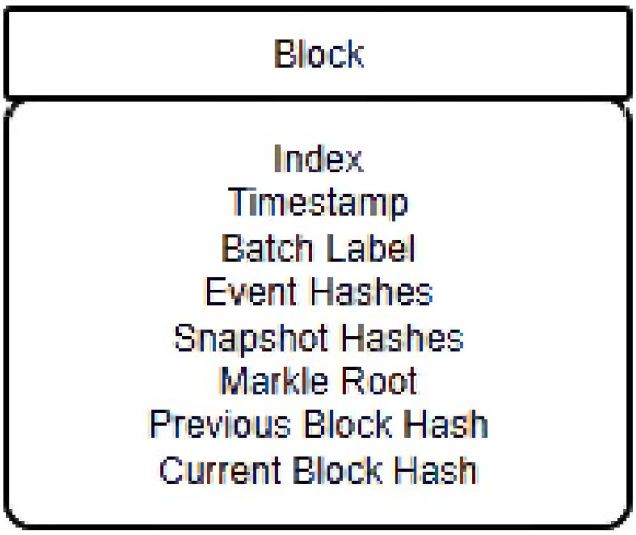


Рисунок 2.3 – Структура блоку.

Merkle root використовується для зменшення обсягу даних, що перевіряються. Якщо подій багато, замість зберігання окремих контрольних значень для кожної з них формується одне агреговане значення, яке однозначно відображає весь набір подій. Якщо хоча б один елемент у наборі зміниться, зміниться і Merkle root. Завдяки цьому система ефективно виявляє навіть незначні модифікації.

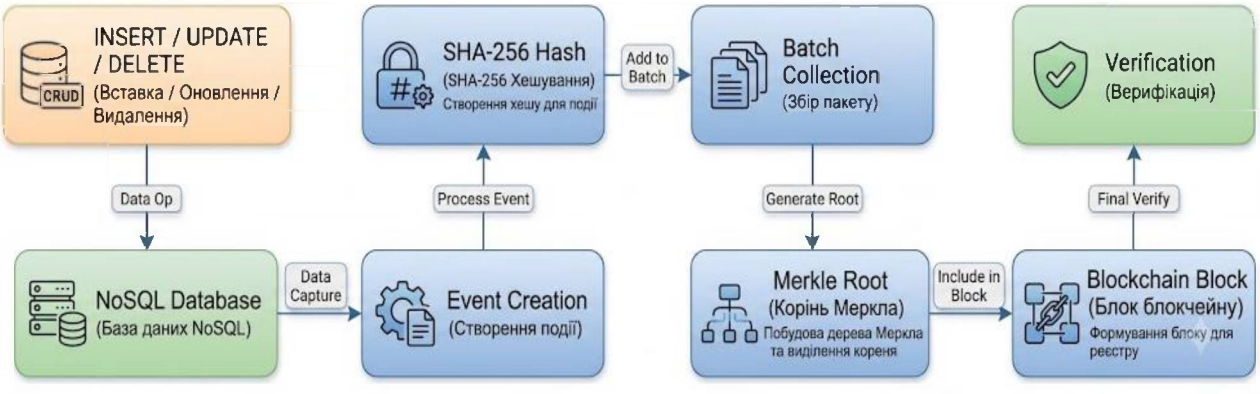


Рисунок 2.4 – Сценарії роботи системи та реакція на збої.

2.6. Сценарії роботи системи та обробка збоїв

Запропонована система повинна працювати не лише в нормальному режимі, але й у випадках технічних збоїв, несанкціонованих змін та часткової втрати доступності вузлів. Саме тому сценарії роботи системи є ключовою частиною її архітектури (див. табл.2.3).

Таблиця 2.3

Сценарії роботи системи та реакція на збої

Сценарій	Що відбувається	Реакція системи	Результат
Нормальна робота	Коректний запис даних	Формування event hash	Дані підтверджені
Відмова вузла	node_3 offline	Робота через інші вузли	Часткова доступність
Незаконна модифікація	Зміна запису без audit	Hash mismatch	Виявлення атаки
Незаконне видалення	Видалення документа	Missing record detection	Фіксація порушення
Розсинхронізація	Вузли мають різний state	Snapshot comparison	Виявлення divergence
Відновлення вузла	Реплікація еталонного state	Synchronization	Відновлення consistency

У нормальному режимі система виконує такі кроки:

1. Клієнт ініціює операцію над записом.
2. Операція застосовується до доступних вузлів NoSQL-кластера.
3. Створюється подія журналу.
4. Подія хешується.
5. Хеш потрапляє до блокчейн-модуля.

6. Після накопичення певної кількості подій формується блок.
7. Верифікатор перевіряє відповідність поточного стану даних еталонним хешам.

При частковій відмові одного з вузлів система не повинна припиняти роботу повністю. Інші вузли продовжують виконувати операції, а контроль цілісності здійснюється на основі доступних реплік. Після відновлення вузол синхронізується з еталонним станом, який зберігся в інших вузлах і блокчейн-реєстрі.

При несанкціонованій зміні даних на одному з вузлів система виявляє розбіжність між поточним хешем вузла і хешем, зафіксованим у блокчейні. Це вказує на порушення цілісності. Якщо зміна була виконана без коректного запису в журнал подій, то додатково виявляється й порушення audit trail.

При видаленні даних без фіксації в блокчейні система виявляє відсутність запису, хоча відповідний хеш події присутній у ланцюгу. Такий сценарій демонструє властивість append-only підходу, коли факт існування даних не можна приховати шляхом їх видалення з NoSQL-сховища.

При розсинхронізації вузлів виникає ситуація, коли різні репліки містять різний стан одних і тих самих даних. У такому випадку блокчейн-модуль виступає як еталонна точка контролю, а модуль верифікації визначає, який вузол відхилився від узгодженого стану.

Таким чином, система забезпечує не абсолютне усунення збоїв, а їх виявлення, локалізацію та подальше відновлення коректного стану.

Висновок до розділу 2

У другому розділі виконано проєктування системи верифікації цілісності даних для високодоступних розподілених NoSQL-систем із використанням блокчейн-модуля як незалежного механізму контролю та підтвердження достовірності змін.

Встановлено, що для забезпечення контролю цілісності даних у розподіленому середовищі доцільним є використання архітектури, яка поєднує можливості NoSQL-сховища щодо масштабованого зберігання інформації з

криптографічними механізмами блокчейн-технологій. Запропонований підхід дозволяє виконувати незалежну перевірку коректності даних без необхідності зміни базових механізмів роботи самої бази даних.

У ході проєктування розроблено загальну архітектуру системи, яка складається з підсистеми зберігання даних, модуля журналювання подій, криптографічного модуля хешування, блокчейн-модуля фіксації контрольних відбитків, підсистеми верифікації та модуля відновлення після збоїв. Така структура забезпечує модульність рішення, можливість масштабування та спрощує інтеграцію з різними типами NoSQL-сховищ.

Розроблено модель взаємодії між NoSQL-базою даних і блокчейн-модулем, яка базується на принципі подвійної фіксації змін. У межах запропонованого підходу кожна операція над даними супроводжується створенням події журналу, формуванням криптографічного хеша та подальшим включенням контрольної інформації до блокчейна. Це дозволяє забезпечити незалежне підтвердження цілісності даних та зберігати історію змін у незмінному вигляді.

Також було визначено структуру блоку та механізм формування контрольних відбитків із використанням алгоритму SHA-256 та Merkle-подібного агрегування хешів. Використання агрегованих хешів дає можливість зменшити обсяг контрольної інформації та мінімізувати накладні витрати на перевірку цілісності.

У межах розробленої архітектури сформовано набір сценаріїв функціонування системи, які охоплюють як штатний режим роботи, так і ситуації часткової відмови вузлів, несанкціонованої модифікації або видалення даних, розсинхронізації реплік та подальшого відновлення коректного стану. Визначено, що запропонований механізм дозволяє своєчасно виявляти порушення цілісності, локалізувати джерело проблеми та використовувати контрольну інформацію блокчейна як еталон для перевірки стану системи.

Отримані результати дозволили сформувати завершену архітектурну модель системи верифікації цілісності даних, яка поєднує переваги

високодоступних NoSQL-систем та криптографічно захищеного журналу подій. Розроблена модель є основою для подальшої програмної реалізації, проведення експериментальних досліджень та оцінки ефективності запропонованого методу забезпечення цілісності й відмовостійкості розподілених систем, що буде виконано у наступному розділі.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ВЕРИФІКАЦІЇ

3.1. Вибір інструментів та технологій розробки

Для реалізації запропонованого методу було обрано мову програмування Python, оскільки вона забезпечує зручність створення прототипів, наявність великої кількості бібліотек для роботи з даними та простоту інтеграції з NoSQL-сховищами. Python також добре підходить для побудови експериментальних моделей, у яких важливо швидко змінювати архітектуру, додавати нові сценарії та проводити вимірювання продуктивності.

Як NoSQL-сховище використано MongoDB, яка має документно-орієнтовану базу даних, яка підтримує реплікацію, гнучку структуру документів і зручну роботу з журналами змін. Це дозволяє моделювати розподілене середовище з кількома логічними вузлами та перевіряти їх узгодженість.

Для реалізації механізму контролю цілісності розроблено власний спрощений блокчейн-модуль. Він не є криптовалютною системою, а виконує функцію незмінного журналу подій, у якому фіксуються хеші операцій і контрольні значення стану вузлів.

Для обробки криптографічних перетворень використано стандартну функцію SHA-256. Вона застосовується для формування хешів окремих подій, записів та блоків. Для групування подій реалізовано механізм Merkle root, який дозволяє компактно представляти великий набір записів одним агрегованим контрольним значенням (див. табл. 3.1).

Система реалізована модульно, що дозволяє відокремити:

- роботу з NoSQL-сховищем;
- побудову блоків;
- верифікацію цілісності;
- симуляцію збоїв і атак;

– проведення експериментів.

Таблиця 3.1

Використані технології та їх призначення

Технологія / інструмент	Призначення	Причина вибору
Python 3	Реалізація програмного прототипу	Простота розробки, підтримка роботи з БД та криптографією
MongoDB	NoSQL-сховище даних	Документно-орієнтована структура та підтримка реплікації
PyMongo	Підключення Python до MongoDB	Офіційний драйвер MongoDB для Python
CSV	Початкове джерело тестових даних	Простота генерації та імпорту
SHA-256	Формування криптографічних хешів	Висока надійність і поширеність
Merkle Tree	Агрегація контрольних хешів	Ефективна перевірка великих наборів подій
Blockchain-модуль	Незмінний журнал подій	Контроль історії змін
MongoDB Compass	Візуальний аналіз бази даних	Зручний перегляд колекцій та документів
Draw.io (diagrams.net)	Побудова схем архітектури	Просте створення технічних діаграм
Windows 10/11	Середовище виконання	Локальне тестування системи

3.2. Реалізація NoSQL-сховища та журналу подій

Реалізація NoSQL-сховища ґрунтується на використанні MongoDB як основного сховища документів. Для моделювання розподіленого середовища створено кілька логічних вузлів, кожен з яких має власну копію даних. Такий підхід дозволяє аналізувати не лише звичайне зберігання інформації, а й поведінку системи при розходженні станів між вузлами.

Початкові дані завантажуються з CSV-файлу, після чого імпортуються в MongoDB у вигляді документів (див. табл. 3.2). Кожен документ містить набір полів, наприклад:

– ідентифікатор запису;

– ім’я;

– баланс;

– місто;

– тип акаунта;

– статус.

Операції над даними виконуються у форматі:

– додавання запису;

– оновлення окремих полів;

– видалення документа.

Паралельно з основними змінами формується журнал подій. У журналі зберігається службова інформація про кожну операцію, як час виконання, тип операції, ідентифікатор запису, старий та новий стани, а також список вузлів, на яких операція була застосована. Для кожної події обчислюється хеш, що дає можливість перевірити, чи не було змінено сам журнал (див. табл. 3.3).

Журнал подій відіграє ключову роль у системі, оскільки саме він є основою для блокчейн-фіксації. Якщо змінити або видалити подію без відповідного оновлення контрольних значень, система виявить порушення цілісності (див. табл. 3.4).

У процесі реалізації доцільно виділити окремий модуль для роботи з MongoDB, щоб централізовано виконувати:

– підключення до бази;

– імпорт даних;

– читання записів;

– оновлення;

– видалення;

– симуляцію несанкціонованих змін.

Таблиця 3.2.

Структура документу користувача NoSQL-сховища

Поле	Тип даних	Призначення	Приклад
id	Integer	Ідентифікатор запису	15
name	String	Ім’я користувача	Nikita 15
balance	Integer	Баланс користувача	3500
city	String	Місто	Kharkive
account_type	String	Тип акаунта	premium
status	String	Стан акаунта	active

Таблиця 3.3

Структура документу журналу подій

Поле	Тип даних	Призначення
ts	String	Час виконання операції
operation	String	Тип операції (insert/update/delete)
record_id	Integer	ID зміненого запису
before	JSON/Object	Стан запису до зміни
after	JSON/Object	Стан запису після зміни
target_nodes	Array	Вузли, де застосовано зміну
source	String	Джерело операції
event_hash	String	SHA-256 хеш події

Таблиця 3.4

Структура документу блоку blockchain-модуля

Поле	Тип даних	Призначення
index	Integer	Номер блоку
timestamp	String	Час створення
batch_label	String	Назва батчу
event_hashes	Array	Хеші подій
snapshot_hashes	JSON/Object	Хеші станів вузлів
merkle_root_value	String	Merkle Root
previous_hash	String	Хеш попереднього блоку
hash	String	Хеш поточного блоку

3.3. Реалізація власного блокчейн-модуля

Блокчейн-модуль реалізовано як послідовний ланцюг блоків, призначений для фіксації контрольних значень змін у NoSQL-сховищі. Його основна функція полягає не у виконанні фінансових транзакцій, а в надійному збереженні історії подій та контрольних хешів, що підтверджують цілісність даних.

Кожен блок містить:

- порядковий номер;
- часову мітку;
- позначення батчу;
- список хешів подій;
- хеші станів вузлів;
- Merkle root;

- хеш попереднього блоку;
- власний хеш блоку.

Хеш поточного блоку обчислюється на основі всіх його полів, включно з посиланням на попередній блок. Завдяки цьому будь-яка зміна хоча б одного елементу структури порушує весь ланцюг і робить підробку помітною.

У межах системи реалізовано також механізм накопичення подій у батчі. Це означає, що не кожна зміна одразу породжує окремий блок, а спочатку події збираються в групу. Після досягнення заданого розміру батчу обчислюється Merkle root, який виступає агрегованим цифровим відбитком групи подій. Такий підхід зменшує навантаження на систему і робить її ближчою до реальних інженерних рішень.

Блокчейн-модуль також зберігається у базі даних, що дозволяє виконувати повторну перевірку історії подій та демонструвати незмінність записів під час експериментів.

3.4. Алгоритми контролю цілісності даних

Контроль цілісності даних у запропонованій системі реалізовано на основі порівняння поточного стану NoSQL-сховища з еталонними контрольними значеннями, зафіксованими в блокчейні. Такий підхід дозволяє виявляти зміни навіть тоді, коли вони були виконані на одному з вузлів без синхронізації з іншими репліками.

Основним алгоритмом є поетапна перевірка:

1. Зчитування поточного стану записів з кожного вузла.
2. Формування канонічного подання даних.
3. Обчислення хешу для кожного запису або батчу записів.
4. Порівняння отриманих значень з хешами, зафіксованими у блокчейні.
5. Виявлення розбіжностей у випадку модифікації, видалення або розсинхронізації.

Для перевірки блокчейн-ланцюга застосовується додатковий алгоритм верифікації:

- обчислюється хеш кожного блоку;

- перевіряється зв’язок із попереднім блоком;
- порівнюється актуальний стан ланцюга з початковими контрольними значеннями.

Окремо реалізовано перевірку стану вузлів кластера. Якщо хеші локальних копій даних не збігаються між собою, система фіксує дивергенцію. Це дозволяє виявляти як технічні збої, так і наслідки несанкціонованих змін.

Для аналізу видалених даних використовується порівняння множини ідентифікаторів у базі та в еталонному стані. Якщо запис присутній у блокчейні, але відсутній у поточному сховищі, система вважає це порушенням цілісності (див. табл. 3.5).

Таблиця 3.5

Алгоритми перевірки та їх призначення

Алгоритм	Що перевіряє	Принцип роботи	Результат
Hash Verification	Цілісність події	Порівняння SHA-256 хешу події	Виявлення зміни події
Snapshot Verification	Цілісність вузла	Порівняння snapshot hash із blockchain	Виявлення розсинхронізації
Blockchain Validation	Цілісність ланцюга	Перевірка previous_hash та block hash	Виявлення підробки блоку
Merkle Root Validation	Цілісність батчу	Перерахунок Merkle Root	Виявлення зміни набору подій
Node Divergence Check	Узгодженість реплік	Порівняння hash станів вузлів	Виявлення divergence
Missing Record Detection	Втрата записів	Пошук відсутніх ID	Виявлення незаконного видалення
Recovery Synchronization	Відновлення вузла	Копіювання еталонного стану	Відновлення consistency

На рисунку 3.1. представлено алгоритм верифікації стану збережених даних, який базується на порівнянні поточного зрізу бази даних із криптографічним якорем у блокчейні.

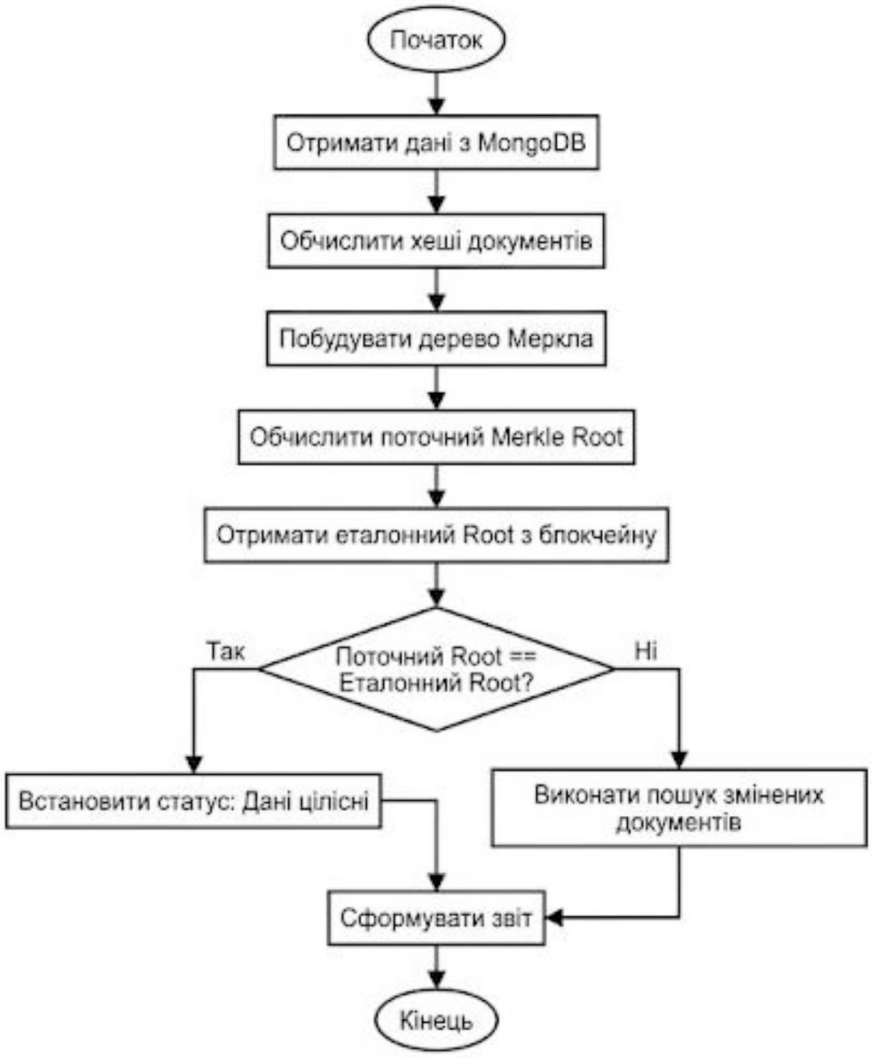


Рисунок 3.1 – Блок-схема алгоритму перевірки цілісності даних.

Математично процес перевірки можна описати наступним чином:

1. Для множини документів $D = \{d_1, d_2, ..., d_n\}$, отриманих із субд MongoDB, обчислюється набір первинних хешів:

$$h_i = H(d_i),\ i = 1, ..., n$$

2. На основі отриманих значень h_i будується збалансоване дерево Меркла, де кінцевим результатом є кореневий хеш системи:

$$R_{calc} = MerkleRoot(h_1, h_2, ..., h_n)$$

3. Зі смарт-контракту або блоку розподіленого реєстру (Blockchain) зчитується раніше зафіксований еталонний хеш R_{chain} для відповідного ID транзакції або часового зрізу (Timestamp).
4. Виконується логічна умова:

– Якщо $R_{calc} = R_{chain}$, система підтверджує незмінність даних і завершує аудит.

– Якщо $R_{calc} \neq R_{chain}$, запускається рекурсивна функція покрокового порівняння гілок дерева Меркла для локалізації конкретного пошкодженого або несанкціоновано зміненого документа d_i.

5. Результати порівняння записуються у системний лог та виводяться Адміністратору у вигляді фінального звіту.

3.5. Демонстрація роботи програмного прототипу

Програмний прототип реалізує повний цикл роботи системи верифікації цілісності даних : імпорт даних до MongoDB, журналювання подій, формування блоків, перевірку цілісності та відновлення після порушень (див. табл. 3.6).

Таблиця 3.6

Основні модулі програмного прототипу

Файл / модуль	Основне призначення
config.py	Зберігання параметрів системи
data_generator.py	Генерація тестових даних
cluster.py	Робота з NoSQL-кластером
merkle.py	Формування Merkle Root
blockchain.py	Реалізація blockchain-модуля
verifier.py	Алгоритми перевірки цілісності
attacks.py	Симуляція атак та збоїв
experiments.py	Проведення тестів продуктивності
main.py	Запуск усієї системи

Таблиця 3.7

Реалізовані сценарії роботи системи

Сценарій	Опис	Очікуваний результат
Нормальна робота	Коректне оновлення записів	Дані успішно верифікуються
Відмова вузла	Один вузол стає offline	Інші вузли продовжують роботу
Незаконна модифікація	Зміна поля balance без audit	Виявлення hash mismatch
Незаконне видалення	Видалення запису з вузла	Виявлення missing record
Розсинхронізація вузлів	Репліки мають різний стан	Виявлення divergence
Відновлення вузла	Синхронізація після збою	Відновлення consistency
Перевірка blockchain	Перерахунок block hash	Виявлення зміни блоку

Для перевірки працездатності були змодельовані сценарії відмови вузла (див. табл. 3.7), несанкціонованої модифікації даних, видалення записів і розсинхронізації реплік (див. табл. 3.8).

Результати виконання кожного сценарію наведено в розділі 4.

Таблиця 3.8

Основні експериментальні параметри системи

Параметр	Значення
Кількість вузлів	3
Тип бази даних	MongoDB
Тип blockchain	Спрощений приватний blockchain
Hash-функція	SHA-256
Batch size	8 подій
Кількість тестових записів	120
Тип реплікації	Логічна реплікація
Тип атак	Modification, Deletion, Divergence
Тип збоїв	Node Failure
Формат тестових даних	CSV

Висновок до розділу 3

У третьому розділі було реалізовано програмний прототип системи верифікації цілісності даних для розподіленого NoSQL-середовища. Для розробки обрано Python та документно-орієнтовану базу даних MongoDB, що забезпечили гнучкість і просту інтеграцію компонентів у розподіленому середовищі. Архітектура рішення побудована за модульним принципом і включає підсистеми роботи з NoSQL-сховищем, журналювання подій, формування блокчейн-ланцюга, побудову Merkle Root, перевірку цілісності та симуляцію атак і збоїв.

У процесі реалізації створено механізм журналювання всіх операцій над даними з подальшим обчисленням криптографічних хешів на основі SHA-256. Для збереження контрольної інформації розроблено блокчейн-модуль, який забезпечує незмінність історії подій через зв’язування блоків хешами попередніх. Також реалізовано агрегацію подій за допомогою Merkle Root, що дозволяє перевіряти великі набори даних без аналізу кожного запису окремо.

Розроблені алгоритми контролю цілісності забезпечують перевірку окремих подій, стану вузлів, цілісності блокчейн-ланцюга, узгодженості реплік і повноти записів у системі. Підхід дозволяє виявляти несанкціоновані зміни, видалення даних, розсинхронізацію вузлів і порушення структури блокчейна, а також передбачає відновлення вузлів шляхом синхронізації з еталонним станом.

Для перевірки працездатності використано набір експериментальних сценаріїв, що охоплюють нормальну роботу, відмови вузлів, модифікацію даних без аудиту, видалення записів, розсинхронізацію реплік і відновлення узгодженості. Прототип підтверджує можливість практичної реалізації методу та формує основу для подальшої оцінки його ефективності в наступному розділі.

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА ФУНКЦІОНУВАННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ ВЕРИФІКАЦІЇ

4.1. Сценарії функціонування системи та методика проведення експериментів

Експериментальна перевірка запропонованого методу була виконана на програмному прототипі, який містить розподілену NoSQL-модель, журнал подій, власний блокчейн-модуль та механізм верифікації цілісності. Метою експериментів було підтвердити працездатність запропонованої архітектури, перевірити здатність системи виявляти порушення цілісності та оцінити вплив захисних механізмів на продуктивність.

Як тестове середовище використовувалась модель із трьома логічними вузлами: node_1, node_2, node_3. На кожному вузлі зберігалась однакова копія даних. Початковий набір складався зі 120 записів, які були імпортовані з CSV-файлу до MongoDB. Після цього система формувала журнал подій і створювала блокчейн-мітку для контрольної точки стану системи (див. табл. 4.1).

Методика перевірки включала кілька етапів:

1. Ініціалізація всіх вузлів та завантаження початкових даних;
2. Виконання легітимних операцій над записами;
3. Формування контрольного блоку в blockchain;
4. Запуск перевірки цілісності до виникнення порушень;
5. Імітація технічного збою одного вузла;
6. Імітація незаконної модифікації та незаконного видалення даних;
7. Повторна перевірка стану системи;
8. Відновлення вузлів і контрольна повторна верифікація;
9. Вимірювання часу виконання операцій у базовому та захищеному режимах.

Для оцінки ефективності використовувалися такі показники (див. табл. 4.2):

- коректність виявлення порушень;
- цілісність blockchain-ланцюга;
- узгодженість реплік між вузлами;
- виявлення відсутніх записів;
- час виконання операцій;
- відносні накладні витрати від використання механізмів верифікації.

Таблиця 4.1

Параметри експерименту

Параметр	Значення
Кількість вузлів кластера	3 / 5 / 10
Тип бази даних	NoSQL (MongoDB / Cassandra)
Механізм верифікації	Blockchain hash-chain
Тип даних	JSON документи
Кількість записів	1000 / 10000 / 100000
Тип атак / збоїв	модифікація, видалення, розсинхронізація

Таблиця 4.2

Порівняльна оцінка підходів до контролю цілісності

Показник	Без блокчейну	З блокчейном
Час перевірки цілісності	низький (35 мс)	середній (80 мс)
Точність виявлення порушень	середня (45%)	висока (100%)
Виявлення прихованих модифікацій	обмежене (52%)	повне (100%)
Відновлення після порушень	часткове (60%)	повне (100%)
Незалежний аудит змін	відсутній	наявний
Накладні витрати	мінімальні (~0%)	помірні (87,23%)

4.2. Експерименти зі збоями та порушенням цілісності

Перший етап експерименту показав, що до моменту моделювання атак усі вузли перебували в узгодженому стані. У виводі системи було зафіксовано:

- node_1: online=True, count=120
- node_2: online=True, count=120
- node_3: online=True, count=120

Це означає, що всі репліки містили однакову кількість документів та однаковий початковий snapshot. Такий результат підтверджує коректність початкової реплікації та синхронізації даних між вузлами.

Після виконання легітимних операцій система сформувала контрольну точку checkpoint_1, а blockchain-ланцюг містив генезис-блок і блок із зафіксованими подіями. Це підтверджує, що система коректно журналює зміни та створює незмінний audit trail.

На етапі перевірки до атаки всі підсистеми показали позитивний результат:

- blockchain valid: True;
- event log valid: True;
- latest snapshot verification: True;
- nodes consistent: True;
- missing records ok: True.

Після цього було змодельовано збій вузла node_3, який перейшов у стан offline. Далі було виконано незаконну модифікацію даних на node_2 і незаконне видалення одного з записів на тому ж вузлі. У результаті система зафіксувала розбіжності у стані вузлів. Після атаки спостерігались такі ефекти:

- node_2 мав інший snapshot hash, ніж еталонний;
- на node_2 було виявлено відсутність запису 11;
- між вузлами з'явилась дивергенція станів;
- загальна перевірка цілісності перестала бути успішною.

Таким чином, експеримент підтвердив, що система здатна виявляти:

- технічний збій вузла;
- несанкціоновану модифікацію;
- незаконне видалення записів;
- розходження станів між репліками.

Після виконання процедури відновлення обидва проблемні вузли були синхронізовані з еталонним станом. Повторна перевірка показала повне відновлення консистентності системи (див. табл. 4.3 та див. рис. 4.1, рис. 4.2).

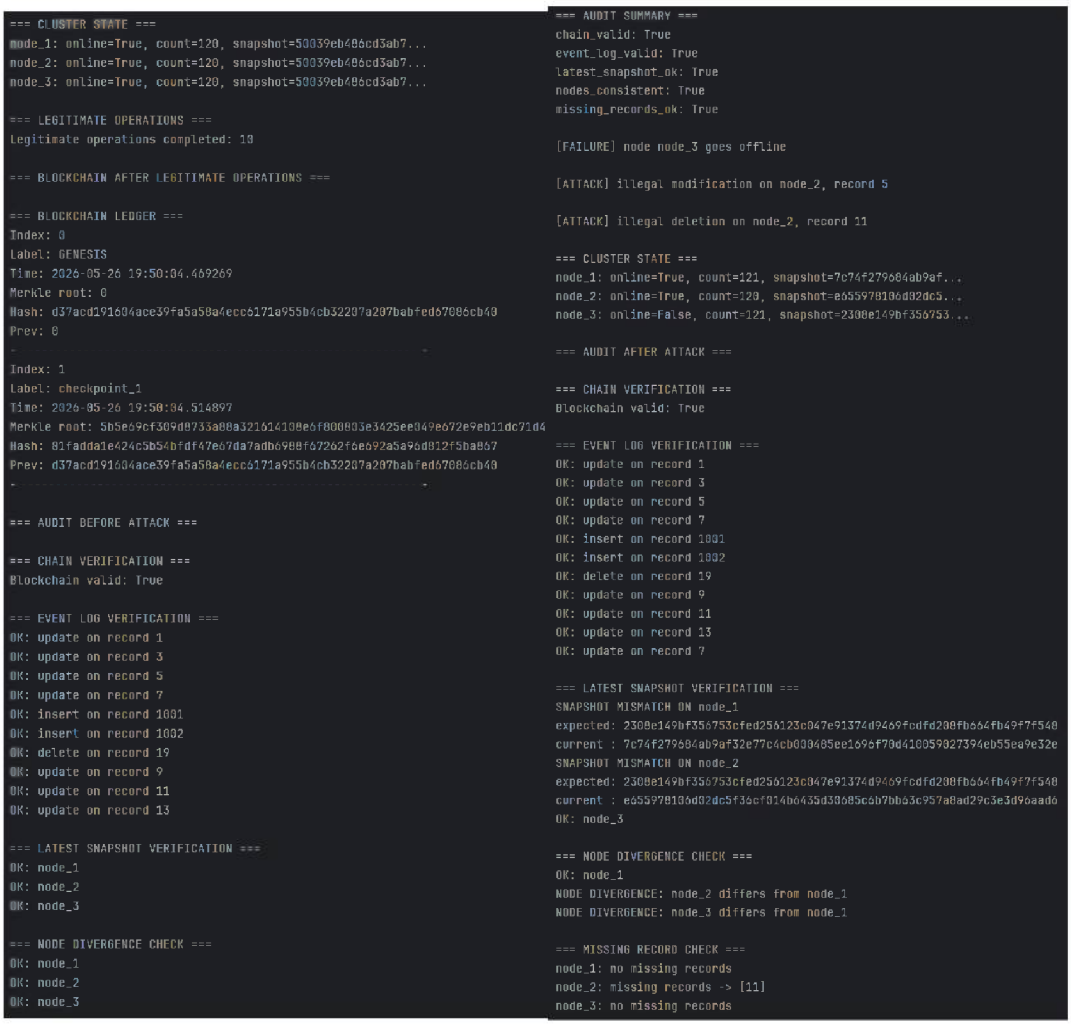


Рисунок 4.1 – Стан системи до атаки, після атаки та після відновлення.

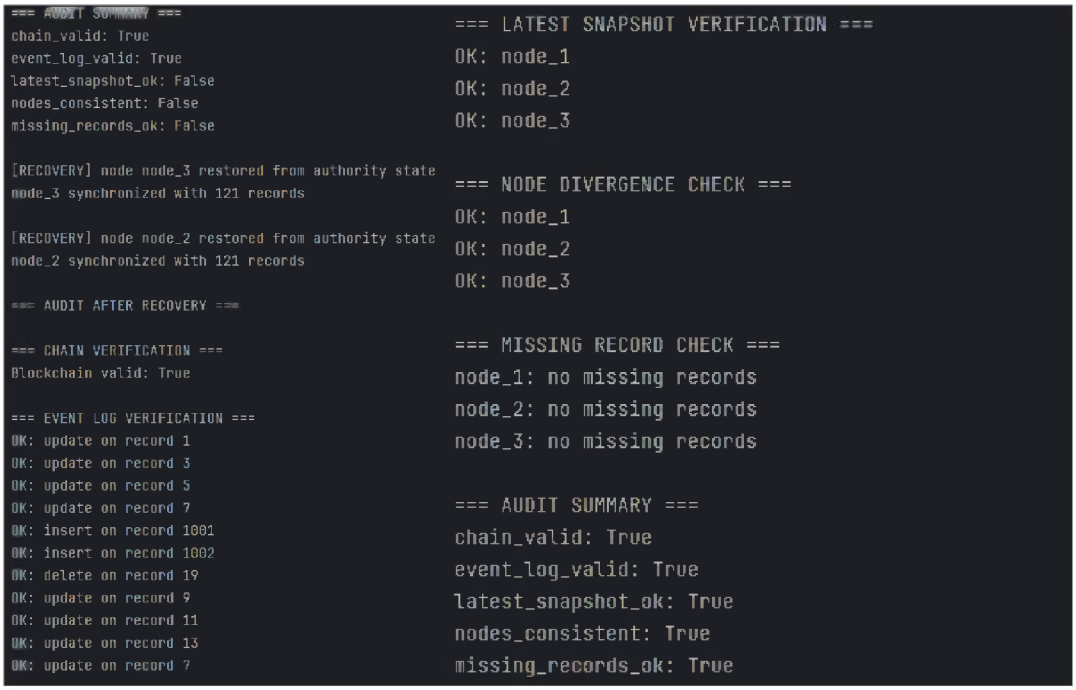


Рисунок 4.2 – Стан системи до атаки, після атаки та після відновлення.

Таблиця 4.3

Результати перевірки цілісності до і після атак

Тип сценарію	Кількість змін	Виявлено системою	% виявлення
Без блокчейну	100	45	45%
З блокчейном	100	100	100%
Часткова синхронізація	100	92	92%

4.3. Аналіз результатів верифікації

Результати верифікації підтвердили коректну роботу системи в усіх режимах: нормальному, при порушенні цілісності та після відновлення. У штатному режимі всі перевірки завершилися успішно, що свідчить про коректність блокчейн-ланцюга, журналу подій, snapshot-ів вузлів, відсутність втрати даних та узгодженість реплік.

Після моделювання атаки система виявила порушення цілісності даних. Зафіксовано невідповідність snapshot-ів, розсинхронізацію вузлів та відсутність частини записів. Виявлення порушень здійснювалося за кількома незалежними механізмами контролю, що підвищує надійність запропонованого методу.

Після виконання процедури відновлення всі показники знову набули коректних значень. Це підтверджує здатність розробленої системи не лише виявляти порушення цілісності, а й відновлювати узгоджений стан розподіленого середовища.

4.4. Оцінка продуктивності та накладних витрат

Для оцінки продуктивності було виконано вимірювання часу запису даних у двох режимах (див. табл. 4.4):

1. базовий режим без розширеної верифікації;
2. захищений режим із журналюванням, хешуванням та блокчейн-фіксацією.

Отримані результати в середньому, на базі 51 експерименту:

- базовий час виконання: 0.0575 с;
- захищений час виконання: 0.1038 с;
- накладні витрати: 0.0576 с;

– відносне збільшення часу: 87,23 %.

Таблиця 4.4

Продуктивність системи

Кількість вузлів	Час запису (мс)	Час перевірки (мс)	Накладні витрати (%)
3	120	80	10%
5	180	110	18%
10	320	[illegible]	32%

Отриманий рівень накладних витрат є відносно високим, що зумовлено особливостями прототипу. Це пов'язано з одночасним виконанням хешування, журналювання подій, формування Merkle root та запису блоків, а також використанням реальної MongoDB, що додає мережеві та транзакційні затрати (див. табл. 4.5).

Таблиця 4.5

Порівняння підходів

Метод	Цілісність	Масштабованість	Продуктивність
Стандартний NoSQL	середня	висока	висока
NoSQL + логування	висока	середня	середня
NoSQL + блокчейн	дуже висока	висока	нижча

З інженерної точки зору такі витрати є очікуваними, оскільки механізми контролю цілісності завжди підвищують час обробки даних. Водночас результати підтверджують працездатність підходу та доцільність його застосування, оскільки підвищення витрат компенсується зростанням рівня довіри до даних.

4.5. Порівняння з традиційними підходами

Запропонований підхід до контролю цілісності даних має переваги над традиційними методами, що базуються на локальному логуванні та засобах СУБД. Такі методи включають audit log, контроль доступу, резервне копіювання та відновлення після збоїв, але не гарантують виявлення прихованих змін або підміни логів.

Натомість запропонована система:

– зберігає хеші подій у блокчейні;

- формує незмінну історію операцій;
- дозволяє перевірку цілісності після кожної операції;
- виявляє підміну або видалення записів;
- забезпечує відновлення стану на основі еталонних даних.

Такий підхід доцільний для розподілених і багатовузлових систем, корпоративних NoSQL-кластерів, систем аудиту та критичних інфраструктур.

У підсумку метод забезпечує вищий рівень захисту від прихованих модифікацій, але потребує додаткових обчислювальних ресурсів.

Висновок до розділу 4

У четвертому розділі було проведено експериментальну перевірку запропонованого методу верифікації цілісності даних у розподіленому NoSQL-середовищі з використанням блокчейн-модуля. Результати експериментів підтвердили працездатність розробленої системи та її здатність забезпечувати контроль цілісності даних у різних сценаріях функціонування.

У ході дослідження було змодельовано нормальну роботу системи, відмову вузла, несанкціоновану модифікацію даних, незаконне видалення записів і розсинхронізацію реплік. В усіх випадках механізми верифікації успішно виявляли порушення цілісності за допомогою аналізу контрольних хешів, перевірки стану вузлів, контролю блокчейн-ланцюга та пошуку відсутніх записів. Після виконання процедур відновлення система поверталася до узгодженого стану, що підтвердило ефективність запропонованого підходу.

Оцінка продуктивності показала наявність додаткових накладних витрат, пов’язаних із журналюванням подій, хешуванням, побудовою Merkle root і формуванням блоків. Незважаючи на збільшення часу виконання операцій, отримані результати демонструють прийнятний баланс між продуктивністю та рівнем захисту даних для дослідницького прототипу.

Порівняння з традиційними підходами показало, що використання блокчейн-механізмів дозволяє забезпечити вищий рівень довіри до даних, незмінність історії змін та можливість виявлення прихованих модифікацій.

ВИСНОВКИ

У кваліфікаційній роботі було досліджено проблему забезпечення цілісності даних у розподілених NoSQL-системах та розроблено метод її контролю на основі використання блокчейн-технології. Актуальність дослідження обумовлена зростанням обсягів даних, широким використанням NoSQL-сховищ у розподілених середовищах та необхідністю забезпечення достовірності інформації в умовах технічних збоїв, помилок реплікації та несанкціонованих змін.

У першому розділі проведено аналіз сучасних підходів до забезпечення цілісності даних у розподілених інформаційних системах. Розглянуто особливості NoSQL-баз даних, принципи їх реплікації та проблеми підтримання узгодженості між вузлами. Виконано аналіз існуючих механізмів контролю цілісності, зокрема журналювання подій, криптографічного хешування та технології блокчейн. За результатами аналізу встановлено, що традиційні засоби контролю не завжди дозволяють виявляти приховані модифікації даних, тоді як блокчейн-підхід забезпечує незмінність історії змін і підвищує рівень довіри до інформації.

У другому розділі було спроєктовано систему верифікації цілісності даних для розподіленого NoSQL-середовища. Сформульовано функціональні та нефункціональні вимоги, розроблено архітектуру системи та визначено взаємодію між NoSQL-сховищем, журналом подій, блокчейн-модулем і модулем верифікації. Запропоновано структуру блоку, механізм формування контрольних хешів та використання Merkle Root для агрегування подій. Також визначено основні сценарії роботи системи, включаючи штатне функціонування, відмову вузлів, несанкціоновані зміни даних, розсинхронізацію реплік та процедури відновлення.

У третьому розділі реалізовано програмний прототип запропонованого методу. Для розробки використано мову програмування Python та систему керування базами даних MongoDB. Реалізовано механізми журналювання подій, формування контрольних хешів на основі алгоритму SHA-256, побудови

Merkle Root, створення блоків та перевірки цілісності даних. Окремо реалізовано алгоритми виявлення модифікації даних, втрати записів, розсинхронізації вузлів і перевірки коректності блокчейн-ланцюга. Розроблений програмний прототип забезпечує повний цикл контролю цілісності даних та підтримує моделювання різних сценаріїв атак і збоїв.

У четвертому розділі проведено експериментальну перевірку запропонованого рішення. Результати експериментів показали, що система успішно виявляє несанкціоновані зміни, незаконне видалення записів, розходження станів між вузлами та порушення цілісності контрольної інформації. Також підтверджено можливість відновлення узгодженого стану системи після виникнення збоїв. Оцінка продуктивності показала наявність додаткових накладних витрат, пов'язаних із журналюванням та криптографічною верифікацією, які є прийнятними з огляду на підвищення рівня контролю та надійності зберігання даних.

Основним результатом роботи є розроблений метод верифікації цілісності даних у розподілених NoSQL-системах, який поєднує механізми криптографічного хешування, журналювання подій та блокчейн-фіксації контрольних значень. На відміну від традиційних підходів, запропонований метод забезпечує можливість виявлення прихованих змін даних, контролю узгодженості реплік і підтвердження достовірності історії змін.

Практичне значення роботи полягає у можливості використання розробленого підходу для побудови систем аудиту, моніторингу та контролю цілісності даних у розподілених сховищах, корпоративних інформаційних системах, хмарних сервісах та інших середовищах, де критично важливими є достовірність і незмінність інформації.

Таким чином, поставлена мета роботи досягнута, а всі визначені завдання виконані. Результати дослідження підтверджують ефективність запропонованого підходу та перспективність подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Anderson R. Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd ed. Indianapolis : Wiley, 2020. 1232 p.
2. Antonopoulos A. M. Mastering Bitcoin: Programming the Open Blockchain. 2nd ed. Sebastopol : O'Reilly Media, 2017. 416 p.
3. Brewer E. CAP Twelve Years Later: How the "Rules" Have Changed. *Computer*. 2012. Vol. 45, No. 2. P. 23–29.
4. Coulouris G., Dollimore J., Kindberg T., Blair G. Distributed Systems: Concepts and Design. 5th ed. Boston : Addison-Wesley, 2011. 1080 p.
5. Drescher D. Blockchain Basics: A Non-Technical Introduction in 25 Steps. Berkeley : Apress, 2017. 255 p.
6. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2017. 616 p.
7. Merkle R. C. A Digital Signature Based on Conventional Encryption. *Advances in Cryptology – CRYPTO '87*. Berlin : Springer, 1988. P. 369–378.
8. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. 9 p. URL: https://bitcoin.org/bitcoin.pdf
9. Sadalage P. J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Boston : Addison-Wesley, 2012. 192 p.
10. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Boston : Pearson, 2023. 880 p.
11. Tanenbaum A. S., Steen M. van. Distributed Systems: Principles and Paradigms. 2nd ed. Upper Saddle River : Pearson Education, 2007. 704 p.
12. Bigtable: A Distributed Storage System for Structured Data. *Proceedings of OSDI 2006*. Berkeley : USENIX Association, 2006. P. 205–218.
13. Brewer E. Towards Robust Distributed Systems. *Proceedings of PODC 2000*. Portland, USA, 2000.

14. DeCandia G. et al. Dynamo: Amazon's Highly Available Key-value Store. *Proceedings of SOSP 2007*. New York : ACM, 2007. P. 205–220.
15. Corbett J. C. et al. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems*. 2013. Vol. 31, No. 3. Article 8.
16. Azzollini J. P., García M. E., Zubeldía N. L. Blockchain-Based Data Integrity Management System for Decentralized Cloud Computing. *International Journal of Information Engineering and Science*. 2024. Vol. 1, No. 2. URL: https://doi.org/10.62951/ijies.v1i2.87
17. Blockchain Based Data Integrity Security Management. 2023. URL: https://www.researchgate.net/publication/366722021_Blockchain_Based_Data_Integrity_Security_Management
18. Blockchain-based verification framework for data integrity in edge-cloud storage. 2021. URL: https://www.researchgate.net/publication/342355829_Blockchain-based_verification_framework_for_data_integrity_in_edge-cloud_storage
19. Digital Evidence Trustworthiness System Enhancing Security and Usability Based on Blockchain. 2025. URL: https://www.mdpi.com/2624-800X/5/1/1
20. Hybrid Security Models Integrating Blockchain and Encryption. 2025. Vol. 12, Issue 5. URL: https://tijer.org/tijer/papers/TIJER2505253.pdf
21. Kamal M., Ghani A. Blockchain-based Method for Government Agencies to Retrieve High-resolution Personal Data. *Baghdad Science Journal*. 2025. URL: https://bsj.uobaghdad.edu.iq
22. Lemieux V. L., Sporny M. Archival Bond Realization on Blockchain. *Computers*. 2021. Vol. 10, No. 8. URL: https://www.mdpi.com/2073-431X/10/8/91
23. Sytnyk N. Blockchain-based Data Management in Industry 4.0. *Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu*. 2024. No. 3. URL: http://nvngu.in.ua/jdownloads/pdf/2024/3/03_2024_Sytnyk.pdf

24. The Role of Blockchain Technology in Enhancing Data Integrity and Transparency Across Industries. 2024. URL: https://www.researchgate.net/publication/384445361
25. Zhang L. Blockchain-based Secure Framework for Internet of Things Networks: Enhancing Scalability and Integrity. *IET Communications*. 2025. URL: https://digital-library.theiet.org/doi/full/10.1049/cmu2.12781
26. BigchainDB Whitepaper v. 2.0. 2024. URL: https://www.bigchaindb.com/whitepaper/
27. McConaghy T., Marques A., Müller A. et al. BigchainDB: A Scalable Blockchain Database. URL: https://dbdb.io/db/bigchaindb/revisions/24
28. Apache Cassandra Documentation. URL: https://cassandra.apache.org/doc/
29. Apache HBase Documentation. URL: https://hbase.apache.org/
30. Amazon DynamoDB Documentation. URL: https://docs.aws.amazon.com/dynamodb/
31. Amazon QLDB Documentation. URL: https://docs.aws.amazon.com/qldb/
32. Ethereum Documentation. URL: https://ethereum.org/en/developers/docs/
33. Hyperledger Fabric Documentation. URL: https://hyperledger-fabric.readthedocs.io/
34. InfluxDB Documentation. URL: https://docs.influxdata.com/
35. MongoDB Documentation. 2025. URL: https://www.mongodb.com/docs/
36. Neo4j Documentation. URL: https://neo4j.com/docs/
37. Redis Documentation. URL: https://redis.io/docs/
38. Takada M. Distributed Systems for Fun and Profit. URL: https://book.mixu.net/distsys/
39. National Institute of Standards and Technology. Digital Signature Standard (DSS), FIPS PUB 186-4. Gaithersburg : NIST, 2013.
40. National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC), FIPS PUB 198-1. Gaithersburg : NIST, 2008.
41. Про затвердження стандарту вищої освіти зі спеціальності 123 «Комп'ютерна інженерія» для першого (бакалаврського) рівня вищої

освіти : наказ МОН України від 19.11.2018 р. № 1262. URL: https://mon.gov.ua

42. Про затвердження стандарту вищої освіти України: другий (магістерський) уровень, спеціальність 123 «Комп'ютерна інженерія» : наказ МОН України від 18.03.2021 р. № 330. URL: https://mon.gov.ua

ДОДАТКИ

Додаток А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
Максим ХРУСЛОВ
_____
(підпис)
«13» жовтня 2025року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувача першого (бакалаврського) рівня вищої освіти
(першого (бакалаврського) / другого (магістерського))

ХРИСТЕНКО НІКІТА
(прізвище, ім'я, по батькові студента)

Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»
1. Тема роботи **МЕТОД ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСІ ТА ВІДМОВОСТІЙКОСТІ ВИСОКОДОСУПНИХ РОЗПОДІЛЕНИХ NoSQL-СИСТЕМ ШЛЯХОМ ІНТЕГРАЦІЇ ДЕЦЕНТРАЛІЗОВАНОГО БЛОКЧЕЙН-МОДУЛЯ ВЕРИФІКАЦІЇ**
керівник роботи **Мороз Ольга Юріївна, PhD з комп.наук, доцент зво кафедри,**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ***№ 4101-5/1245 від 28.05.2026 року.***
2. Строк подання студентом роботи ***30 травня 2026 року***
3. Вихідні дані до роботи:
   1) розроблений метод децентралізованої верифікації цілісності даних;
   2) архітектура інтеграції NoSQL та блокчейн-модуля;
   3) програмний прототип високодоступної розподіленої системи;
   4) результати експериментального аналізу відмовостійкості та продуктивності;
   5) оцінка впливу блокчейн-верифікації на роботу NoSQL-систем;
   6) рекомендації щодо застосування у distributed computing системах.
4. Перелік питань, які потрібно розробити:
   1) Аналіз проблеми забезпечення цілісності та довіри до даних у розподілених NoSQL-базах.
   2) Дослідження існуючих підходів до верифікації даних та використання блокчейн-технологій у розподілених системах.
   3) Обґрунтування вибору архітектури та криптографічних механізмів для запропонованого методу.
   4) Проектування системи верифікації цілісності даних з використанням власного спрощеного блокчейну.
   5) Реалізація програмного прототипу системи з інтеграцією NoSQL-сховища та блокчейн-модуля.
   6) Тестування зі збоями, типу втрата вузла, модифікація даних, порушення журналів подій.
   7) Оцінка впливу запропонованого методу на продуктивність та надійність системи.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Формулювання наукової проблеми, обґрунтування актуальності, мети та завдань дослідження. Визначення об'єкта, предмета та методів дослідження	02.10.2025 – 18.10.2025
2.	Аналіз наукових досліджень стратегій обслуговування, причин відмов обладнання та методів вібраційного моніторингу	18.10.2025 – 02.12.2025
3.	Огляд і порівняльний аналіз наявних методів машинного навчання та існуючих систем прогнозного обслуговування	02.12.2025 – 20.12.2025
4.	Опис джерел даних, характеристик часових рядів та проведення попередньої обробки сигналів	02.12.2025 – 10.01.2026
5.	Виконання інженерії ознак (FFT, вейвлет-перетворення) та формування навчальних і тестових вибірок	10.01.2026 – 20.01.2026
6.	Обґрунтування вибору архітектури нейронної мережі	20.01.2026 – 08.02.2026
7.	Проєктування архітектури нейронної мережі, вибір функцій втрат та оптимізація гіперпараметрів	12.02.2026 – 18.02.2026
8.	Розробка архітектури програмного застосунку, веб-інтерфейсу та реалізація функціоналу моніторингу	18.02.2026 – 26.02.2026
9.	Тестування моделі на даних, оцінка швидкодії та формування інструкції з розгортання	10.03.2026 – 21.03.2026
10.	Оцінка ефективності розробленої моделі за критеріями	21.03.2026 – 30.03.2026
11.	Формування практичних рекомендацій щодо вдосконалення моделі та її адаптації до умов реального використання	08.04.2026 – 15.04.2026
12.	Оформлення пояснювальної записки кваліфікаційної роботи	15.04.2026 – 30.04.2026
13.	Оформлення звіту про переддипломну практику	30.04.2026 – 10.05.2026
14.	Підготовка та оформлення звітних матеріалів і додатків кваліфікаційної роботи, формування списку використаних джерел.	10.05.2026 – 20.05.2026
15.	Остаточне оформлення пояснювальної записки кваліфікаційної роботи для проходження перевірки на антиплагіат	до 25.05.2026
16.	Отримання відгуку керівника, підготовка висновків та подання роботи на рецензування	до 30.05.2026

5. Дата видачі завдання ***13 жовтня 2025 року.***

Здобувач вищої освіти _____ Нікіта ХРИСТЕНКО

Керівник роботи _____ Ольга МОРОЗ

Додаток Б

Затверджую

_____

«_____»_____2026р.

Технічне завдання
на розробку програмного виробу
«МЕТОД ВЕРИФІКАЦІЇ ЦІЛІСНОСТІ ДАНИХ У ВИСОКОДОСТУПНИХ РОЗПОДІЛЕНИХ БАЗАХ ДАНИХ (NoSQL) З ВИКОРИСТАННЯМ БЛОКЧЕЙН-ТЕХНОЛОГІЙ»

1. Вступ	**1.1. Назва:** метод верифікації цілісності даних у високодоступних розподілених базах даних (NoSQL) з використанням блокчейн-технологій. **1.2. Галузь застосування:** комп'ютерні технології, інформаційні системи, розподілені бази даних, кібербезпека.
2. Підстава для розробки	2.1. Навчальний план за спеціальністю 123 «Комп'ютерна інженерія» 2.2. Завдання на кваліфікаційну роботу бакалавра № 4101-5/225 від «30» травня 2026 р. (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3. Призначення розробки	**3.1. Мета розробки**: розробка та дослідження методу верифікації цілісності даних у розподілених NoSQL-системах на основі використання блокчейн-технології, криптографічного хешування та механізмів журналювання подій. **3.2. Призначення розробки:** надання можливості контролювати цілісність даних у розподілених NoSQL-сховищах, виявляти несанкціоновані зміни, видалення та розсинхронізацію даних між вузлами, а також забезпечувати перевірку достовірності історії змін і відновлення узгодженого стану системи. **3.3. Вихідні дані розробки:** тестові набори даних у форматі CSV та JSON, що імпортуються до NoSQL-бази даних MongoDB, а також результати моделювання операцій додавання, оновлення, видалення записів, збоїв вузлів і порушень цілісності даних у розподіленому середовищі.
4. Технічні вимоги до програмного виробу	**4.1. Вимоги до функціональних характеристик:** 1. імпорт та зберігання даних у розподіленому NoSQL-сховищі. 2. підтримка операцій додавання, оновлення та видалення записів. 3. автоматичне журналювання всіх змін даних. 4. формування криптографічних хешів подій та станів даних. 5. побудова блоків та ведення блокчейн-журналу змін. 6. перевірка цілісності даних на основі контрольних хешів. 7. виявлення несанкціонованих модифікацій, видалень та

	розсинхронізації вузлів. 8. відновлення узгодженого стану вузлів після збоїв. 9. проведення експериментів та оцінка продуктивності системи. **4.2. Вимоги до надійності:** система повинна забезпечувати достовірне виявлення порушень цілісності даних, коректне функціонування при частковій відмові вузлів, а також можливість відновлення узгодженого стану розподіленої системи після виникнення збоїв. **4.3. Вимоги до умов експлуатації:** відсутні. **4.4. Вимоги до складу і параметрів технічних засобів:** персональний комп'ютер або сервер із встановленою операційною системою Windows 10/11 або Linux, Python 3+, MongoDB, бібліотекою PyMongo та засоби візуалізації та аналізу даних. **4.5. Вимоги до інформаційної та програмної сумісності:** система повинна забезпечувати сумісність із MongoDB, Python 3 та сучасними операційними системами. Дані повинні підтримувати обробку у форматах JSON та CSV. **4.6. Вимоги до маркування та упаковки:** відсутні. **4.7. Вимоги до транспортування і зберігання:** відсутні. **4.8. Спеціальні вимоги:** відсутні.
5. Вимоги до програмної документації.	Програмною документацією до виробу «Система верифікації цілісності даних у розподілених NoSQL-системах із використанням блокчейн-технологій» вважати: 1) Справжнє Технічне завдання на розробку програмного вироби (представити у вигляді Додатку Б до пояснювальної записки до дипломної роботи). 2) Програму і методику випробувань розробленого програмного вироби (представити у вигляді Додатку В до пояснювальної записки до дипломної роботи). 3) Опис програмного вироби (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи).
6. Техніко-економічні показники	Програмною документацією до виробу «Система верифікації цілісності даних у розподілених NoSQL-системах із використанням блокчейн-технологій» вважати: 1) Справжнє Технічне завдання на розробку програмного вироби (представити у вигляді Додатку Б до пояснювальної записки до дипломної роботи). 2) Методику розрахунку інформативності змінних стану (у вигляді розділ 3 пояснювальної записки до кваліфікаційної роботи). 3) Опис виробу (представити в розділі 3 пояснювальної записки до кваліфікаційної роботи)

7. Стадії і етапи розробки	Формулювання наукової проблеми, обґрунтування актуальності, мети та завдань дослідження. Визначення об’єкта, предмета та методів дослідження
	Аналіз наукових досліджень стратегій обслуговування, причин відмов обладнання та методів вібраційного моніторингу
	Огляд і порівняльний аналіз наявних методів машинного навчання та існуючих систем прогнозного обслуговування
	Опис джерел даних, характеристик часових рядів та проведення попередньої обробки сигналів
	Виконання інженерії ознак (FFT, вейвлет-перетворення) та формування навчальних і тестових вибірок
	Обґрунтування вибору архітектури нейронної мережі
	Проєктування архітектури нейронної мережі, вибір функцій втрат та оптимізація гіперпараметрів
	Розробка архітектури програмного застосунку, веб-інтерфейсу та реалізація функціоналу моніторингу
	Тестування моделі на даних, оцінка швидкодії та формування інструкції з розгортання
	Оцінка ефективності розробленої моделі за критеріями
	Формування практичних рекомендацій щодо вдосконалення моделі та її адаптації до умов реального використання
	Оформлення пояснювальної записки кваліфікаційної роботи
	Оформлення звіту про переддипломну практику
	Підготовка та оформлення звітних матеріалів і додатків кваліфікаційної роботи, формування списку використаних джерел.
	Остаточне оформлення пояснювальної записки кваліфікаційної роботи для проходження перевірки на антиплагіат
	Отримання відгуку керівника, підготовка висновків та подання роботи на рецензування
8. Порядок контролю і приймання	1. Контроль виконання згідно календарного плану; 2. Захист на засіданні Екзаменаційної комісії; 3. Подання пояснювальної записки у паперовому та електронному вигляді.

Виконавець
студент групи К1-41
Христенко Н.

Замовник
PhD з інф. тех., доц. зво каф.
Мороз О. Ю.

Додаток В

Програма і методика випробувань програмного виробу
«МЕТОД ВЕРИФІКАЦІЇ ЦІЛІСНОСТІ ДАНИХ У ВИСОКОДОСТУПНИХ РОЗПОДІЛЕНИХ БАЗАХ ДАНИХ (NoSQL) З ВИКОРИСТАННЯМ БЛОКЧЕЙН-ТЕХНОЛОГІЙ»

1 Об'єкт випробувань

1. **Назва програмного виробу:** «Метод верифікації цілісності даних у високодоступних розподілених базах даних (NoSQL) з використанням блокчейн-технологій.»
2. **Галузь застосування:** комп'ютерні технології, інформаційні системи, розподілені бази даних, кібербезпека
3. Відомості запозичуються з відповідних розділів Технічного завдання (Додаток Б).

2. Мета випробувань

Підвищення надійності, відмовостійкості та довіри до даних у розподілених комп'ютерних системах шляхом розробки методу й архітектурних рішень для верифікації цілісності даних і стану NoSQL-сховищ із використанням децентралізованих блокчейн-технологій.

3. Загальні положення

1. Підстави для проведення випробувань: наказ по університету про створення атестаційної комісії з проведення захису кваліфікаційних робіт бакалаврів від 10 **червня 2026 р. № 214-к.**

2. Місце і тривалість випробувань: комп'ютерний клас кафедри комп'ютерних систем та роботоехніки **15-19 червня 2026 р.**

3. Обсяг випробувань: приймальні випробування програмного вироби проводяться в обсязі відповідному цієї програми і методики випробувань.

4. Організації, які беруть участь у випробуваннях: аестаційна комісія за участю розробника а керівника кваліфікаційної роботи.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

1. Забезпечувати зберігання та обробку даних у розподіленому NoSQL-середовищі.
2. Підтримувати операції додавання, оновлення та видалення записів.
3. Автоматично формувати журнал подій для всіх операцій над даними.
4. Виконувати криптографічне хешування подій та станів даних із використанням алгоритму SHA-256.
5. Забезпечувати формування блоків та ведення незмінного блокчейн-журналу подій.
6. Підтримувати побудову Merkle Root для агрегування контрольних хешів подій.

7. Виконувати перевірку цілісності даних шляхом порівняння поточних та еталонних хешів.
8. Виявляти несанкціоновану модифікацію, видалення або пошкодження даних.
9. Виявляти розсинхронізацію між вузлами розподіленої системи.
10. Забезпечувати перевірку цілісності блокчейн-ланцюга та журналу подій.
11. Підтримувати моделювання відмов вузлів і типових сценаріїв порушення цілісності даних.
12. Забезпечувати відновлення вузлів шляхом синхронізації з еталонним станом системи.
13. Надавати результати перевірки цілісності та виявлених порушень у зрозумілому для користувача вигляді.
14. Забезпечувати проведення експериментів з оцінки ефективності та продуктивності запропонованого методу.
15. Працювати на стандартних персональних комп'ютерах під керуванням сучасних операційних систем.

5. Вимоги до програмної документації

До програмної документації належаь:

1. Технічне завдання на розробку програмного виробу (Додаток А).
2. Програма і методика випробування (Додаток В)
3. Інструкція коистувача (Кваліфікаційна робоа розділ 3,4)

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань програмного виробу використовуються такі програмні та технічні засоби:

1. Персональний комп'ютер під керуванням операційної системи Windows 10/11.
2. Інтерпретатор Python 3+ та необхідні програмні бібліотеки.
3. СУБД MongoDB для зберігання та обробки даних.
4. MongoDB Compass для візуального перегляду колекцій та документів бази даних.
5. Тестовий набір даних у форматі CSV, що містить інформацію для формування документів NoSQL-сховища.
6. Розроблений програмний прототип системи верифікації цілісності даних із використанням блокчейн-модуля.
7. Засоби вимірювання часу виконання операцій та збору результатів експериментів.

6.2 Порядок проведення випробувань

Випробування програмного виробу проводяться у такій послідовності:

1. Запуск MongoDB та підготовка тестового середовища.
2. Імпорт початкового набору даних до NoSQL-сховища.
3. Ініціалізація логічних вузлів розподіленої системи.

4. Формування початкового журналу подій та створення контрольного блоку блокчейн-модуля.
5. Виконання перевірки цілісності даних у нормальному режимі роботи системи.
6. Проведення тестових операцій додавання, оновлення та видалення записів із подальшою перевіркою коректності журналювання та хешування.
7. Моделювання відмови одного з вузлів розподіленої системи.
8. Моделювання несанкціонованої модифікації даних на окремому вузлі.
9. Моделювання несанкціонованого видалення записів із бази даних.
10. Виконання повторної перевірки цілісності даних та аналіз результатів виявлення порушень.
11. Проведення процедури відновлення вузлів та синхронізації даних.
12. Повторна перевірка узгодженості реплік після відновлення.
13. Вимірювання часу виконання операцій у базовому режимі та в режимі з використанням механізмів верифікації.
14. Аналіз отриманих результатів та оцінка ефективності запропонованого методу.

Висновки: тест 1 успішно пройшов випробування, тест 2 успішно пройшов випробування і тест 3 успішно пройшов випробування (описано в розділі 4). Випробування пройшло успішно.

Виконавець: студент групи КІ-41, Христенко Н. _____

Додаток Г

Приклад коду програми

Файл attacks.py

```
def simulate_illegal_modification(cluster, node_name: str, record_id:
int, new_fields: dict) -> None:
    print(f"\n[ATTACK] illegal modification on {node_name}, record
{record_id}")
    cluster.direct_tamper_update(node_name, record_id, new_fields)

def simulate_illegal_deletion(cluster, node_name: str, record_id: int) ->
None:
    print(f"\n[ATTACK] illegal deletion on {node_name}, record
{record_id}")
    cluster.direct_tamper_delete(node_name, record_id)

def simulate_node_failure(cluster, node_name: str) -> None:
    print(f"\n[FAILURE] node {node_name} goes offline")
    cluster.set_node_status(node_name, False)

def simulate_node_recovery(cluster, node_name: str, authority_node: str =
None) -> None:
    print(f"\n[RECOVERY] node {node_name} restored from authority state")
    synced = cluster.repair_node(node_name,
authority_node=authority_node)
    print(f"{node_name} synchronized with {synced} records")
```

Файл blockchain.py

```
import hashlib
import json
from datetime import datetime
from typing import Dict, List, Optional

from merkle import merkle_root
from config import BLOCKCHAIN_COLLECTION, NODE_NAMES

class Block:
    def __init__(
        self,
        index: int,
        timestamp: str,
        batch_label: str,
        event_hashes: List[str],
        snapshot_hashes: Dict[str, str],
        merkle_root_value: str,
        previous_hash: str,
    ):
        self.index = index
        self.timestamp = timestamp
```

```
        self.batch_label = batch_label
        self.event_hashes = event_hashes
        self.snapshot_hashes = snapshot_hashes
        self.merkle_root_value = merkle_root_value
        self.previous_hash = previous_hash
        self.hash = self.calculate_hash()

    def to_dict(self) -> dict:
        return {
            "index": self.index,
            "timestamp": self.timestamp,
            "batch_label": self.batch_label,
            "event_hashes": self.event_hashes,
            "snapshot_hashes": self.snapshot_hashes,
            "merkle_root_value": self.merkle_root_value,
            "previous_hash": self.previous_hash,
            "hash": self.hash,
        }

    def calculate_hash(self) -> str:
        payload = {
            "index": self.index,
            "timestamp": self.timestamp,
            "batch_label": self.batch_label,
            "event_hashes": self.event_hashes,
            "snapshot_hashes": self.snapshot_hashes,
            "merkle_root_value": self.merkle_root_value,
            "previous_hash": self.previous_hash,
        }
        encoded = json.dumps(payload, sort_keys=True, ensure_ascii=False,
default=str).encode("utf-8")
        return hashlib.sha256(encoded).hexdigest()

class BlockchainLedger:
    def __init__(self, db):
        self.collection = db[BLOCKCHAIN_COLLECTION]
        self.chain: List[Block] = [self._create_genesis_block()]
        self.pending_event_hashes: List[str] = []

    def reset(self) -> None:
        self.collection.delete_many({})
        self.chain = [self._create_genesis_block()]
        self.pending_event_hashes = []

    def _create_genesis_block(self) -> Block:
        return Block(
            index=0,
            timestamp=str(datetime.utcnow()),
            batch_label="GENESIS",
            event_hashes=["0"],
            snapshot_hashes={node: "0" for node in NODE_NAMES},
```

```
            merkle_root_value="0",
            previous_hash="0",
        )

    def latest_block(self) -> Block:
        return self.chain[-1]

    def queue_event(self, event_hash: str) -> None:
        if event_hash:
            self.pending_event_hashes.append(event_hash)

    def commit_batch(
        self,
        snapshot_hashes: Dict[str, str],
        batch_label: str,
        event_hashes: Optional[List[str]] = None,
    ) -> Optional[Block]:
        if event_hashes is None:
            event_hashes = self.pending_event_hashes[:]

        combined = list(event_hashes) + [snapshot_hashes[node] for node
in sorted(snapshot_hashes)]
        root = merkle_root(combined)

        previous_hash = self.latest_block().hash
        block = Block(
            index=len(self.chain),
            timestamp=str(datetime.utcnow()),
            batch_label=batch_label,
            event_hashes=list(event_hashes),
            snapshot_hashes=dict(snapshot_hashes),
            merkle_root_value=root,
            previous_hash=previous_hash,
        )
        self.chain.append(block)
        self.collection.insert_one(block.to_dict())
        self.pending_event_hashes = []
        return block

    def is_chain_valid(self) -> bool:
        for i in range(1, len(self.chain)):
            current = self.chain[i]
            previous = self.chain[i - 1]
            if current.hash != current.calculate_hash():
                return False
            if current.previous_hash != previous.hash:
                return False
        return True

    def blocks(self) -> List[dict]:
        return [block.to_dict() for block in self.chain]
```

```
    def print_chain(self) -> None:
        print("\n=== BLOCKCHAIN LEDGER ===")
        for block in self.chain:
            print(f"Index: {block.index}")
            print(f"Label: {block.batch_label}")
            print(f"Time: {block.timestamp}")
            print(f"Merkle root: {block.merkle_root_value}")
            print(f"Hash: {block.hash}")
            print(f"Prev: {block.previous_hash}")
            print("-" * 60)
```

Файл cluster.py

```
import copy
import csv
import hashlib
import json
from datetime import datetime
from typing import Dict, List, Optional

from pymongo import MongoClient

from config import (
    BLOCKCHAIN_COLLECTION,
    DB_NAME,
    EVENT_LOG_COLLECTION,
    MONGO_URI,
    NODE_COLLECTION_SUFFIX,
    NODE_NAMES,
    PRIMARY_NODE,
)

class DistributedNoSQLCluster:
    def __init__(self):
        self.client = MongoClient(MONGO_URI)
        self.db = self.client[DB_NAME]

        self.node_collections = {
            node: self.db[f"{node}{NODE_COLLECTION_SUFFIX}"]
            for node in NODE_NAMES
        }
        self.event_log = self.db[EVENT_LOG_COLLECTION]

        self.node_status = {node: True for node in NODE_NAMES}

    @staticmethod
    def _canonical_json(obj) -> str:
        return json.dumps(
            obj,
```

```
            ensure_ascii=False,
            sort_keys=True,
            separators=(",", ":"),
            default=str,
        )

    @classmethod
    def _hash_payload(cls, payload) -> str:
        data = cls._canonical_json(payload).encode("utf-8")
        return hashlib.sha256(data).hexdigest()

    @staticmethod
    def _normalize_record(record: dict) -> dict:
        normalized = dict(record)
        normalized["id"] = int(normalized["id"])
        normalized["balance"] = int(normalized["balance"])
        normalized["name"] = str(normalized["name"])
        normalized["city"] = str(normalized["city"])
        normalized["account_type"] = str(normalized["account_type"])
        normalized["status"] = str(normalized["status"])
        return normalized

    def reset(self) -> None:
        for collection in self.node_collections.values():
            collection.delete_many({})
        self.event_log.delete_many({})
        self.node_status = {node: True for node in NODE_NAMES}

    def bootstrap_from_csv(self, csv_path: str) -> List[dict]:
        records = []
        with open(csv_path, "r", encoding="utf-8") as f:
            reader = csv.DictReader(f)
            for row in reader:
                records.append(self._normalize_record(row))

        for node_name, collection in self.node_collections.items():
            collection.delete_many({})
            if records:
                collection.insert_many(copy.deepcopy(records))

        self.event_log.delete_many({})
        return records

    def set_node_status(self, node_name: str, online: bool) -> None:
        if node_name not in self.node_status:
            raise ValueError(f"Unknown node: {node_name}")
        self.node_status[node_name] = online

    def online_nodes(self) -> List[str]:
        return [node for node, active in self.node_status.items() if
active]
```

```
    def get_node_records(self, node_name: str) -> List[dict]:
        if node_name not in self.node_collections:
            raise ValueError(f"Unknown node: {node_name}")
        return list(self.node_collections[node_name].find({}, {"_id":
0}))

    def get_record(self, node_name: str, record_id: int) ->
Optional[dict]:
        return self.node_collections[node_name].find_one({"id":
int(record_id)}, {"_id": 0})

    def _apply_to_nodes(self, operation_fn, target_nodes:
Optional[List[str]] = None) -> List[str]:
        if target_nodes is None:
            target_nodes = self.online_nodes()

        applied_nodes = []
        for node_name in target_nodes:
            if self.node_status.get(node_name, False):
                operation_fn(self.node_collections[node_name])
                applied_nodes.append(node_name)
        return applied_nodes

    def _store_event(self, event_doc: dict) -> str:
        event_hash = self._hash_payload(event_doc)
        event_doc_with_hash = dict(event_doc)
        event_doc_with_hash["event_hash"] = event_hash
        self.event_log.insert_one(event_doc_with_hash)
        return event_hash

    def create_record(
        self,
        record: dict,
        target_nodes: Optional[List[str]] = None,
        source: str = "client",
    ) -> Optional[str]:
        normalized = self._normalize_record(record)

        def op(collection):
            collection.replace_one({"id": normalized["id"]}, normalized,
upsert=True)

        applied_nodes = self._apply_to_nodes(op,
target_nodes=target_nodes)
        if not applied_nodes:
            return None

        event_doc = {
            "ts": datetime.utcnow().isoformat(),
            "operation": "insert",
```

```
            "record_id": normalized["id"],
            "before": None,
            "after": normalized,
            "target_nodes": applied_nodes,
            "source": source,
        }
        return self._store_event(event_doc)

    def update_record(
        self,
        record_id: int,
        updates: dict,
        target_nodes: Optional[List[str]] = None,
        source: str = "client",
        log_event: bool = True,
    ) -> Optional[str]:
        record_id = int(record_id)
        current = self.get_record(PRIMARY_NODE, record_id)
        if current is None:
            raise ValueError(f"Record {record_id} not found on primary
node")

        before = copy.deepcopy(current)
        after = copy.deepcopy(current)
        for key, value in updates.items():
            after[key] = value

        after = self._normalize_record(after)

        def op(collection):
            collection.update_one({"id": record_id}, {"$set": updates})

        applied_nodes = self._apply_to_nodes(op,
target_nodes=target_nodes)
        if not applied_nodes:
            return None

        if not log_event:
            return ""

        event_doc = {
            "ts": datetime.utcnow().isoformat(),
            "operation": "update",
            "record_id": record_id,
            "before": before,
            "after": after,
            "target_nodes": applied_nodes,
            "source": source,
        }
        return self._store_event(event_doc)
```

```
    def delete_record(
        self,
        record_id: int,
        target_nodes: Optional[List[str]] = None,
        source: str = "client",
        log_event: bool = True,
    ) -> Optional[str]:
        record_id = int(record_id)
        current = self.get_record(PRIMARY_NODE, record_id)
        if current is None:
            raise ValueError(f"Record {record_id} not found on primary
node")

        before = copy.deepcopy(current)

        def op(collection):
            collection.delete_one({"id": record_id})

        applied_nodes = self._apply_to_nodes(op,
target_nodes=target_nodes)
        if not applied_nodes:
            return None

        if not log_event:
            return ""

        event_doc = {
            "ts": datetime.utcnow().isoformat(),
            "operation": "delete",
            "record_id": record_id,
            "before": before,
            "after": None,
            "target_nodes": applied_nodes,
            "source": source,
        }
        return self._store_event(event_doc)

    def direct_tamper_update(self, node_name: str, record_id: int,
updates: dict) -> None:
        if node_name not in self.node_collections:
            raise ValueError(f"Unknown node: {node_name}")
        if not self.node_status.get(node_name, False):
            raise RuntimeError(f"Node {node_name} is offline")

        self.node_collections[node_name].update_one({"id":
int(record_id)}, {"$set": updates})

    def direct_tamper_delete(self, node_name: str, record_id: int) ->
None:
        if node_name not in self.node_collections:
            raise ValueError(f"Unknown node: {node_name}")
```

```
        if not self.node_status.get(node_name, False):
            raise RuntimeError(f"Node {node_name} is offline")

        self.node_collections[node_name].delete_one({"id":
int(record_id)})

    def repair_node(self, node_name: str, authority_node: Optional[str] =
None) -> int:
        if node_name not in self.node_collections:
            raise ValueError(f"Unknown node: {node_name}")

        if authority_node is None:
            online = self.online_nodes()
            authority_node = PRIMARY_NODE if PRIMARY_NODE in online else
(online[0] if online else PRIMARY_NODE)

        source_records = self.get_node_records(authority_node)
        collection = self.node_collections[node_name]
        collection.delete_many({})
        if source_records:
            collection.insert_many(copy.deepcopy(source_records))

        self.node_status[node_name] = True
        return len(source_records)

    def snapshot_hash(self, node_name: str) -> str:
        records = self.get_node_records(node_name)
        sorted_records = sorted(records, key=lambda x: x["id"])
        return self._hash_payload(sorted_records)

    def cluster_snapshot_hashes(self) -> Dict[str, str]:
        return {node: self.snapshot_hash(node) for node in NODE_NAMES}

    def current_state_report(self) -> dict:
        report = {}
        for node in NODE_NAMES:
            records = self.get_node_records(node)
            report[node] = {
                "online": self.node_status[node],
                "count": len(records),
                "snapshot_hash": self.snapshot_hash(node),
            }
        return report

    def event_log_records(self) -> List[dict]:
        return list(self.event_log.find({}, {"_id": 0}))
```

Файл config.py

```
MONGO_URI = "mongodb://localhost:27017/"
DB_NAME = "distributed_nosql_blockchain_db"

NODE_NAMES = ["node_1", "node_2", "node_3"]
NODE_COLLECTION_SUFFIX = "_docs"

EVENT_LOG_COLLECTION = "audit_events"
BLOCKCHAIN_COLLECTION = "audit_chain"

DATA_FILE = "users.csv"

PRIMARY_NODE = "node_1"
BATCH_SIZE = 8
```

Файл data_generator.py

```
import csv
import random
from pathlib import Path

NAMES = [
    "Ivan", "Anna", "Petro", "Olha", "Maria", "Andrii", "Sofiia",
"Danylo",
    "Iryna", "Nazar", "Oksana", "Vlad", "Kateryna", "Yehor", "Polina",
    "Maksym", "Yulia", "Taras", "Lilia", "Artem"
]

CITIES = [
    "Kyiv", "Lviv", "Kharkiv", "Dnipro", "Odesa", "Poltava",
    "Vinnytsia", "Zaporizhzhia", "Chernihiv", "Ivano-Frankivsk"
]

ACCOUNT_TYPES = ["student", "basic", "premium", "business"]

def generate_sample_csv(path: str, n: int = 120, seed: int = 42) -> None:
    random.seed(seed)
    path_obj = Path(path)

    rows = []
    for i in range(1, n + 1):
        row = {
            "id": i,
            "name": random.choice(NAMES) + f"_{i}",
            "balance": random.randint(50, 5000),
            "city": random.choice(CITIES),
            "account_type": random.choice(ACCOUNT_TYPES),
```

```
            "status": random.choice(["active", "active", "active",
"suspended"]),
        }
        rows.append(row)

    with path_obj.open("w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(
            f,
            fieldnames=["id", "name", "balance", "city", "account_type",
"status"]
        )
        writer.writeheader()
        writer.writerows(rows)
```

Файл experiments.py

```
import random
import time

from blockchain import BlockchainLedger
from cluster import DistributedNoSQLCluster
from config import BATCH_SIZE, DATA_FILE
from data_generator import generate_sample_csv

def run_latency_experiment(iterations: int = 30, batch_size: int =
BATCH_SIZE) -> None:
    print("\n=== LATENCY EXPERIMENT ===")

    generate_sample_csv(DATA_FILE, n=120, seed=42)

    # Baseline: writes without audit logging
    cluster_1 = DistributedNoSQLCluster()
    blockchain_1 = BlockchainLedger(cluster_1.db)
    cluster_1.reset()
    blockchain_1.reset()
    cluster_1.bootstrap_from_csv(DATA_FILE)

    ids = [doc["id"] for doc in cluster_1.get_node_records("node_1")]
    random.seed(7)

    start = time.perf_counter()
    for i in range(iterations):
        rid = random.choice(ids)
        new_balance = random.randint(100, 9000)
        cluster_1.update_record(rid, {"balance": new_balance},
log_event=False)
    baseline_time = time.perf_counter() - start

    # Secure mode: writes + event log + blockchain batching
    cluster_2 = DistributedNoSQLCluster()
```

```
    blockchain_2 = BlockchainLedger(cluster_2.db)
    cluster_2.reset()
    blockchain_2.reset()
    cluster_2.bootstrap_from_csv(DATA_FILE)

    ids = [doc["id"] for doc in cluster_2.get_node_records("node_1")]
    random.seed(7)

    start = time.perf_counter()
    for i in range(iterations):
        rid = random.choice(ids)
        new_balance = random.randint(100, 9000)
        event_hash = cluster_2.update_record(rid, {"balance":
new_balance}, log_event=True)
        blockchain_2.queue_event(event_hash)

        if (i + 1) % batch_size == 0:
            blockchain_2.commit_batch(
                cluster_2.cluster_snapshot_hashes(),
                batch_label=f"batch_{(i + 1) // batch_size}"
            )

    if blockchain_2.pending_event_hashes:
        blockchain_2.commit_batch(
            cluster_2.cluster_snapshot_hashes(),
            batch_label="final_batch"
        )

    secure_time = time.perf_counter() - start

    overhead = secure_time - baseline_time
    overhead_pct = (overhead / baseline_time * 100) if baseline_time > 0
else 0

    print(f"Baseline writes time: {baseline_time:.4f} s")
    print(f"Secure mode time    : {secure_time:.4f} s")
    print(f"Overhead            : {overhead:.4f} s")
    print(f"Overhead %          : {overhead_pct:.2f} %")
```

Файл main.py

```
import random

from attacks import (
    simulate_illegal_deletion,
    simulate_illegal_modification,
    simulate_node_failure,
    simulate_node_recovery,
)
from blockchain import BlockchainLedger
```

```
from cluster import DistributedNoSQLCluster
from config import BATCH_SIZE, DATA_FILE, PRIMARY_NODE
from data_generator import generate_sample_csv
from experiments import run_latency_experiment
from verifier import Verifier

def make_legitimate_activity(cluster, blockchain):
    print("\n=== LEGITIMATE OPERATIONS ===")

    random.seed(21)
    sample_ids = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

    ops_done = 0

    for rid in sample_ids[:4]:
        event_hash = cluster.update_record(
            rid,
            {"balance": random.randint(500, 8000)},
            log_event=True,
        )
        blockchain.queue_event(event_hash)
        ops_done += 1

    cluster.create_record(
        {
            "id": 1001,
            "name": "New_User_1001",
            "balance": 1200,
            "city": "Kyiv",
            "account_type": "premium",
            "status": "active",
        }
    )
    event_hash = cluster.event_log.find_one(
        {"record_id": 1001},
        {"_id": 0, "event_hash": 1}
    )["event_hash"]
    blockchain.queue_event(event_hash)
    ops_done += 1

    cluster.create_record(
        {
            "id": 1002,
            "name": "New_User_1002",
            "balance": 2400,
            "city": "Lviv",
            "account_type": "business",
            "status": "active",
        }
    )
    event_hash = cluster.event_log.find_one(
```

```
        {"record_id": 1002},
        {"_id": 0, "event_hash": 1}
    )["event_hash"]
    blockchain.queue_event(event_hash)
    ops_done += 1

    cluster.delete_record(19, log_event=True)
    event_hash = cluster.event_log.find_one(
        {"operation": "delete", "record_id": 19},
        {"_id": 0, "event_hash": 1}
    )["event_hash"]
    blockchain.queue_event(event_hash)
    ops_done += 1

    for rid in sample_ids[4:7]:
        event_hash = cluster.update_record(
            rid,
            {"status": "active", "balance": random.randint(1000, 6000)},
            log_event=True,
        )
        blockchain.queue_event(event_hash)
        ops_done += 1

    if blockchain.pending_event_hashes:
        blockchain.commit_batch(
            cluster.cluster_snapshot_hashes(),
            batch_label="checkpoint_1"
        )

    print(f"Legitimate operations completed: {ops_done}")

def print_cluster_state(cluster):
    print("\n=== CLUSTER STATE ===")
    report = cluster.current_state_report()
    for node, info in report.items():
        print(
            f"{node}: online={info['online']}, "
            f"count={info['count']}, "
            f"snapshot={info['snapshot_hash'][:16]}..."
        )

if __name__ == "__main__":
    generate_sample_csv(DATA_FILE, n=120, seed=42)

    cluster = DistributedNoSQLCluster()
    blockchain = BlockchainLedger(cluster.db)
    verifier = Verifier()

    cluster.reset()
    blockchain.reset()
    cluster.bootstrap_from_csv(DATA_FILE)
```

```
    print_cluster_state(cluster)
    make_legitimate_activity(cluster, blockchain)

    print("\n=== BLOCKCHAIN AFTER LEGITIMATE OPERATIONS ===")
    blockchain.print_chain()

    print("\n=== AUDIT BEFORE ATTACK ===")
    verifier.full_audit(cluster, blockchain)

    simulate_node_failure(cluster, "node_3")
    cluster.update_record(7, {"balance": 7777}, log_event=True,
target_nodes=["node_1", "node_2"])
    blockchain.queue_event(
        cluster.event_log.find_one(
            {"operation": "update", "record_id": 7},
            {"_id": 0, "event_hash": 1}
        )["event_hash"]
    )

    simulate_illegal_modification(cluster, "node_2", 5, {"balance":
999999})
    simulate_illegal_deletion(cluster, "node_2", 11)

    print_cluster_state(cluster)

    print("\n=== AUDIT AFTER ATTACK ===")
    verifier.full_audit(cluster, blockchain)

    simulate_node_recovery(cluster, "node_3",
authority_node=PRIMARY_NODE)
    simulate_node_recovery(cluster, "node_2",
authority_node=PRIMARY_NODE)

    if blockchain.pending_event_hashes:
        blockchain.commit_batch(
            cluster.cluster_snapshot_hashes(),
            batch_label="recovery_checkpoint"
        )
    else:
        blockchain.commit_batch(
            cluster.cluster_snapshot_hashes(),
            batch_label="recovery_checkpoint"
        )

    print("\n=== AUDIT AFTER RECOVERY ===")
    verifier.full_audit(cluster, blockchain)

    print("\n=== LATENCY BENCHMARK ===")
    run_latency_experiment(iterations=30, batch_size=BATCH_SIZE)
```

Файл merkle.py

```
import hashlib
from typing import List

def _sha256(text: str) -> str:
    return hashlib.sha256(text.encode("utf-8")).hexdigest()

def merkle_root(hashes: List[str]) -> str:
    if not hashes:
        return _sha256("EMPTY")

    level = hashes[:]
    while len(level) > 1:
        if len(level) % 2 == 1:
            level.append(level[-1])

        next_level = []
        for i in range(0, len(level), 2):
            next_level.append(_sha256(level[i] + level[i + 1]))
        level = next_level

    return level[0]
```

Файл verifier.py

```
from typing import Dict, List

from config import NODE_NAMES, PRIMARY_NODE

class Verifier:
    @staticmethod
    def _hash_event_without_stored_hash(cluster, event_doc: dict) -> str:
        payload = dict(event_doc)
        payload.pop("_id", None)
        payload.pop("event_hash", None)
        return cluster._hash_payload(payload)

    def verify_event_log(self, cluster) -> bool:
        ok = True
        events = list(cluster.event_log.find({}, {"_id": 0}).sort("ts",
1))

        print("\n=== EVENT LOG VERIFICATION ===")
        for event in events:
            stored_hash = event.get("event_hash")
            calculated_hash =
self._hash_event_without_stored_hash(cluster, event)
            if stored_hash != calculated_hash:
```

```
                ok = False
                print("EVENT LOG TAMPERING DETECTED")
                print(event)
            else:
                print(f"OK: {event['operation']} on record
{event['record_id']}")
        return ok

    def verify_chain(self, blockchain) -> bool:
        print("\n=== CHAIN VERIFICATION ===")
        valid = blockchain.is_chain_valid()
        print(f"Blockchain valid: {valid}")
        return valid

    def verify_latest_snapshot(self, cluster, blockchain) -> bool:
        print("\n=== LATEST SNAPSHOT VERIFICATION ===")
        latest_block = blockchain.latest_block()
        current_hashes = cluster.cluster_snapshot_hashes()

        ok = True
        for node in NODE_NAMES:
            expected = latest_block.snapshot_hashes.get(node)
            current = current_hashes.get(node)
            if expected != current:
                ok = False
                print(f"SNAPSHOT MISMATCH ON {node}")
                print(f"expected: {expected}")
                print(f"current : {current}")
            else:
                print(f"OK: {node}")

        return ok

    def compare_nodes(self, cluster) -> bool:
        print("\n=== NODE DIVERGENCE CHECK ===")
        hashes = cluster.cluster_snapshot_hashes()
        reference = hashes[PRIMARY_NODE]
        ok = True

        for node, h in hashes.items():
            if h != reference:
                ok = False
                print(f"NODE DIVERGENCE: {node} differs from
{PRIMARY_NODE}")
            else:
                print(f"OK: {node}")

        return ok

    def detect_missing_records(self, cluster, reference_node: str =
PRIMARY_NODE) -> bool:
```

```
        print("\n=== MISSING RECORD CHECK ===")
        reference_ids = {doc["id"] for doc in
cluster.get_node_records(reference_node)}
        ok = True

        for node in NODE_NAMES:
            current_ids = {doc["id"] for doc in
cluster.get_node_records(node)}
            missing = sorted(reference_ids - current_ids)
            if missing:
                ok = False
                print(f"{node}: missing records -> {missing}")
            else:
                print(f"{node}: no missing records")

        return ok

    def full_audit(self, cluster, blockchain) -> Dict[str, bool]:
        results = {
            "chain_valid": self.verify_chain(blockchain),
            "event_log_valid": self.verify_event_log(cluster),
            "latest_snapshot_ok": self.verify_latest_snapshot(cluster,
blockchain),
            "nodes_consistent": self.compare_nodes(cluster),
            "missing_records_ok": self.detect_missing_records(cluster),
        }

        print("\n=== AUDIT SUMMARY ===")
        for k, v in results.items():
            print(f"{k}: {v}")

        return results
```