

Національний Технічний Університет України
«Київський Політехнічний Інститут імені Ігоря Сікорського»
Міністерство освіти і науки України
Національний Технічний Університет України
«Київський Політехнічний Інститут імені Ігоря Сікорського»
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

НІКІТІН ВАЛЕРІЙ АНДРІЙОВИЧ

УДК 004.65

МЕТОДИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ УЗГОДЖЕНОСТІ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ

126 Інформаційні системи та технології

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело _____ В. А. Нікітін

Науковий керівник Крилов Євген Володимирович, к. т. н., доцент

Київ – 2024

АНОТАЦІЯ

Нікітін В. А. Методи підвищення ефективності узгодженості даних в інформаційних системах. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 126 – Інформаційні системи та технології в галузі знань 12 – Інформаційні технології. – Національний Технічний Університет України «Київський Політехнічний Інститут імені Ігоря Сікорського», Київ, 2024.

Дисертаційна робота присвячена розробці методів для розподілених документо-орієнтованих баз даних, які дозволяють пришвидшити узгодження даних та покращити колізійну стійкість в процесі пошуку неузгоджених даних в різноманітних мережевих інформаційних системах, таких як, IoT, гетерогенні мультикомп'ютерні системи, аналітичні системи органів адміністративного управління, фінансові системи, дослідження екологічної безпеки та природокористування та інші. Також, було реалізовано спеціальне програмне забезпечення для проведення дослідів. В результаті виконання дисертаційної роботи було отримано такі результати.

Розроблено метод забезпечення узгодженості даних у розподілених нереляційних документо-орієнтованих базах даних з використанням транзакційного годинника. Він отримує транзакції від клієнтських додатків та зберігає їх у відповідних чергах. Черги опрацьовуються в залежності від пріоритету транзакції. При найвищому пріоритеті черга опрацьовується раніше, в порівнянні з тими, які мають нижчий пріоритет. Це дозволяє виділяти критично важливі дані (такі, як фінансові транзакції), які мають бути опрацьовані першочергово. При опрацьованні черги, транзакційний годинник об'єднує транзакції в результуючу транзакцію. Для цього він використовує час створення транзакції, що дозволяє глобально впорядковувати та об'єднувати їх в тому порядку, в якому вони були створені, а не отримані годинником. Після отримання результуючої транзакції, вона передається іншим реплікам, на яких відбувається запис. Слід зазначити, що при використанні

транзакційного годинника, операції читання відбуваються напряму з реплік, що дозволяє зменшити навантаження на хостову машину.

Удосконалено метод активної антиентропії з використанням спектрального фільтру Блума та алгоритму хешування замість дерева Меркла. Його створення обумовлено тим, що класичний механізм активної антиентропії використовує багато затратних операцій хешування. Також, при хешуванні великої кількості даних збільшується вірогідність колізій, яка може вплинути на вчасну ідентифікацію неузгодженості. Якщо таке буде відбуватися, то система може знаходитись у неузгодженому стані через те, що запуск процедури узгодження розпочнеться пізніше. Тому, для розподілених нереляційних документо-орієнтованих баз даних було вирішено використати протокол пліток, який полягає у децентралізованому способі взаємодії вузлів. Це забезпечує надійність, в порівнянні з централізованим підходом, оскільки виведення з ладу одного вузла не вплине на доступність системи. Проблема використання централізованого підходу полягає в тому, що виведення головного вузла з ладу вимагає певної затримки через використання консенсусного протоколу, який використовується для обрання нового головного вузла з існуючих реплік. Децентралізований підхід дозволяє інформаційній системі бути доступною для операцій запису, хоча це ускладнює підтримку узгодженості. Для пошуку неузгодженостей використовується певний знімок, який складається зі спектрального фільтру Блума та хеш-значення.

Спеціально для розробленого методу активної антиентропії було модифіковано алгоритм формування спектрального фільтру Блума, який дозволяє пришвидшити ідентифікування неузгодженості даних в методі активної антиентропії. Швидкість формування спектрального фільтру Блума було збільшено через використання алгоритму на основі простих чисел замість використання хеш-функцій. Результати дослідів показують, що розроблений алгоритм має вищу колізійну стійкість, ніж при використанні однієї хеш-функції та вищу швидкодію у порівнянні з використанням декількох функцій хешування.

Також, спеціально для розробленого методу активної антиентропії було розроблено спосіб хешування, який стійкий до колізій. Його мета полягає у зменшенні кількості колізій при хешуванні даних, які відрізняються за розміром. Це є дуже важливим для активної антиентропії, оскільки дозволяє вчасно знайти неузгодженість.

Реалізовано спеціальний сервіс транзакційного годинника із власним програмним інтерфейсом для можливості застосування в інформаційних системах розробленого методу забезпечення узгодженості з використанням транзакційного годинника з розподіленою базою даних MongoDB.

Реалізовано спеціальний сервіс активної антиентропії із власним програмним інтерфейсом для використання в інформаційних системах розробленого методу активної антиентропії з розподіленою базою даних MongoDB.

Реалізовано прототип фінансової інформаційної системи, в якій розподілена база даних складається з восьми вузлів. З її використанням були отримані результати дослідження розроблених методів забезпечення узгодженості даних. Програмний інтерфейс було реалізовано з використанням мови програмування Python3. Для створення тестового середовища використовувався Docker та docker-compose для оркестрації необхідними компонентами.

Дисертаційна робота складається зі вступу, 3 розділів, загальних висновків, списку використаних джерел із 49 найменувань та 6 додатків. Загальний обсяг дисертації становить 165 сторінок, з яких 133 сторінки основного тексту, містить 76 рисунків та 5 таблиць.

Ключові слова: база даних, Інтернет речей, канали комунікації, критичні інформаційні ресурси, система управління, математична модель, гетерогенні мультикомп'ютерні системи, потоки текстових даних, стиснення даних, інформаційні технології, достовірність передачі інформації, проблеми оптимізації, цілісність, відмовостійкість, алгоритми консенсусу.

ABSTRACT

Nikitin V. A. Methods of increasing the efficiency of data consistency in information systems. - Qualifying scientific work is presented on the rights of the manuscript.

The philosophy doctor thesis is carried out in specialty 126 – Information Systems and Technologies, of the knowledge field 12 – Information Technologies. – National Technical University of Ukraine “Kyiv Polytechnic Institute”, Ministry of Education and Science of Ukraine, Kyiv, 2024.

The dissertation work is devoted to the development of methods for distributed document-oriented databases that allow speeding up data reconciliation and improving collision resistance in the process of searching for inconsistent data in various network information systems, such as IoT, heterogeneous multicomputer systems, analytical systems of administrative management, financial systems, research on environmental safety and nature management, and others. Also, special software for conducting experiments was implemented. The following results were obtained as a result of the dissertation work.

A method of ensuring data consistency in distributed NoSQL document-oriented databases using a transactional clock has been developed. It receives transactions from client applications and stores them in appropriate queues. Queues are processed depending on the priority of the transaction. With the highest priority, the queue is processed earlier, compared to those with a lower priority. This allows you to figure out critical data (for example, transmitting financial transactions) that should be processed first. When the queue is processed, the transaction clock merges the transactions into a resulting transaction. To do this, it uses the creation time of the transaction, which allows them to be globally ordered and merged in the order in which they were created, rather than received by the transactional clock. After receiving the resulting transaction, it is transmitted to other replicas on which writing takes place. It should be noted that when using a transactional clock, read operations take place directly from replicas, which allows to reduce the load on the host machine.

The active anti-entropy method has been improved using a Bloom spectral filter and a hashing algorithm instead of a Merkle tree. Its creation is due to the fact that the classical

mechanism of active anti-entropy uses many expensive hashing operations. Also, when hashing a large amount of data, the probability of collisions increases, which can affect the timely identification of inconsistencies. If this happens, the system may be in an inconsistent state because the start of the reconciliation procedure will occur later. Therefore, for distributed NoSQL document-oriented databases, it was decided to use the gossip protocol, which consists in a decentralized method of node interaction. This provides reliability, compared to a centralized approach, because the failure of one node will not affect the availability of the system. The problem with using a centralized approach is that failing a master requires some latency due to the consensus protocol used to elect a new master from existing replicas. A decentralized approach allows the information system to be available for record operations, although this makes it difficult to maintain consistency. To search for inconsistencies, a certain snapshot is used, which consists of a spectral Bloom filter and a hash value.

The algorithm to produce the Bloom spectral filter was modified specifically for the developed method of active anti-entropy, which makes it possible to speed up the identification of data inconsistencies in the method of active anti-entropy. The speed of forming the spectral Bloom filter has been increased by using an algorithm based on prime numbers instead of using hash functions. The results of the experiments show that the developed algorithm has higher collision resistance comparing to single hash function and higher speed comparing to usage of a several hashing functions.

Also, a collision-resistant hashing method was developed specifically for the developed active anti-entropy method. Its purpose is to reduce the number of collisions when hashing data that differs in size. This is very important for active antientropy, as it allows early mismatch detection.

A special transaction clock service with its own application programming interface has been implemented for the possibility of using the developed method of ensuring consistency using a transaction clock with a distributed MongoDB database in information systems.

A special service of active antientropy with its own application programming interface has been implemented for the use in information systems of the developed method of active antientropy with the MongoDB distributed database.

A prototype of the financial information system was implemented, in which the distributed database consists of eight nodes. With its use, the results of the study of the developed methods of ensuring data consistency were obtained. The application programming interface was implemented using the Python3 programming language. Docker and docker-compose were used to create test environment and orchestrate the necessary components.

The thesis consists of an introduction, four chapters, general conclusions, the reference list with 49 references and 6 appendices. The total volume of the thesis is 165 pages, of which 133 pages are the main text, contains 76 figures and 5 tables.

Keywords: database, IoT, Communication channels, critical informational resources, control system, mathematical model, heterogeneous multicomputer systems, streams of text data, data reduction, information technologies, the reliability of information transmission, optimization problems, consistency, fault tolerance, consensus algorithms.

Список публікацій здобувача

Наукові праці, в яких опубліковано основні наукові результати дисертації:

1. **Nikitin V.**, Krylov E., Kornaga Y., Anikin V. Combined indexing method in NoSQL databases // Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. №1(38), 2021. P. 3 – 9. DOI: <https://doi.org/10.20535/1560-8956.38.2021.232948>;
2. **Nikitin V.**, Krylov E., Kornaga Y., Anikin V. Modification of hashing algorithm to increase rate of operations in NoSQL databases // Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. № 2 (39), 2021. P. 39 – 43. DOI: <https://doi.org/10.20535/1560-8956.39.2021.247395> ;
3. Mukhin V., Kornaga Y., Zavgorodnii V., Fartushnyi I., Pashov R., **Nikitin V.**, Stepanov A. Method of determining the required number of database nodes in a distributed data processing system / 2021 IEEE 3rd International Conference on Advanced Trends in Information Theory (ATIT) // IEEE. DOI: <https://doi.org/10.1109/ATIT54053.2021.9678569> ;
4. **Nikitin V.**, Krylov E. Comparison of hashing methods for supporting consistency in distributed databases // Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. № 1 (40), 2022. P. 48 – 53. DOI: <https://doi.org/10.20535/1560-8956.40.2022.261646> ;
5. **Nikitin V.**, Krylov E. A collision-resistant hashing algorithm for maintaining consistency in distributed nosql databases // Adaptive Systems of Automatic Control Interdepartmental scientific and technical collection. № 2 (41), 2022. P. 45 – 57. DOI: <https://doi.org/10.20535/1560-8956.41.2022.271338> ;
6. Крилов Є. В., **Нікітін В. А.** Використання транзакційного годинника для пришвидшення процесу узгодження даних в розподілених системах // Фахове видання категорії Б “Науковий вісник Ужгородського університету. Серія «Математика і інформатика». № 1 (42), 2023. С. 188 – 192. DOI: [https://doi.org/10.24144/2616-7700.2023.42\(1\).188-192](https://doi.org/10.24144/2616-7700.2023.42(1).188-192) ;

7. **Nikitin V.**, Krylov E. Primary-based Spectral Bloom filter for the ensuring consistency in distributed document-based NoSQL databases using active anti-entropy mechanism // Computer Systems and Information Technologies. №3, (2023). P. 75 – 80. DOI: <https://doi.org/10.31891/csit-2023-3-9> ;

8. **Nikitin V.**, Krylov E. Active anti-entropy mechanism based on Spectral Bloom filter and PH-2 hash algorithm for reconciliation of replicas of NoSQL distributed document oriented databases // *Information Technology and Society*. №3 (9), 2023. P. 63 – 67. DOI: <https://doi.org/10.32689/maup.it.2023.3.8> .

Наукові праці, які засвідчують апробацію матеріалів дисертації:

9. **Nikitin V.**, Krylov E. Consistency optimization methods in distributed NoSQL databases // Інженерія програмного забезпечення і передові інформаційні технології (SoftTech-2022): матеріали тез доповідей III Всеукраїнської наук.-практ. конф. молодих вчених та студентів (м. Київ, 23–25 листопада 2022 р.). Секція кафедри інформатики та програмної інженерії. – К. : КПІ ім. Ігоря Сікорського, 2022. – С. 27 – 30. URL: https://drive.google.com/file/d/1CP9EaBTT_rJAXsINbanSVGnPP2jkg9FJ0/view ;

10. Белоус Р. В., **Нікітін В. А.** Варіанти забезпечення суворої узгодженості в NoSQL // Grail of Science, (24), с. 364 – 365. DOI: <https://doi.org/10.36074/grail-of-science.17.02.2023.065> .

ЗМІСТ

ЗМІСТ	10
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	14
ВСТУП.....	15
1 ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ УЗГОДЖЕННЯ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ.....	23
1.1 Використання розподілених баз даних в інформаційних системах	23
1.2 Властивості баз даних та поняття	25
1.2.1 ACID	25
1.2.2 BASE.....	26
1.2.3 Теорема CAP	28
1.2.4 Поняття транзакції та її обробка.....	30
1.2.5 Теорема PACELC	31
1.3 Узгодженість даних в розподілених базах даних.....	32
1.3.1 Суворі узгодженість.....	34
1.3.2 Послідовна узгодженість	35
1.3.3 Причинно-наслідкова узгодженість.....	37
1.3.4 Кінцева узгодженість	39
1.4 Існуючі методи та способи підтримки узгодженості	40
1.4.1 Архітектурні рішення.....	40
1.4.2 Протокол пліток	43
1.4.3. Логічні годинники	45
1.4.4 Механізм читання виправлення	48

1.4.5 Двофазова фіксація.....	49
Висновки до розділу 1	52
2 МЕТОДИ ТА ТЕХНОЛОГІЇ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ УЗГОДЖЕНОСТІ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ.....	54
2.1 Загальна структура розподіленої інформаційної системи	54
2.2 Розподілена база даних.....	56
2.3 Загальнопоширені алгоритми узгодження даних в інформаційних системах....	58
2.3.1 Механізм активної антиентропії	58
2.3.2 Механізми з використанням консенсусних протоколів та централізованими операціями записів	66
2.3.3 Полегшені транзакції	67
2.3.4 Потоки DynamoDB	69
2.4 Метод підвищення ефективності узгодженості даних за допомогою транзакційного годинника з пріоритизацією	70
2.4.1 Логіка роботи та математичний опис транзакційного годинника з пріоритизацією	71
2.4.2 Розробка транзакційного годинника з пріоритизацією.....	75
2.5 Модернізація процесу узгодженості даних за допомогою механізму активної антиентропії.....	77
2.5.1 Модифікований механізм активної антиентропії	79
2.5.2 Математична модель реплікації механізму активної антиентропії.....	84
2.5.3 Модифікований метод генерації спектрального фільтру Блума	85
2.5.4 Стійкий до колізій алгоритм хешування	89
Висновки до розділу 2	96

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНИХ МЕТОДІВ ТА ТЕХНОЛОГІЙ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ УЗГОДЖЕНОСТІ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ З РОЗПОДІЛЕНОЮ БАЗОЮ ДАНИХ	97
3.1 Розробка інтегрального середовища фінансової інформаційної системи для проведення експериментального дослідження	97
3.1.1 Бізнес-логіка інтегрального середовища фінансової інформаційної системи	98
3.1.2 Опис структурних складових системи	99
3.1.3 Проектування інтегрального середовища фінансової інформаційної системи	101
3.1.4 Структура проекту та реалізація серверного додатку.....	107
3.1.5 Створення бібліотеки для взаємодії із програмним інтерфейсом реалізованої інформаційної системи	109
3.1.6 Графічний інтерфейс реалізованого середовища	110
3.2 Програмна реалізація транзакційного годинника з пріоритезацією	112
3.2.1 Архітектура транзакційного годинника	112
3.2.2 Вибір необхідних засобів для реалізації транзакційного годинника	116
3.2.3 Тестування розробки	117
3.2.4 Математичний апарат для розрахунку часу узгодження	121
3.3 Програмна реалізація методу активної антиентропії.....	128
3.3.1 Проектування та програмна реалізація	128
3.3.2 Реалізація та тестування алгоритму хешування RH2-48	134
3.3.3 Реалізація та тестування модифікованого спектрального фільтру Блума ..	138

3.4 Експериментальні дослідження із застосуванням розробленої інформаційної системи.....	141
3.5 Розробка рекомендацій по впровадженню запропонованих методів	145
Висновки до розділу 3	148
ВИСНОВКИ.....	150
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	152
ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ В ТОВ “ІНТЕХФОРВАРД”	157
ДОДАТОК Б. АКТ ВПРОВАДЖЕННЯ В НАВЧАЛЬНИЙ ПРОЦЕС	158
ДОДАТОК В. РЕАЛІЗАЦІЯ ФОРМУВАННЯ СПЕКТРАЛЬНОГО ФІЛЬТРУ БЛУМА З ВИКОРИСТАННЯМ ПРОСТИХ ЧИСЕЛ.....	159
ДОДАТОК Г. РЕАЛІЗАЦІЯ АЛГОРИТМУ ХЕШУВАННЯ PH2-48.....	160
ДОДАТОК Е. DOCKER-COMPOSE ФАЙЛ ДЛЯ РОЗГОРТАННЯ РОЗПОДІЛЕНОЇ БАЗИ ДАНИХ.....	162
ДОДАТОК Д. КОМАНДИ ДЛЯ НАЛАШТУВАННЯ MONGODB REPLICASET....	165

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

2PC	– two-phase commit – протокол для виконання операцій на різних вузлах
5G	– fifth generation – п’яте покоління
ACID	– Atomicity, Consistency, Isolation, Durability – Атомарність, Узгодженість, Ізольованість, Довговічність
API	– Application Programming Interface – Прикладний Програмний Інтерфейс
BASE	– Basically Available, Soft state, Eventually consistent – Концетрування на доступності, легкому налаштуванні стану, забезпечення кінцевої узгодженості
CAP	– Consistency, Availability, Partition tolerance – Узгодженість, Доступність, Стійкість до розділу
CQL	– Cassandra Query Language – мова запитів в Cassandra
CRUD	– Create, Read, Update, Delete – основні операції з базою даних
DBMS	– Database Management System – система керування базою даних
FIFO	– First in, First out – принцип накопичення та обробки
IBM	– International Busyness Machines – американська корпорація
IoT	– Internet of Things – Інтернет речей
LWT	– lightweight transactions – спеціальний вид транзакцій в Cassandra
NoSQL	– non SQL – не реляційний підхід зберігання даних
PACELC	– Partition, Availability, Consistency, Else Latency, Consistency – Стійкість до розділу, Доступність, Узгодженість, Або Затримка, Узгодженість
SIEM	– security information and management – система керування подіями та інформаційною безпекою
TCP	– Transmission Control Protocol – протокол транспортного рівня моделі OSI
UDP	– User Datagram Protocol – протокол транспортного рівня моделі OSI
OSI	– The Open Systems Interconnection model – модель мережевої взаємодії
UTC	– Coordinated Universal Time – стандарт регулювання часу
СКБД	– система керування базою даних

ВСТУП

Актуальність теми. Діяльність людини має різні напрямки і будь-який з них насичений різноманітними даними та інформацією. Обчислювальна техніка дозволяє не тільки ефективно зберігати їх, але і швидко передавати на великі відстані. Технології передачі даних покращуються з кожним роком, надаючи все більше можливостей для використання. Наприклад, розгортання 5G технології надає можливості для широкого використання в області Інтернету речей. Тисячі датчиків можуть бути підключеними до мережі і генерувати гігабайти даних про будь-які існуючі об'єкти та явища [1].

Як приклад, можуть слугувати дослідницькі станції в Антарктиді. Існуюча мережа Sigfox має занадто низьку пропускну здатність. Навіть такі задачі як потокове відео або відправка фотографій, з якими ми стакаємось щодня, є проблематичними. Але було зроблено широке покриття навколо норвезької дослідницької станції Troll. Це відкриває нові горизонти для дослідницьких спільнот використовувати мобільну мережу для збору даних у польових умовах, досліджень та моніторингу навколишнього середовища в океані Антарктики. З іншого боку, збільшення кількості даних збільшує навантаження на інформаційні системи, які використовуються там, та вимагають впровадження додаткових методів узгодження даних [2].

Повертаючись до більш розповсюджених інформаційних систем слід зазначити фінансові інформаційні системи, з якими стикалась кожна людина. Вимоги до таких систем високі і вимагають максимального забезпечення швидкодії та надійності. Особливо важливим є зберігання та опрацювання фінансових транзакцій, які повинні зберігатися роками для встановлення причинно-наслідкового зв'язку між фінансовими операціями [3].

Також, як приклад можна навести підсистеми, які використовуються в кібербезпеці. Інформаційні системи захищаються спеціалістами, які використовують необхідні програмні засоби для цього. Одним з таких засобів є так звані SIEM, які

використовуються для управління інформаційною безпекою та подіями. Вона забезпечує аналіз даних в реальному часі, які були отримані від мережевих пристроїв та додатків. Однією з вимог є довготривале зберігання узгоджених даних з різних джерел впорядкованих за глобальним часом. Це необхідно, оскільки проведення комп'ютерно-технічних експертиз відбувається через часову затримку від моменту інциденту [4].

Вищезазначені приклади – це лише маленька частина всього можливого різноманіття інформаційних систем, кожна з яких має свої вимоги, задачі та специфіку роботи.

Поступове зростання інформаційної системи вимагає оптимізації, оскільки збільшення навантаження призводить до погіршення роботи, що негативно впливає на взаємодію кінцевого користувача з нею. Наприклад, до цього можуть бути дуже чутливі брокерські інформаційні системи, для яких дуже важливі останні зміни у котируванні акцій.

Це може вирішуватись декількома шляхами: вертикальним або горизонтальним. Вертикальне масштабування вимагає заміни апаратної частини серверу. В деяких ситуаціях це можливо і є рішенням, але залишається питання часу, коли доведеться повторити процедуру. Горизонтальне масштабування вирішує проблему навантаження шляхом розподілу навантаження між декількома екземплярами сервісу. Це тягне за собою обов'язкове створення розподіленої бази даних, оскільки при наявності лише одного екземпляру бази даних, така система може працювати так само або гірше з точки зору доступності через мережеві затримки та навантаженням на опрацювання запитів [5].

Розподілена база даних вирішує вищезазначені проблеми, але створює нові, які пов'язані з актуальністю версій даних. Операції запису вимагають впровадження спеціальних методів та способів підтримки узгодженості системи. Неузгоджені дані негативно впливають на виконання функцій, покладених на інформаційну систему. Кінцеві користувачі не можуть приймати правильних рішень оперуючи

неактуальними даними, що може призводити до матеріальних, а в деяких випадках, і до людських втрат [6].

З іншого боку, впровадження таких механізмів може впливати на доступність системи, оскільки для забезпечення найвищого рівня узгодженості потрібно виконувати спеціальні дії, які можуть блокувати новоутворені операції. Блокування обумовлює певну затримку, під час якої, кінцевому користувачу потрібно чекати.

Таким чином, підтримка узгодженості даних в інформаційній системі є важливою задачею, для якої необхідно шукати та застосовувати нові способи та методи, оскільки кількість інформаційних систем та об'єктів, які створюють дані та інформацію, збільшуються.

Зв'язок роботи з науковими програмами, планами, темами.

Наукове дослідження за темою дисертації проводилось у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» у відповідності до Переліку пріоритетних тематичних напрямів наукових досліджень і науково-технічних розробок на період до 2025 року. Робота виконувалась в рамках Ініціативної теми “Оптимізація роботи веб-орієнтованих систем з великим об'ємом даних” з державним реєстраційним номером 0117U004913. Пріоритетним напрямом науково-технічної діяльності є “Інформаційні та комунікаційні технології”. Галуззю застосування є дослідження та розробки в галузі природничих та технічних наук.

Мета і завдання дослідження.

Метою дисертаційної роботи є підвищення ефективності процесу узгодження даних в розподілених інформаційних системах через розробку нових методів та можливість їх впровадження в роботу існуючих інформаційних систем. Для досягнення мети поставлені та вирішені такі задачі:

- огляд розвитку баз даних від реляційних до розподілених нереляційних,

- аналіз існуючих механізмів узгодження даних в реляційних та нереляційних базах даних, можливі способи комунікації у розподілених системах, аналіз моделей узгодженості даних та методів синхронізації;
- розробка методу узгодження даних з використанням транзакційного годинника з пріоритезацією та його подальшою реалізацією для розподіленої нереляційної документо-орієнтованої бази даних;
- розробка способу активної антиентропії на основі протоколу пліток та використанні спектрального фільтру Блума та алгоритму хешування з подальшою реалізацією для розподіленої нереляційної документо-орієнтованої бази даних;
- модифікація алгоритму формування спектрального фільтру Блума з використанням простих чисел для використання в розробленому способі активної антиентропії;
- розробка способу хешування даних з підвищеною стійкістю до колізій з подальшим використанням в розробленому способі активної антиентропії;
- проєктування та реалізація прототипу фінансової інформаційної системи для проведення дослідів.

Об’єкт дослідження – процес узгодження даних у розподілених нереляційних документо-орієнтованих базах даних.

Предмет дослідження – розробка нових методів та алгоритмів підвищення швидкодії та колізійної стійкості процесу узгодження даних у розподілених нереляційних базах даних.

Методи дослідження – теорія алгоритмів, теорія ймовірностей та математичної статистики, методи об’єктно-орієнтованого та процедурного програмування, емпіричний метод.

Наукова новизна отриманих результатів:

- удосконалено метод узгодження даних з використанням транзакційного годинника, яка дозволяє підвищити швидкодію синхронізації критично важливих даних в розподілених інформаційних системах з великою кількістю операцій запису за рахунок введення пріоритетності транзакцій та використання транзакційної логіки;
- удосконалено процес активної антиентропії, який відрізняється від існуючого використанням алгоритму хешування РН2 замість дерева Меркла та використанням модифікованого спектрального фільтра Блума, що дозволяє підвищити швидкодію процесу узгодження даних в розподілених інформаційних системах;
- модифіковано спосіб формування спектрального фільтра Блума для пришвидшення розробленого способу активної антиентропії;
- вперше розроблено метод хешування на основі простих чисел РН2, який відрізняється від відомих CRC32 та MurmurHash3 та дозволяє збільшити колізійну стійкість процесу узгодження даних в розподілених інформаційних системах;
- набула подальшого розвитку інформаційна технологія, яка ґрунтується на комплексному застосуванні запропонованих методів та моделей, що дозволяє підвищити швидкодію та колізійну стійкість.

Практичне значення отриманих результатів:

- реалізовано транзакційний годинник з використанням мови програмування Python3 для застосування в розподіленій базі даних на основі MongoDB;
- реалізовано розроблений спосіб активної антиентропії з використанням мови програмування Python3 для застосування в розподіленій базі даних на основі MongoDB;

- реалізовано програмну бібліотеку з модифікованим спектральним фільтром Блума та створеним алгоритмом хешування з використанням мови програмування Python3, яке дозволяє застосувати отримані наукові результати при реалізації інформаційних систем;
- розроблено прототип фінансової інформаційної системи для дослідження методів та способів узгодження даних з використанням розподіленої бази даних.

Особистий внесок здобувача. Усі результати, що виносяться автором до захисту дисертаційної роботи, отримані особисто у процесі науково-дослідницької роботи. У наукових працях, опублікованих у співавторстві, автору належать:

- [9] – у статті розглядаються основні методи індексування, які використовуються в найбільш поширених нереляційних базах даних. Ця робота базується на огляді та аналізі літератури з оптимізації баз даних з використанням алгоритмів заснованих на бінарних деревах та який це робить вплив на інформаційну систему;
- [34] – публікація містить теоретичне та практичне підґрунтя щодо розподілених баз даних. Вона містить один із способів для визначення кількості вузлів, при яких буде забезпечуватись максимальна ефективність роботи інформаційної системи;
- [36] – у статті порівнюється алгоритм на основі ділення та криптографічні хеш-функції MD5 та SHA-1 з метою модифікування існуючих методів узгодження даних інформаційних систем, які використовують розподілені бази даних;
- [38] – у статті розглядаються найпоширеніші алгоритми хешування та запропонований алгоритм хешування на основі простих чисел і двійкової системи числення. Ця робота базується на основній теоремі арифметики, яка стверджує можливість розкладання будь-якого натурального числа на

множини з унікальним набором простих чисел. Такий підхід дає можливість використовувати математичний апарат для обґрунтування властивостей алгоритму, що є важливим при вирішенні задачі узгодження даних;

- [46] – у статті описується транзакційний годинник, його логіка роботи та математичний апарат. Приводяться приклади його використання та обґрунтування його використання в підтримки узгодженості даних інформаційних систем з розподіленою базою даних;
- [47] – у статті детально описано удосконалений метод активної антиентропії на основі протоколу пліток та використання спектрального фільтру Блума та алгоритму хешування для підтримки узгодженості в інформаційній системі;
- [48] – у статті розписан алгоритм формування спектрального фільтру Блумана основі простих чисел, проведено експериментальні дослідження його властивостей та порівняно з класичним способом формування;
- [49] – у статті розписано алгоритм, який формує стійке до колізій хеш-значення при хешуванні даних різних за розміром. Наведені приклади хешування та колізійні ситуації.

Апробація результатів дисертації. Основні результати досліджень, що включені до дисертаційної роботи, доповідались і обговорювались на конференціях:

- Третя Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (22.11.2022 – 25.11.2022 року, Київ);
- V Міжнародна науково-практична конференція «SCIENTIFIC RESEARCHES AND METHODS OF THEIR CARRYING OUT: WORLD EXPERIENCE AND DOMESTIC REALITIES» (17.02.2023, Вінниця).

Публікації. За результатами дисертаційних досліджень опубліковано 8 наукових публікацій, серед яких:

- 1 публікація, яка проіндексована в Scopus базі даних;
- 7 статей у фахових наукових виданнях категорії «Б», одна з яких проіндексована в «Index Copernicus International» (Республіка Польща);

Структура та обсяг роботи. Дисертація складається зі вступу, 3 розділів, висновків, списку використаних джерел з 49 найменувань та 6 додатків. Загальний обсяг роботи складає 165 сторінок, з них 133 сторінки основного тексту, 76 рисунків та 5 таблиць.

1 ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ УЗГОДЖЕННЯ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ

1.1 Використання розподілених баз даних в інформаційних системах

Організації покладаються на інформаційні системи з багатьох причин, і вони відіграють важливу роль у кількох контекстах: управління операціями, взаємодія з клієнтами, прийняття рішень, співпраця та командна робота, стратегічні ініціативи та індивідуальна продуктивність.

Дані, інформація та знання – це терміни, які відображають те, як необроблені факти можна об'єднати, зібрати й проаналізувати, щоб додати сенс і цінність. Характеристики інформації, які додають її цінності, включають своєчасність, точність і повноту.

Чотири компоненти будь-якої інформаційної системи включають: люди, технології, процеси та дані. Охоплюється увесь людський ресурс і залучає людей з багатьох різних частин організації. Замовники та постачальники також беруть участь у вдосконаленні процесів та усуненні помилок.

Інформаційні системи сприяють успіху в кожному функціональному відділі та в усіх типах організацій. Хоча інформаційні системи мають надзвичайні перспективи, вони також представляють ризики та етичні проблеми. Порушення конфіденційності трапляються часто, і може завдати суттєвих збитків. Ризик зіпсування репутації дуже великий, оскільки швидкість поширення інформації може бути дуже швидкою [7]. Для зберігання та обробки даних використовуються спеціальне програмне забезпечення.

База даних — це організований набір структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді в комп'ютерній системі. База даних зазвичай контролюється системою керування базами даних. Разом дані та СКБД разом із додатками, які з ними пов'язані, називають системою баз даних, яку часто скорочують до просто бази даних [8].

Вони дозволяють зберігати та обробляти необхідні дані в інформаційних системах, спрощуючи тим самим взаємодію потенційних користувачів з сервісом. Банальна інформація, яку ми використовуємо для авторизації також зберігається певним чином у базі даних, що спрощує механізми авторизації та автентифікації. Для оптимізації пошуку використовуються різномінітні дерева, такі як B-Tree, T-Tree, O₂-Tree [9].

Розподіл даних між вузлами системи не є додатковою можливістю, а є потребою задля забезпечення високих показників швидкодії та надійності. Це дозволяє не тільки рівномірно розподілити навантаження клієнтських запитів, але й підвищити доступність системи загалом через наявність реплік, які можуть працювати в той момент, коли інші стають недоступними.

NoSQL — це підхід до керування базами даних, який може включати широкий спектр моделей даних, у тому числі ключ-значення, документ, стовпець і графічний формат. База даних NoSQL зазвичай означає, що вона нереляційна, розподілена, гнучка та масштабована. Додаткові загальні функції бази даних NoSQL включають відсутність схеми бази даних, кластеризацію даних, підтримку реплікації та можливу узгодженість, на відміну від типової узгодженості транзакцій ACID (атомарність, узгодженість, ізоляція та довговічність) реляційних баз даних [10].

Загалом можна виділити чотири типи систем баз даних NoSQL. Кожен використовує свій тип моделі даних, що призводить до значних відмінностей між ними.

- Документо-орієнтовані бази даних, які також називаються сховищами документів, зберігають напівструктуровані дані та описи цих даних у форматі документа. Вони дозволяють розробникам створювати й оновлювати програми без необхідності спиратися на головну схему. Використання баз даних документів збільшилося разом із використанням JavaScript і JavaScript Object Notation (JSON), формату обміну даними, який набув широкого поширення серед розробників веб-додатків. Couchbase

Server, CouchDB, MarkLogic і MongoDB є прикладами документо-орієнтованих баз даних;

- Графові бази даних організовують дані як вузли, які схожі на рядки в реляційній базі даних, і ребра, які представляють зв'язки між вузлами. Оскільки система графів зберігає зв'язки між вузлами, вона може підтримувати більш багате представлення зв'язків даних. До таких баз даних відносяться AllegroGraph, IBM Graph і Neo4j;
- Сховища ключ-значення також відомі як бази даних ключ-значення, ці системи реалізують просту модель даних, яка поєднує унікальний ключ із пов'язаним значенням. Оскільки ця модель проста, її можна використовувати для розробки високомасштабованих і продуктивних програм. До таких баз даних відносяться Aerospike, DynamoDB, Redis;
- Колонкові бази даних використовують знайомі таблиці, стовпці та рядки, як таблиці реляційної бази даних, але назви стовпців і форматування можуть відрізнятися від рядка до рядка в одній таблиці. Accumulo, Amazon SimpleDB, Cassandra, HBase та Hypertable є прикладами баз даних із широкими колонками [11].

1.2 Властивості баз даних та поняття

Кожна розподілена база даних представляє собою цілу підсистему, яка має свої правила функціонування. Це, у свою чергу, впливає на властивості, які характеризують роботу клієнтів з базою даних.

Вони можуть відноситись безпосередньо до цілісності даних під час зберігання, швидкість читання, швидкість поширення оновлень та інше.

1.2.1 ACID

ACID – це аббревіатура, що означає атомарність, консистентність, ізолюваність, довговічність. У системах баз даних ACID відноситься до стандартного набору

властивостей, які гарантують надійну обробку транзакцій бази даних. ACID особливо корисний коли база даних відновлюється після будь-якої помилки, яка може статися під час обробки транзакції. База даних, сумісна з ACID, гарантує, що дані в базі даних залишаються точними та узгодженими, незважаючи на будь-які такі збої [12].

- Атомарність гарантує, що або всі транзакції будуть успішними, або жодна з них. Транзакція може бути виконана повністю, а частково - ні. Якщо одна частина транзакції зазнає невдачі, вся транзакція зазнає невдачі;
- Узгодженість гарантує, що на всіх вузлах бази даних будуть знаходитись актуальні версії даних. Тобто, під час запису на один вузол, всі вузли системи отримують нову версію через певний проміжок часу;
- Ізоляція гарантує, що всі транзакції відбуватимуться ізольовано. Одні транзакції не можуть вплинути на інші, а читання результату іншої транзакції відбувається тоді і тільки тоді, коли вона завершена;
- Довговічність гарантує, що після здійснення транзакції вона залишатиметься в системі, навіть якщо відразу після транзакції станеться збій системи. Будь-які зміни транзакції повинні зберігатися стабільно. Якщо клієнт отримав повідомлення, що транзакція виконана успішно, то транзакція насправді має бути успішною.

Усі основні реляційні бази даних дотримуються принципів ACID, але розробники нереляційних баз даних, такі як MarkLogic, OrientDB і Neo4j, пропонують сумісні з ACID системи керування базами даних NoSQL [12].

1.2.2 BASE

Принципи BASE були представлені як альтернативний підхід до проектування системи. Введений Деном Прітчеттом у 2008 році термін «BASE» набув популярності та викликав дискусії навколо послаблення узгодженості заради масштабованості та

доступності. Пом'якшуючи гарантії узгодженості, системи BASE пріоритезують доступність, дозволяють змінювати стани даних і досягають узгодженості з часом [13].

Суворе дотримання принципів ACID може перешкоджати продуктивності та масштабованості у сильно розподілених і масштабованих середовищах. Тут вступають у дію принципи BASE. Принципи BASE можуть допомогти приймати обґрунтовані рішення при проектуванні масштабованих і високодоступних систем. Вони встановлюють пріоритети доступності, пристосовання до змінних станів даних і досягають узгодженості з часом (рис. 1.1) [13].

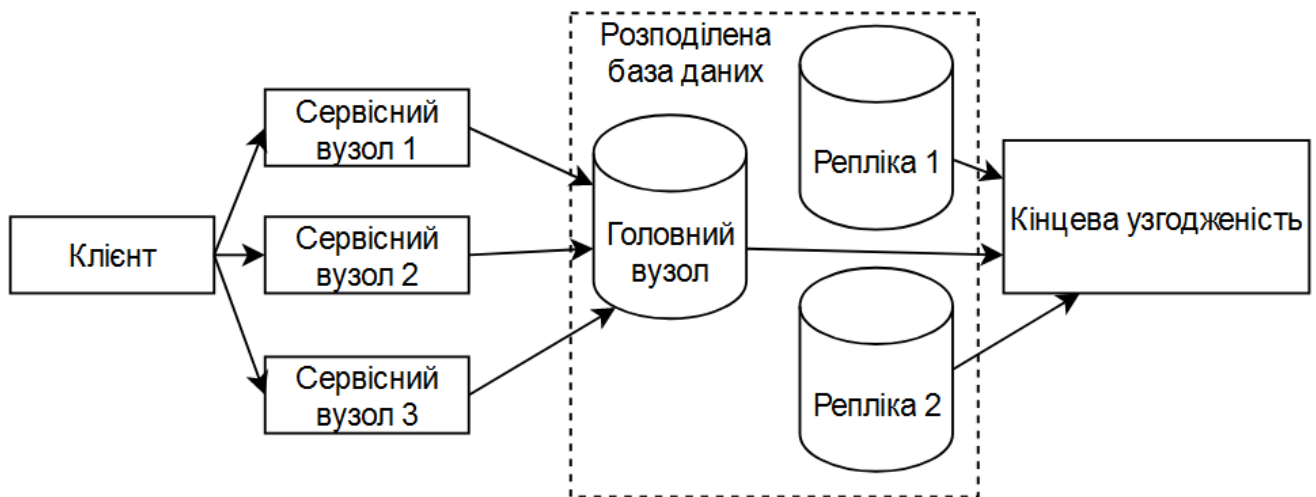


Рисунок 1.1 – Приклад архітектури із застосуванням принципів BASE

Особливості використання цих принципів:

- У розподілених системах підтримка високого рівня доступності є надзвичайно важливою. Такі системи надають перевагу доступності над високою узгодженістю, тобто система залишається працездатною, незважаючи на можливі збої або розділення мережею. Замість того, щоб зупиняти роботу під час збою, системи BASE забезпечують базовий рівень функціональності, щоб забезпечити безперервне використання та доступність для клієнтів;

- Це означає, що дані системи не завжди узгоджені і можуть перебувати в проміжних станах у системах BASE. Такий стан дозволяє системам бути більш гнучкими, вміщуючи одночасні оновлення та забезпечуючи кращу продуктивність і масштабованість;
- На відміну від миттєвої узгодженості ACID, системи BASE віддають пріоритет кінцевій узгодженості. За умови достатнього часу без оновлень або конфліктів усі репліки в розподіленій системі в кінцевому підсумку зійдуться до одного стану [13].

1.2.3 Теорема CAP

Теорема CAP (або теорема Брюера) є ствердженням, що розподілена система може забезпечити лише дві з трьох бажаних характеристик: узгодженість, доступність і толерантність до розділів. Розподілена система — це система, яка складається з певного набору вузлів (фізичних або віртуальних машин), кожен з яких зберігає дані. Теорему CAP також називають теоремою Брюера, оскільки її вперше запропонував професор Ерік А. Брюер під час виступу про розподілені обчислення у 2000 році [14].

- Узгодженість означає, що всі клієнти бачать однакові дані одночасно, незалежно від того, до якого вузла вони роблять запити. Щоб це сталося, щоразу, коли дані записуються на один вузол, вони повинні бути миттєво репліковані на всі інші вузли в системі;
- Доступність означає, що будь-який клієнт, який робить запит на дані, отримує відповідь, навіть якщо один або кілька вузлів не працюють. При такій організації, всі робочі вузли в розподіленій системі повертають адекватну відповідь на будь-який запит без винятку;
- Стійкість до розділу означає, що кластер повинен продовжувати працювати, незважаючи на будь-яку кількість порушень зв'язку між вузлами в системі через мережеві збої [14].

Бази даних NoSQL ідеально підходять для розподілених систем. На відміну від своїх вертикально масштабованих реляційних аналогів, бази даних NoSQL є горизонтально масштабованими та розподіленими за необхідністю — вони можуть швидко масштабуватись у зростаючій мережі, що складається з кількох взаємопов'язаних вузлів. На даний час, бази даних NoSQL поділяються на основі двох характеристик CAP, які вони підтримують (рис. 1.2):

- Бази даних CP забезпечують узгодженість і стійкість до розділів через погіршення доступності. Коли виникає проблема зв'язку між будь-якими двома вузлами, система повинна відокремити неузгоджений вузол (тобто зробити його недоступним), доки проблема не буде вирішена;
- Бази даних AP забезпечують доступність і стійкість до розділів через погіршення узгодженості. Коли виникає проблема зв'язку між двома вузлами, усі вузли залишаються доступними, але вони під впливом цього фактору можуть повертати неузгоджені дані. Коли проблема вирішена, бази даних AP зазвичай повторно синхронізують вузли, щоб виправити всі невідповідності в системі;
- Бази даних CA забезпечує узгодженість і доступність на всіх вузлах. Однак вона не може зробити це, якщо між будь-якими двома вузлами в системі є розділ, і тому не може забезпечити стійкість до розділення [15].

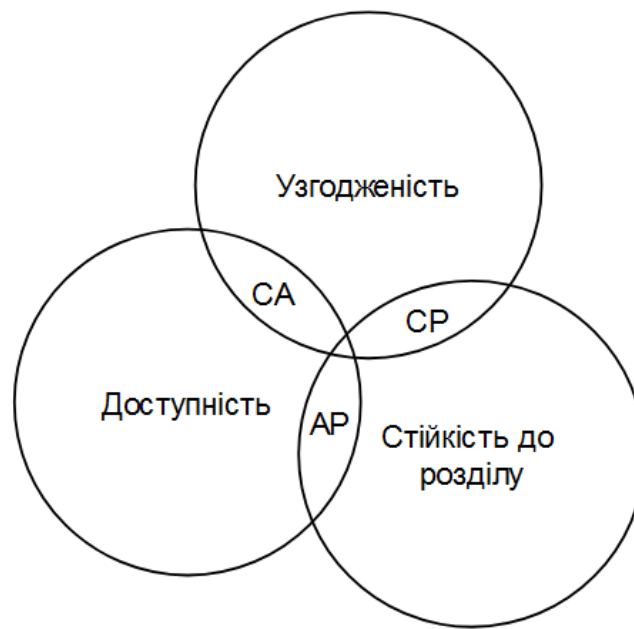


Рисунок 1.2 — CAP теорема

1.2.4 Поняття транзакції та її обробка

Обробка транзакцій є фундаментальною концепцією управління базами даних та інформаційних систем. Це стосується процесу управління та виконання послідовності операцій бази даних, відомого як транзакція, надійним, послідовним та атомарним (неподільним) способом. Системи обробки транзакцій (TPS) мають вирішальне значення в різних програмах і галузях, де важливі цілісність даних, надійність і контроль паралельності [16].

Основні характеристики та принципи обробки транзакцій:

- Транзакції є атомарними, тобто вони розглядаються як єдина неподільна одиниця роботи;
- Транзакції переводять базу даних з одного узгодженого стану в інший;
- Транзакції виконуються ізольовано одна від одної;
- Після завершення транзакції її зміни є постійними та залишаються в силі після системних збоїв (наприклад, відключення електроенергії чи апаратного збою);

- Системи обробки транзакцій дозволяють одночасно виконувати кілька транзакцій, щоб підвищити продуктивність системи;
- Транзакції часто описуються за допомогою властивостей ACID [16].

1.2.5 Теорема PACELC

Теорема PACELC є розширенням теореми CAP. Одне із запитань, на яке CAP теорема не може дати відповідь, це ситуація коли система не має розділу. Через це до трьох основних властивостей додається затримка. Акронім PACELC означає Partitioned, Availability, Consistency Else Latency, Consistency. Ця теорема стверджує, що у випадку мережевого розділу «Р» розподілена система може мати компроміси між доступністю «А» та узгодженістю «С». Інакше «Е», якщо немає мережевого розділу, розподілена система може мати компроміси між затримкою «L» і узгодженістю «С» (рис. 1.3) [17].

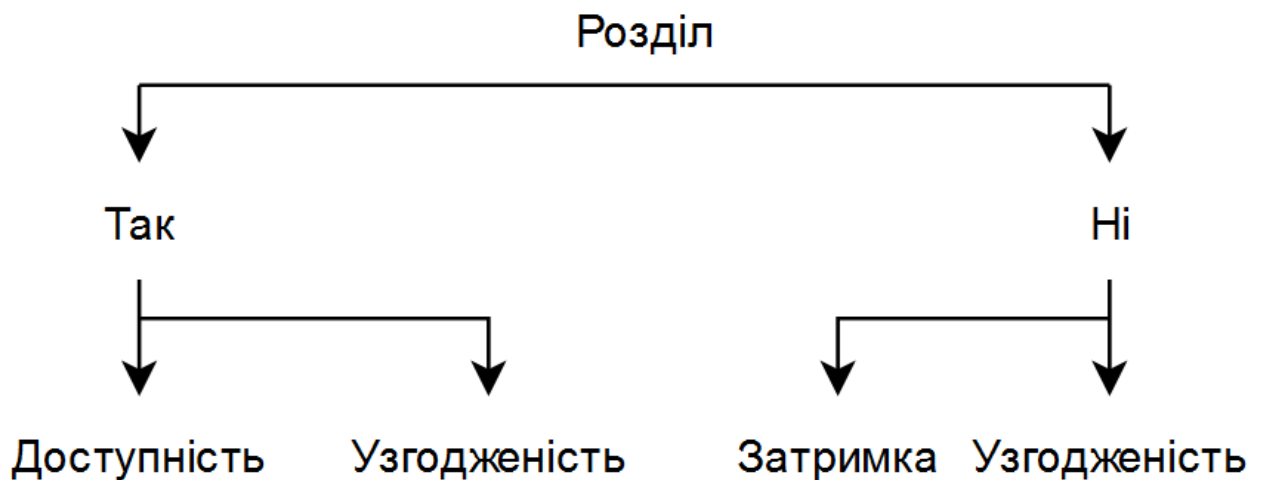


Рисунок 1.3 — PACELC теорема

Одним із головних недоліків теореми CAP є те, що вона не враховує продуктивність чи затримку. Інакше кажучи, теорема CAP не містить компромісів, коли система нормально функціонує або не розділена. В якості прикладу можна навести ситуацію, коли є система і людина робить реєстрацію в ній, але відповідь

приходить через добу. Розподілена система доступна, узгоджена, але затримка є занадто великою. Саме такі ситуації підштовхнули науковців до перегляду існуючого бачення та врахування таких нюансів. Перші ідеї були опубліковані у 2010 році в якості посту у блозі. У 2012 році вийшла повноцінна стаття і лише у 2018 році дана теорема була формально доведена [17].

Таким чином, теорема PACELC розглядає компроміси затримки та узгодженості, які завжди були поширеними. Це допомагає зосередитись над ефективним способом вибору методів та проектуванням розподілених систем.

1.3 Узгодженість даних в розподілених базах даних

Для клієнтів відсутні проблеми при роботі з єдиним екземпляром бази даних, оскільки операції запису відбуваються під контролем та передача даних відбувається в межах одного єдиного процесу. До таких баз даних можна віднести всі ті, які відносяться до СА типу за CAP теоремою. Реалізація однієї лінеаризованої черги, в якій чітко визначен порядок тієї чи іншої операції, не складає особливих складнощів в контексті забезпечення узгодженості.

У випадку, коли ми маємо вузли з різним географічним розташуванням та станом мережі, то збереження послідовності виконання транзакцій створює ряд задач та моментів для розмірковування.

Узгодженість даних означає властивість того, що значення даних однакові на всіх вузлах у розподіленій системі, або що будь-які зміни в даних поширюються та відображаються своєчасно та узгоджено. Узгодженість даних гарантує, що система поводить ся так, як очікувалося, і що дані є дійсними та значущими для користувачів інформаційної системи [18].

Узгодженість даних важлива, оскільки вона впливає на продуктивність, доступність і коректність розподіленої системи. Без узгодженості даних система може видавати неактуальні дані, втратити дані або порушити бізнес-логіку. Вона також

дозволяє користувачам і програмам-клієнтам довіряти даним і приймати рішення на їх основі.

Досягнення узгодженості даних у розподілених інформаційних системах є складним завданням через кілька факторів, таких як затримки та збої мережі, які можуть ускладнити синхронізацію та оновлення даних. Паралельність і конфлікти також можуть виникати, коли кілька вузлів намагаються отримати доступ до тих самих даних або змінити їх одночасно, що призводить до конфліктних або неузгоджених версій даних. Крім того, реплікація та розділення даних між різними вузлами для масштабованості та відмовостійкості може збільшити складність і вартість під час спроби підтримки узгодженості даних.

Узгодженість даних у розподілених системах визначається різними моделями, які визначають правила та гарантії оновлення та доступу до даних. Кожна модель тим чи іншим способом впливає на доступність та продуктивність системи. Від цього залежить час відповіді клієнту [18].

Узгодженість даних у розподілених системах може бути досягнута та підтримувана за допомогою різних методів, залежно від вимог до системи та обраної моделі. Транзакції забезпечують набір операцій, які виконуються атомарно, послідовно, ізольовано та довгостроково (ACID), таким чином забезпечуючи узгодженість даних до та після виконання. Кворуми — це підмножини вузлів, які повинні узгодити значення даних або операції, що забезпечує баланс між узгодженістю даних і доступністю системи. Розв'язання конфліктів — це процес, який використовується для вирішення розбіжностей між різними версіями даних системою. Це може базуватися на таких критеріях, як мітки часу, версії або реалізований спеціалізований алгоритм [19].

Вибір найкращого підходу узгодженості даних для розподілених систем не є універсальним рішенням, оскільки різні підходи мають свої переваги та недоліки. Найбільш прийнятний підхід залежить від цілей системи, характеристик і компромісів. Вибираючи підхід, потрібно враховувати такі фактори, як тип і значення даних, їх

розмір і розподіл, а також частоту й затримку. Наприклад, числові або фактичні дані можуть потребувати більшої узгодженості, ніж текстові; великі або географічно розкидані дані можуть вимагати меншої узгодженості, ніж невеликі або локалізовані дані; і часті або чутливі до часу дані можуть потребувати більшої узгодженості, ніж рідкісні або нечутливі до часу дані [20].

Модель узгодження даних повинна оптимально вирішувати дану задачу виходячи із вимог системи. Та чи інша модель дозволяє підтримувати необхідний рівень узгодженості у розподіленій базі даних. Згідно CAP теорему, неможливо досягнути максимально швидкого поширення даних між вузлами, оскільки це може впливати на інші властивості [21].

Іншими словами, кожна модель визначає певний рівень узгодженості даних в розподіленій базі даних та дозволяє оцінити потенційні проблеми, які можуть виникнути при використанні. Слід зауважити, що кожна модель не визначає однозначний перелік методів та способів для досягнення поставленої мети та характеризує поведінку системи загалом.

1.3.1 Суворі узгодженість

Суворі узгодженість гарантує, що кожна операція читання в розподіленій базі даних повертає останнє значення даних. Іншими словами, будь-яка операція читання після операції запису відображатиме оновлене значення. Ця модель узгодженості забезпечує той самий рівень узгодженості, що й система з одним вузлом, але її може бути складно досягти в розподілених системах через компроміси між узгодженістю та доступністю. Якщо бізнес-логіка системи допускає обмежену доступність даних, то це цілком реально [21].

Основні характеристики суворі узгодженості включають:

- можливість лінеаризації для лінійного впорядкування таким чином, ніби вони відбуваються в одній безперервній і миттєвій послідовності. Ця лінеаризація відповідає порядку операцій у реальному часі;

- всі вузли в розподіленій системі дотримуються однакового глобального порядку операцій. На цей порядок не впливають фізичні або мережеві затримки між вузлами;
- результат операції запису одразу видно всім вузлам, щойно операція підтверджується. Це означає, що після завершення запису всі наступні операції читання бачать оновлені дані.

Незважаючи на те, що сувора узгодженість забезпечує найвищий рівень узгодженості даних і спрощує міркування про розподілені системи, вона часто має значні витрати з точки зору продуктивності та доступності. Досягнення суворої узгодженості вимагає надійних механізмів координації, таких як розподілені блокування та операції запису на основі кворуму, які можуть викликати затримку та зменшити доступність системи [21].

Як наслідок, сувора узгодженість зазвичай використовується помірковано в розподілених системах, особливо в сценаріях, де це абсолютно необхідно, наприклад, у певних фінансових додатках або коли підтримання суворої узгодженості є юридичною вимогою.

1.3.2 Послідовна узгодженість

Послідовна модель узгодженості є рівнем нижче лінеаризації. У лінеаризації ми вимагаємо повного порядку всіх операцій, що відповідає реальним годинникам. У послідовній узгодженості ми відмовляємось від узгодженості даних в логічному сенсі.

Логіку роботи зображено на рис. 1.4.

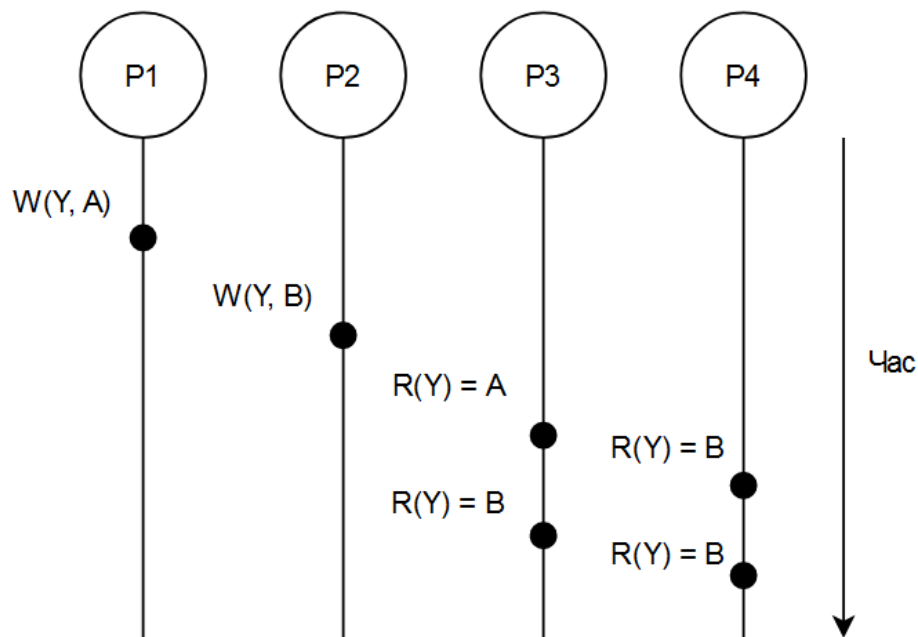


Рисунок 1.4 – Логіка роботи послідовної узгодженості

Нехай у нас є розподілена база даних, яка складається з чотирьох вузлів. На кожному вузлі є процес P , який може записути та читати певну ділянку пам'яті. Узгодженість забезпечується певним механізмом, який відноситься до послідовної моделі.

При використанні такої моделі, інші процеси можуть бачити не останню версію даних. Вузли P_1 та P_2 отримують нові значення, які асоціюються зі значенням Y . Операції читання не блокуються, поки працює механізм узгодження. Через це процес P_3 при читанні отримує значення A , а через певний проміжок часу P_3 та P_4 отримує вже B . Але ключовим моментом є те, що всі процеси обробляють операції в однаковому порядку, не залежно від вузла [22].

Тобто суть послідовної моделі у тому, щоб узгодити дані згідно порядку отриманих оновлень, але операції від клієнтів не блокуються.

На рис. 1.5 зображено приклад роботи системи, коли досягається узгодженість, але не послідовно. Процеси читають різні значення, бо система не є загальноузгодженою і порядок оновлень не впорядковується.

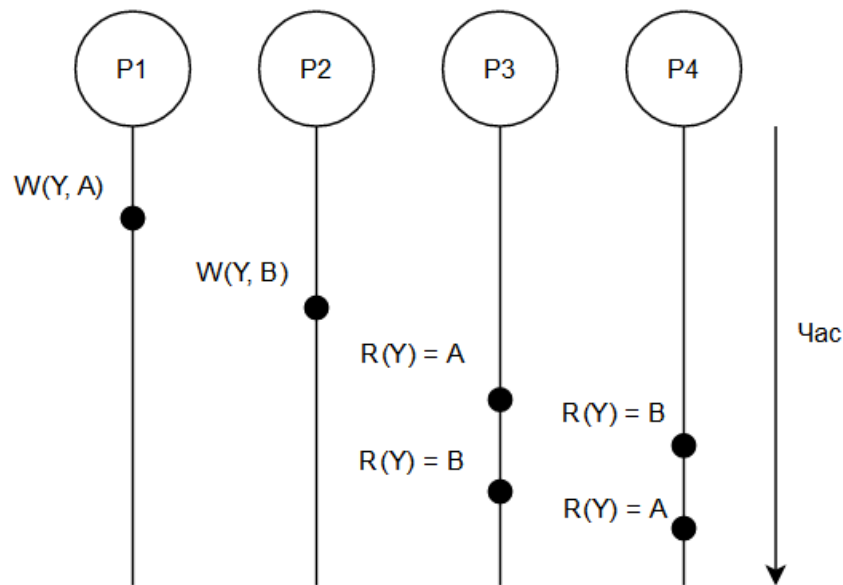


Рисунок 1.5 – Приклад ситуації слабшої моделі узгодження

1.3.3 Причинно-наслідкова узгодженість

Причинно-наслідкова узгодженість – це слабка форма узгодженості, яка зберігає порядок причинно пов'язаних операцій. В цьому випадку, система бачить загалом взаємопов'язані операції. Тобто відбувається встановлення залежності однієї події від іншої.

Іншими словами, якщо a відбувається перед b , то a має виконуватися перед b на всіх репліках. Усі інші одночасні операції можна побачити в іншому порядку.

Ключова відмінність між причинно-наслідковою та послідовною узгодженістю полягає в тому, що причинно-наслідкова узгодженість не потребує повного порядку.

Такий вид узгодженості передбачає, що зчитування оновлених даних відбувається лише в тому випадку, якщо операції причинно залежні від цих операцій записів. Крім того, лише причинно-наслідкові записи впорядковані всіма репліками однаково. Паралельні записи можуть виконуватися в різних порядках [23].

У розподіленій системі процеси можуть виконуватися одночасно та спілкуватися, надсилаючи повідомлення один одному. Коли процес оновлює спільний фрагмент даних, важливо встановити причинно-наслідковий зв'язок між цим

оновленням і будь-яким подальшим доступом до даних іншими процесами.

Причинно-наслідкова узгодженість гарантує збереження причинно-наслідкових зв'язків між подіями (оновленнями та доступами). Якщо подія А причинно залежить від події В, то будь-який процес, який спостерігає подію А, також повинен спостерігати подію В перед нею. Однак події, які є причинно незалежними, можна спостерігати в будь-якому порядку.

Причинно-наслідкова послідовність забезпечує баланс між сильною послідовністю та остаточною послідовністю. Це дозволяє одночасне виконання та забезпечує поняття причинно-наслідкового зв'язку, гарантуючи, що процеси спостерігають за оновленнями в послідовному порядку, коли існує причинно-наслідковий зв'язок.

Реалізація причинно-наслідкової узгодженості в проєкті системи вимагає ретельного поводження з подіями та їх залежностями. Для встановлення та відстеження причинно-наслідкових зв'язків між подіями можна використовувати такі методи, як векторний годинник або відстеження залежностей. Ці методи допомагають гарантувати, що процеси спостерігають події в причинно-наслідковому порядку [23].

Дві операції є причинно-наслідковими, якщо ми можемо сказати, що одна сталася раніше іншої. Будь-яка система, яка реалізує причинно-наслідкову узгодженість, гарантує збереження цього порядку в усіх репліках. Якщо операція а відбувається перед операцією b, жодна репліка не повинна побачити ефект операції b до того, як побачить ефект операції а. Тут ми говоримо, що операція b причинно залежить від операції а або а є залежністю від b.

Така модель обумовлює впорядкування операцій, які причинно-залежні. Якщо на двох різних вузлах відбулися операції запису, то вони будуть виконані в цьому самому порядку, але перед/між/після можуть бути виконані інші операції (рис. 1.6).

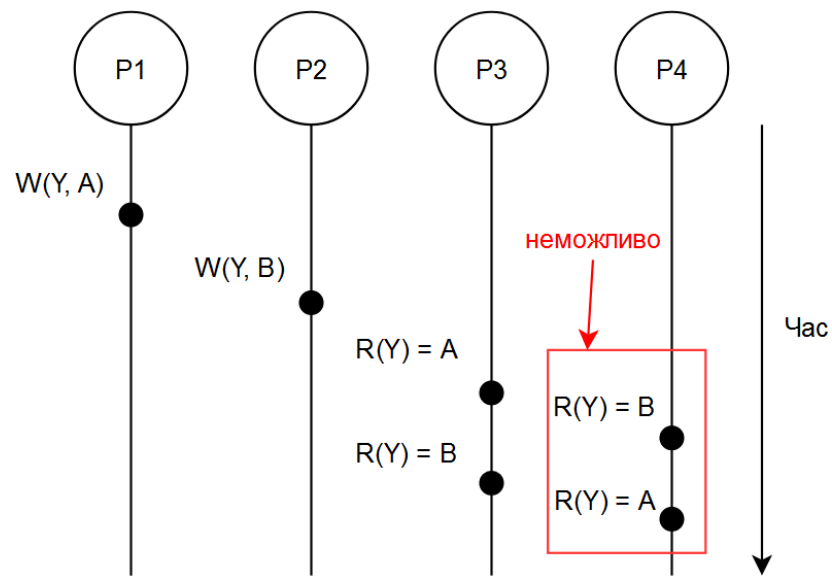


Рисунок 1.6 – Приклад роботи причинно-наслідкової узгодженості

1.3.4 Кінцева узгодженість

Цей вид узгодженості полягає в тому, що «за відсутності оновлень (записів) усі клієнти бачитимуть точно такий же стан системи через деякий час». Отже, перестануть надходити нові записи, система зрештою зійдеться до якогось узгодженого стану. Це дуже слабке обмеження, яке (ймовірно) простіше реалізувати та може бути дуже продуктивним.

Незважаючи на те, що багато нових продуктів баз даних було побудовано на основі моделі остаточної узгодженості, є низка причин, чому остаточна узгодженість недостатня в сучасну еру 5G.

Термін кінцева узгодженість використовується для опису моделі, яка зазвичай зустрічається в розподілених системах, яка гарантує, що якщо елемент не отримує нових оновлень, то врешті-решт усі звернення до цього елемента повертатимуть те саме значення, що відповідає останньому оновленню, таким чином забезпечуючи узгоджений результат. Ключовою перевагою узгодженої бази даних є те, що вона підтримує модель високої доступності NoSQL. Можлива узгодженість у мікросервісах

може підтримувати завжди доступний API, який має бути чуйним, навіть якщо в результатах запиту іноді може бути відсутнім останній запис [24].

Мабуть, найпоширенішим прикладом є використання асинхронної реплікації між двома або більше екземплярів баз даних. Асинхронність розповсюдження запису та зменшення накладних витрат на координацію дозволяють зменшити затримку та підвищити пропускну здатність для операцій читання та запису. Систему можна легше масштабувати та обробляти більшу кількість одночасних запитів [24].

Через його асинхронність, коли запис виконується, він зберігається лише локально, а пізніше інший процес піклується про копіювання та застосування тієї самої операції до репліки. Оскільки цей процес не залежить від початкової операції запису та не є миттєвим, є час, коли репліка не містить оновленої інформації.

1.4 Існуючі методи та способи підтримки узгодженості

1.4.1 Архітектурні рішення

Один із методів комунікації є централізований підхід. Суть полягає в тому, що всі операції запису проходять через головний вузол. Це дозволяє уникнути ситуацій, коли запис транзакцій відбувається лише для певної підмножини, а інші вузли зберігатимуть стару версію запису.

В контексті розподілених баз даних, всі операції йдуть через головний вузол, який робить записи до всіх реплік. Операції читання можуть бути направлені до реплік для зменшення навантаження на головний вузол.

Вибір головного вузла може відбуватися з використанням консенсусних протоколів, таких як Paxos або Raft.

Недоліком такого підходу є слабе місце у вигляді головного вузла. Його виведення з ладу погіршує доступність системи для виконання операцій записів на певний час. З іншого боку, операції читання можуть відбуватися через вторинні вузли, що забезпечує частину функціональності в робочому стані [25].

1.4.1.1 Paxos

Paxos — це сімейство алгоритмів, призначених для вирішення проблеми досягнення консенсусу між групою розподілених вузлів або процесів у мережі, навіть за наявності збоїв і мережевих розділів. Алгоритми консенсусу, такі як Paxos, мають вирішальне значення в розподілених системах, щоб переконатися, що вузли узгоджують один запис або послідовність записів, навіть якщо деякі вузли можуть вийти з ладу або мережа може бути ненадійною [26].

Основна ідея Paxos полягає в досягненні консенсусу між групою вузлів, залучаючи їх до серії раундів або фаз спілкування. Ці раунди призводять до узгодження вузлів щодо запису або послідовності записів, навіть якщо деякі вузли несправні або повідомлення втрачено.

Схему роботи зображено на рис. 1.7.

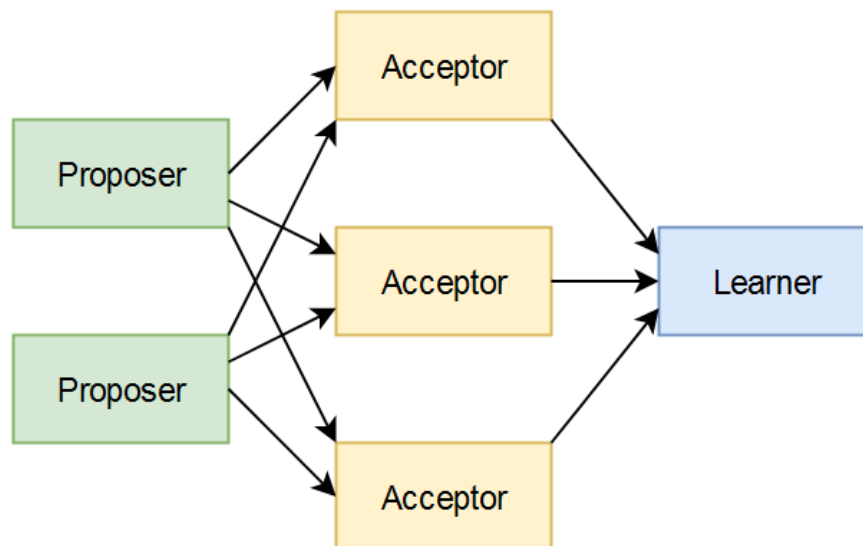


Рисунок 1.7 — Схеми роботи Paxos

1.4.1.2 Raft

Raft – це алгоритм консенсусу, розроблений для досягнення розподіленого консенсусу у відмовостійкий та зрозумілий спосіб. Як і Paxos, Raft використовується в розподілених системах, щоб гарантувати, що група вузлів може узгодити запис або

послідовність записів, навіть за наявності збоїв вузлів і мережових розділів. Raft відомий своєю простотою та ясністю, що полегшує його розуміння, впровадження та підтримку порівняно з деякими іншими консенсусними алгоритмами [27].

Реплікація журналу працює шляхом надсилання записів журналу з головного вузла до інших вузлів у системі (рис. 1.8).

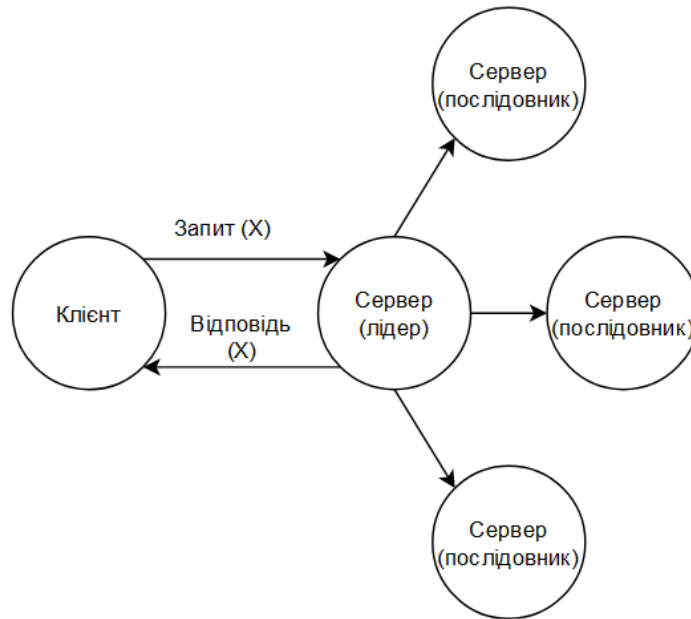


Рисунок 1.8 — Схема роботи Raft

Щоб зберегти повноваження лідера кластера, вузол-лідер надсилає спеціальні повідомлення, щоб підтвердити працездатність іншим вузлам-послідовникам. Вибори лідера відбуваються, коли у одного із вузлів закінчується час очікування повідомлення від вузла лідера. У цей момент вузол змінює свій стан на стан кандидата, голосує за себе та видає запит до інших для встановлення більшості та спроби стати лідером (рис. 1.9).

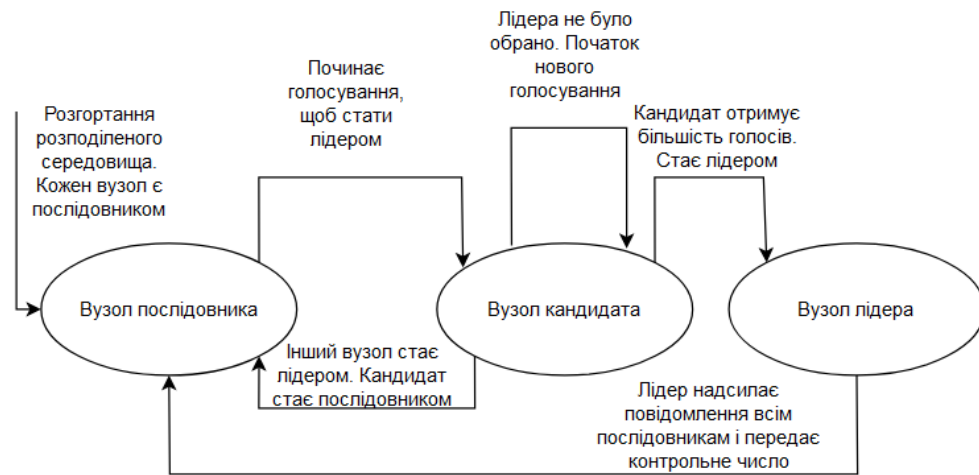


Рисунок 1.9 – Схема ініціалізації та процедура виборів

1.4.2 Протокол пліток

Протокол пліток — це децентралізована однорангова комунікаційна техніка для передачі повідомлень у розподіленій системі. Ключова концепція протоколу пліток полягає в тому, що кожен вузол періодично надсилає повідомлення набору інших випадкових вузлів. Вся система врешті-решт отримає певне повідомлення з високою ймовірністю. Інакше кажучи, протокол пліток — це техніка для побудови взаємодії всередині системи через обмежені локальні взаємодії [28].

Протокол пліток побудований на надійному, масштабованому та, зрештою, узгодженому алгоритмі. Він зазвичай використовується для підтримки списку учасників вузла, досягнення консенсусу та виявлення помилок у розподіленій системі (рис. 1.10).

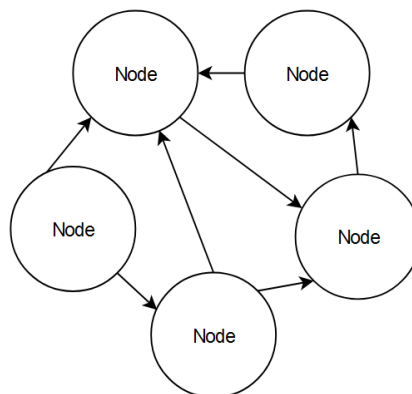


Рисунок 1.10 – Схема роботи протоколу пліток

Протокол пліток є надійним, оскільки збій вузла можна подолати шляхом повторної передачі повідомлення іншим вузлом. Трансляція FIFO, трансляція причинно-наслідкових зв'язків і трансляція загального порядку можуть бути реалізовані за допомогою протоколу пліток [28].

Цей протокол є оптимальним вибором в якості протоколу взаємодії у масштабних розподілених системах через наступні характеристики:

- обмежує кількість повідомлень, що передаються кожним вузлом;
- обмежує споживання пропускної здатності, щоб запобігти погіршенню продуктивності програми;
- витримує збої мережі та вузлів.

Протокол пліток можна використовувати для підтримки узгодженості вузлів лише тоді, коли виконувані операції є комутативними і можливість серіалізації не потрібна.

Однак, треба враховувати необхідний час для розповсюдження повідомлення по системі та характеристики мережі, оскільки саме це може впливати на ефективність. Протокол пліток можна розділити на такі типи:

- антиентропійний алгоритм було введено, щоб зменшити ентропію між репліками зі збереженням стану, наприклад, база даних. Відтворені дані порівнюються, а різниця між репліками виправляється. Вузол із найновішим повідомленням ділиться ним з іншими вузлами в кожному раунді комунікації;
- протокол розповсюдження повідомлень. Цикл поширення чуток відбувається відносно частіше, ніж антиентропійні цикли, і наповнює мережу найгіршим навантаженням;
- aggregation model.

Антиентропійна модель зазвичай передає весь набір даних, що призводить до неефективного використання пропускної здатності. Для зменшення навантаження на мережу та підвищення ефективності можуть бути використані різні методи, які

використовують контрольні суми, список останніх оновлень і дерево Меркла. Вони дозволяють визначити відмінності між вузлами та уникнути передачу всього набору даних і зменшити використання пропускну здатності мережі. Антиентропійний протокол пліток надсилатиме необмежену кількість повідомлень без завершення для контролю та вчасної ідентифікації неузгодженості

Поширення повідомлень може відбуватися на основі різних моделей:

- push model обумовлює надсилання повідомлення до випадкового набору інших вузлів;
- pull model обумовлює активне опитування випадковий набір вузлів стосовно оновлення. Цей підхід ефективний, коли є багато оновлень, тому що зростає ймовірність натрапити на оновлений вузол;
- push-pull model є оптимальною для швидкого та надійного розповсюдження повідомлень про оновлення. Вузол може надсилати нове повідомлення про оновлення, а також може запитувати вузли про оновлення. Підхід push ефективний на початковому етапі, коли є лише кілька вузлів з повідомленнями про оновлення. Підхід pull ефективний під час останньої фази, коли є багато вузлів із багатьма повідомленнями про оновлення [28].

1.4.3. Логічні годинники

Для логічних годинників не важлива конкретна дата та час, коли відбулася певна подія. Вони можуть встановлювати порядок та послідовність подій, які відбуваються. У розподіленій системі це допомагає підтримувати послідовний порядок подій між вузлами. Водночас використання логічного годинника спричиняє багато граничних ситуацій і ускладнює керування версіями даних [29].

До ключових аспектів можна віднести:

- причинність у розподілених системах означає залежність – зв'язок між двома подіями. Дві події причинно пов'язані, якщо існування однієї зумовлене

іншою (причина-наслідок). Подія В причинно залежить від іншої події А, якщо В відбувається лише тому, що А вже відбулася;

- частковий порядок, якщо два елементи не можна детерміновано порівняти [29].

Історія логічного годинника починається в 1978 році, коли Леслі Лампорт опублікував статтю під назвою «Час, годинники та порядок подій у розподілених системах». Частковий порядок можна описати наступним чином [30]:

- якщо в процесі Р виникає подія А перед іншою подією В, то $A \rightarrow B$;
- якщо процес Р відправляє подію іншому процесу Q, при цьому А представляє собою подію на вузлі Р, а В при цьому представляє собою підтвердження тієї самої події А процесом Q, то $A \rightarrow B$;
- якщо $A \rightarrow B$ та $B \rightarrow C$, тоді $A \rightarrow C$, що представляє собою транзитивне відношення [30].

Розглянемо N процесів $P_1, P_2, P_3, \dots, P_n$, що виконуються на різних машинах. Всі вони керують своїми локальними лічильниками — єдиного глобального лічильника не існує. Усі лічильники ініціалізуються зі значенням 0. Процеси можуть синхронізуватися один з одним, надсилаючи повідомлення m (рис. 1.11).

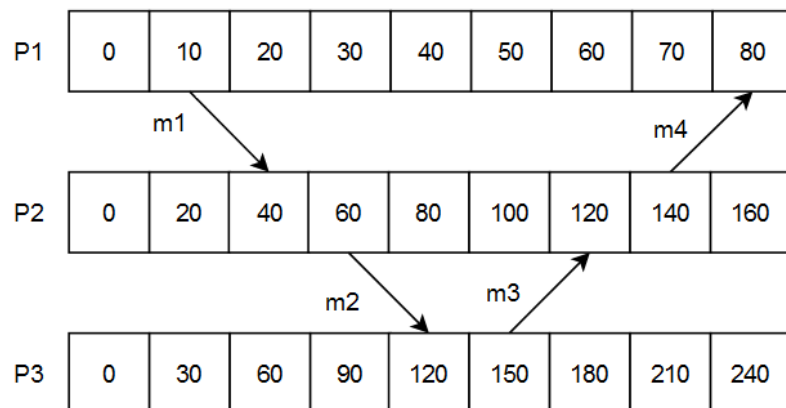


Рисунок 1.11 – Робота годинника

Процеси можуть збільшувати лічильники використовуючи різне значення кроку: P_1 збільшує свій лічильник на 10, P_2 збільшує на 20, а P_3 збільшує на 30. Зрештою, незалежно від того, як вони збільшуються, вони мають бути монотонними за своєю природою [30].

Кожен процес збільшує свій власний лічильник між двома послідовними подіями. Отже, перед надсиланням події через мережу процес збільшує свій лічильник, так само після отримання повідомлення процес збільшує свій лічильник і доставляє повідомлення відповідному процесу.

Векторні годинники є більш узагальненими годинниками Лампорта. На відміну від годинника Лампорта, кожен процес у векторному годиннику також знає про значення лічильників в інших процесах. Якщо існує N процесів, кожен процес підтримує вектор або масив з N лічильників. Коли процеси синхронізуються один з одним, лічильники оновлюються відповідним чином [31].

Для більш загального розуміння, замість терміну процес можна використовувати термін актор, який може означати процес, сервер, вузол. Якщо подія A сталася перед іншою подією B ($A \rightarrow B$), векторний годинник V_b буде більшим за V_a . Подібним чином, зворотне також вірно — на відміну від годинника Лампорта, порівнюючи V_a та V_b , ми можемо зробити висновок, що A відбулося раніше B .

Векторний тактовий сигнал V_i вважається меншим за інший векторний тактовий сигнал V_j , якщо всі елементи V_i менші або рівні V_j .

Векторний годинник можна представити у вигляді $[C(A_i), C(A_{i+1}), C(A_{i+2}), \dots, C(A_j)]$, де $C(A_i)$ — значення лічильника актора A_i [31].

Таким чином, якщо у нас є два векторні годинники $V_i = [1, 2, 1]$ та $V_j = [2, 3, 2]$, то $V_i < V_j$ оскільки кожне значення V_i менше за аналогічне в V_j .

На рис. 1.12 зображено приклад роботи векторного годинника.

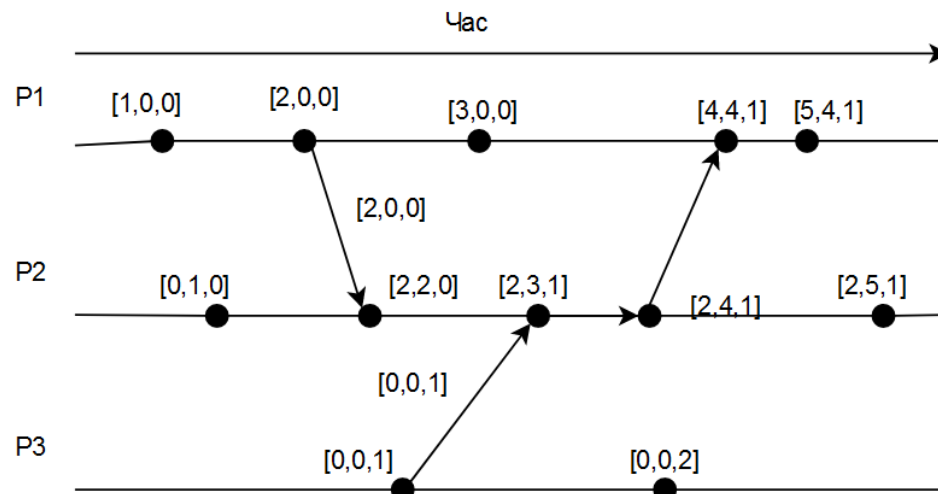


Рисунок 1.12 – Приклад векторного годинника з трьома процесами

Векторні годинники використовуються в Apache Cassandra та Riak з метою вирішення наступних задач:

- векторні годинники використовуються для відстеження оновлень і встановлення часткового порядку подій, які пов'язані із оновленнями об'єктів реплік, оскільки оновлення можуть відбуватися одночасно через розподілену архітектуру;
- векторний годинник певного об'єкта дозволяє легко встановити актуальність версій;
- векторні годинники дозволяють підтримувати кінцеву узгодженість бази даних і гарантують, що система в решті решт перейде в узгоджений стан [31].

1.4.4 Механізм читання виправлення

Механізм читання-виправлення – це техніка, яка використовується в розподілених базах даних для проактивного виявлення та виправлення невідповідностей або конфліктів у репліках даних під час операцій читання. Основна мета полягає в тому, щоб дані, зчитані клієнтом, були узгодженими в усіх репліках, і запобігти передачі застарілих або розбіжних даних клієнтам. Це допомагає підтримувати послідовність і цілісність даних у системах розподілених баз даних [32].

Алгоритм роботи механізму:

- операція читання, коли клієнт ініціює операцію читання, запит надсилається до одного з вузлів у кластері розподіленої бази даних;
- отримання даних, коли вибраний вузол отримує запитувані дані зі своєї локальної репліки;
- перевіряє, чи немає невідповідностей, розбіжних версій або конфліктних оновлень у репліках;
- виправлення при виявленні будь-яких розбіжностей або невідповідностей в даних, він ініціює процес виправлення. Відбувається визначення правильної або найновішої версії даних серед реплік. Після цього, вузол розповсюджує правильну версію та оновлює локальну версію;
- відповідь клієнту після процесу відновлення вузол надсилає відповідь клієнту, гарантуючи, що клієнт отримує найновіші та послідовні дані.

Механізм читання-виправлення часто використовується в розподілених базах даних, які надають пріоритет узгодженості даних і використовують реплікацію для забезпечення відмовостійкості та високої доступності. Проактивно виявляючи та виправляючи невідповідності під час операцій читання, система мінімізує ймовірність надання клієнтам застарілих або неправильних даних [32].

1.4.5 Двофазова фіксація

Двофазова фіксація (2PC) – це стандартизований протокол, який забезпечує атомарність, послідовність, ізоляцію та довговічність (ACID) транзакції; це атомарний протокол для розподілених систем [33].

У розподіленій системі транзакції передбачають зміну даних у кількох базах даних або менеджерах ресурсів, що ускладнює обробку, оскільки база даних має координувати внесення або відкочування змін через виконання транзакції як самостійної одиниці; або вся транзакція фіксується, або вся транзакція відкочується.

Менеджер транзакцій використовує 2PC для забезпечення цілісності даних, а також цілісності глобальної бази даних – колекції баз даних, які беруть участь у транзакції – а також для моніторингу фіксацій або відкочування розподілених транзакцій.

Для виконання розподіленої транзакції, потрібен спеціальний компонент, відомий як координатор. Координатор відповідає за організацію дій та синхронізацію між розподіленими вузлами.

Фаза 1 (фаза підготовки) – протокол гарантує, що всі менеджери ресурсів зберегли операції транзакції в стабільному сховищі. Кожен вузол, якому потрібно зафіксувати, записує свої поточні записи у журнал. Це необхідно для відкочування змін і повернення до попереднього стану. Якщо серверу це не вдається, він відповідає повідомленням про помилку; якщо це успішно, тоді він надсилає повідомлення-підтвердження [33].

На цьому першому етапі вузол-ініціатор просить всі інші вузли-учасники гарантувати або фіксування, або відкочування транзакції.

Існує три типи відповідей, які вузол-відповідач може надіслати назад:

- відповідь готовності надається, коли дані на вузлі були успішно підготовлені для фіксації або відкочування. Ця відповідь також гарантує, що блокування, утримуване для транзакції, може пережити збій;
- відповідь лише для читання означає, що дані на вузлі поставлені на читання, тому ніякої підготовки не потрібно;
- відповідь на переривання – вказує на те, що вузол не може успішно підготуватися до фіксації.

Щоб фаза підготовки була завершена та було надіслано одне з трьох повідомлень, кожен вузол, за винятком вузла, який ініціює фіксацію, повинен виконати кілька кроків. По-перше, вузол повинен запитати чи готові інші вузли до фіксації. Потім вузол перевіряє, чи змінює транзакція дані на ньому самому або на інших вузлах. Якщо дані

не змінюються, то вузол пропускає решту кроків і відповідає відповіддю лише для читання [33].

Якщо дані змінюються, то вузол призначає ресурси, необхідні для фіксування транзакції. Вузол вносить операції транзакції в локальний журнал. Після цього відбувається блокування таблиці (документу), для запобігання читання. Після цього вузол готовий робити запис, але потрібна команда координатора.

Далі вузол гарантує, що блокування, утримувані для транзакції, можуть пережити збій. Якщо всі кроки йдуть за планом, то вузол видає позитивну відповідь. Однак, якщо спроби вузла або одного з його наступних вузлів виявляються невдалими в підготовці до фіксації, тоді він видає відповідь про скасування.

Потім підготовлені вузли чекають запит на фіксацію або відкочування від глобального координатора. Підготовлені вузли вважаються сумнівними, доки всі зміни не буде зафіксовано або відбудеться відкочування.

Фаза 2 (фаза фіксації) – якщо перша фаза успішна і всі учасники надсилають відповідь ОК, тоді друга фаза повідомляє всім менеджерам ресурсів про фіксацію. Після фіксації кожен вузол реєструє свою фіксацію в записі та надсилає координатору повідомлення про те, що її фіксація була успішною. Якщо перший етап завершується невдачею, то другий етап повідомляє менеджерам ресурсів про переривання, усі сервери відкочуються, і кожен вузол надсилає відповідь про те, що відкочування успішно виконано [33].

Фазу фіксації можна розбити на такі кроки:

- глобальний координатор сповіщує ключовий вузол зробити фіксацію;
- ключовий вузол робить фіксацію та надсилає відповідь глобальному координатору, повідомляючи, що операція успішна;
- глобальні та локальні координатори дають вказівки всім іншим вузлам зафіксувати транзакцію;

- база даних кожного вузла знімає свої блокування та фіксує свою локальну порцію розподіленої транзакції;
- база даних кожного вузла реєструє додатковий запис у своєму локальному журналі про успішну фіксацію;
- усі вузли-учасники сповіщають глобального координатора про стан свого успішного фіксування транзакції.

Після завершення фази фіксації всі вузли в розподіленій системі містять узгоджені дані [33].

Висновки до розділу 1

Інформаційні системи представляють собою розподілені системи, які вимагають нових підходів та засобів для забезпечення ключових властивостей, до яких відносяться узгодженість, доступність та здатність до розділу мережею.

Кожна властивість так чи інакше впливає на взаємодію кінцевих користувачів із системою. При низькому рівні узгодженості інформація, яка зберігається в розподіленій базі даних, може відрізнятися одна від одної на різних екземплярах. Це вимагатиме наявності спеціально розроблених процедур для вирішення таких конфліктів.

В якості прикладу можна навести інформаційну систему, яка спеціалізується на бронюванні готельних номерів. Якщо система не має спеціальних механізмів для підтримки узгодженості, то можуть бути такі ситуації, коли декілька людей забронювали і проплатили за один і той самий номер. Тоді бізнес повинен бути готовим до такого і у разі потреби мати резервні номери, які зможе запропонувати клієнтам.

Незалежно від типу бази даних, спільним для них всіх є те, яку модель узгодженості вони реалізують для забезпечення ефективного виконання функцій інформаційної системи.

Модель узгодженості не може відповісти на питання “Які способи та методи краще використовувати?”, оскільки кожна інформаційна система унікальна і вимагає ретельного дослідження не тільки існуючих механізмів забезпечення узгодженості, а і глибокого розуміння функцій та задач, які покладені на цю систему.

Моделі узгодженості відповідають на питання “Як система себе може поводити?” та “Як система себе не може поводити?”. Кожна модель надає певні гарантії щодо актуальності інформації, яку буде отримувати кінцевий користувач та потенційні незручності, які можуть на нього очікувати.

В кінцевому результаті, узгодженість даних в будь-якій інформаційній системі, яка використовує розподілену базу даних, буде залежати від ефективності використання тих чи інших методів, що дозволять уникнути людських або матеріальних втрат через паралельне виконання бізнес-процесів.

2 МЕТОДИ ТА ТЕХНОЛОГІЇ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ УЗГОДЖЕНОСТІ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ

Розглянуті методи, які використовуються в розподілених нереляційних базах даних, дозволяють реалізувати ту чи іншу модель узгодженості. В контексті кінцевої узгодженості вирішальну роль має швидкість процесу узгодження.

Одним із завдань може бути пошук неузгодженості у системі. Для цього можуть використовувати дерево Меркла, яке базується на хешуванні. Цей спосіб дозволяє доволі швидко локалізувати неузгодженість, але піддатливий до хеш-колізій, оскільки при зростанні кількості сегментів зростає вірогідність виникнення колізії. Тобто, вплив різних сегментів може призвести до збільшення часу, при якому система знаходиться в стані неузгодженості, що є неприпустим для будь-якої інформаційної системи і може бути критичним в окремих випадках.

Також, можна відзначити такий момент, який відноситься до централізованого підходу. Транзакції виконуються одна за одною, що може бути неефективним, оскільки операції запису на жорсткий диск доволі витратні в порівнянні з використанням оперативної пам'яті. Окрім цього, немає сенсу виконувати транзакції по черзі, які відносяться до одного і того самого об'єкту.

На швидкість цього процесу можуть впливати різні фактори, але вирішальну роль все-таки відіграє саме використання правильних алгоритмів, які дозволять підвищити точність пошуку узгодженості та мінімізувати кількість операцій запису на жорсткі носії.

2.1 Загальна структура розподіленої інформаційної системи

Інформаційні системи можуть відрізнятися структурною схемою та покладеними задачами, але в них завжди буде місце для розподілених даних, які повинні бути доступними для користувачів (рис. 2.1).

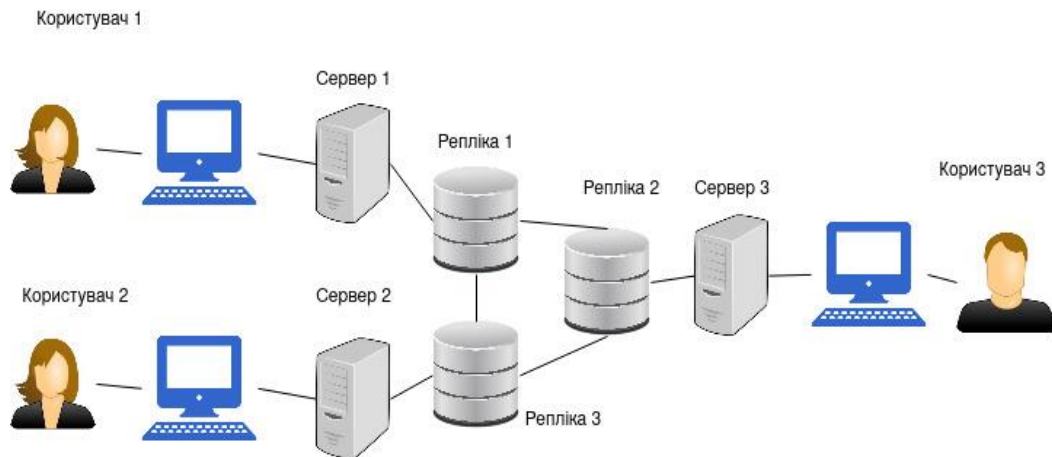


Рисунок 2.1 – Приклад інформаційної системи

Різні відділи компаній та організацій можуть мати власні інтерфейси для взаємодії із інформаційною системою, але розподілена база даних є тим спільним знаменником, який використовується для всіх користувачів не залежно від цілей. Якщо брати в якості прикладу фінансову установу, то окрім звичайних користувачів у вигляді фізичних або юридичних осіб можуть бути спеціальні відділи компанії для проведення фінансового моніторингу.

При побудові інформаційної системи повинні бути враховані багато нюансів та факторів, які можуть впливати на проектування та впровадження розподілених систем. Розробники та архітектори повинні пам'ятати, що розподілені обчислення принципово відрізняються від одновузлових.

Комп'ютерна мережа, що з'єднує вузли розподіленої системи, не завжди надійна та доступна. Насправді в мережах можуть виникати збої, втрата пакетів та затримки при передачі. Це має бути враховано.

Слід пам'ятати, що зв'язок між компонентами в розподіленій системі відбувається не миттєво і нульовою затримкою. На практиці затримка мережі може значно відрізнятись, а віддалені виклики процедур або передача даних можуть зайняти час.

Також, пропускна здатність обмежена при передачі даних між розподіленими компонентами. Це може призводити до затримок між серверними додатками та базами даних, що може погіршити властивості.

Слід не забувати про те, що мережа представляє собою розподілене середовище передачі інформації і вона не є безпечною за замовчуванням. Під час транспортування інформації можуть відбуватися перехоплення та втручання. Для забезпечення безпеки використовуються різні методи та способи, до яких можуть входити механізми шифрування та автентифікації, які також додають додаткові витрати.

Структура та топологія розподіленої системи є динамічними та змінюються протягом часу: додаються нові вузли, видаляються або просто виходять з ладу. Адаптація системи важлива при будь-яких структурних змінах, оскільки впливає на її доступність.

Також, на практиці розподілені системи можуть включати кілька адміністраторів, організацій або навіть автономних вузлів. Координація та управління можуть бути ускладненими і потребувати додаткових механізмів для контролю змін.

Пересилання даних між компонентами в розподіленій системі має багато витрат. Для транспортування задіюються багато ресурсів таких, як CPU, пам'ять і пропускна здатність мережі.

Окрім цього, всі компоненти розподіленої системи необов'язково повинні працювати на однакових платформах, з однаковими операційними системами та конфігураціями. Хоча існує велика кількість стандартів, які уніфікують рішення, але все-одно це потрібно враховувати.

2.2 Розподілена база даних

Розподілена база даних представляє собою спеціальне сховище для зберігання та обробки даних. Для кінцевих користувачів вона має єдиний інтерфейс і її використання прозоре, що дозволяє приховати реалізацією за абстракцією. Це може

бути корисним у випадках є необхідність розподілу навантаження через зростання інформаційної системи. Приклад розподіленої бази даних можна побачити на рис. 2.2.

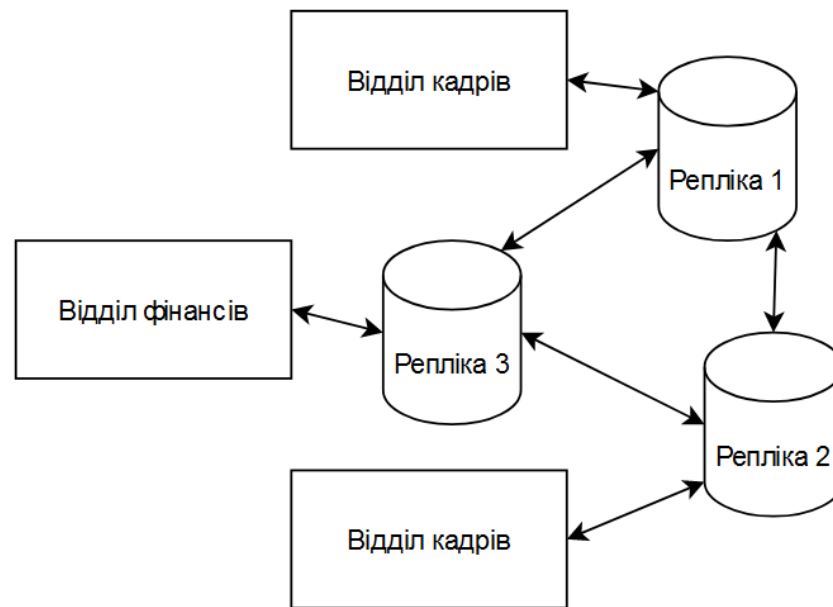


Рисунок 2.2 – Розподілена база даних

Розподілені бази даних мають багато переваг, таких як масштабованість, доступність, продуктивність і відмовостійкість, але також створюють деякі складні проблеми, такі як узгодженість даних, контроль паралельності, реплікація та безпека. Однією з важливих задач є методи визначення кількості вузлів, які повинні надавати зручні варіанти розрахунку [34].

Це дає можливість додавати або видаляти вузли, бази даних або елементи даних, не впливаючи на загальну функціональність системи, полегшуючи роботу зі збільшенням або зменшенням обсягів даних і вимог користувачів. Крім того, розподілені системи можуть гарантувати, що інформаційна система завжди буде доступна та працездатна, навіть якщо деякі вузли чи бази даних виходять з ладу або відключаються. Це зменшує ризик втрати даних і простоїв. Також, покращиться швидкість і ефективність обробки даних і виконання запитів через розподіл робочого навантаження між кількома базами даних і розташувавши дані фізично ближче до

користувачів. Це час затримки на мережу та споживання пропускну здатності. Реплікація дозволяє відновитися після збоїв, зберігаючи цілісність і послідовність даних.

Під час проектування розподіленої бази аналізуються вимоги та цілі інформаційної системи, а також визначаються джерела даних, користувачів і місцеположення. Потім розробляється логічна схема бази даних з визначення моделей даних, структур та зв'язків. Одним із важливих кроків є схема реплікації бази даних для розуміння того, як буде розповсюджуватися дані між різними вузлами бази даних. Це дозволяє підібрати необхідні методи та способи узгодження даних та контролю паралельності.

2.3 Загальнопоширені алгоритми узгодження даних в інформаційних системах

2.3.1 Механізм активної антиентропії

Цей метод базується на протоколі пліток, суть якого полягає в комунікації між різними вузлами системи напряду [35].

Класичним способом для цього є використання дерева Меркла [36]. Дерево Меркла – це структура даних, яка дуже корисна в розподілених сценаріях, оскільки дозволяє безпечно перевіряти структури даних без необхідності її повної передачі стороні, яка буде перевіряти. Ця структура представляє собою дерево, в якому кожен листовий вузол містить хеш-значення блоку даних, а кожен нелістовий вузол - його дочірніх вузлів. Частинами дерева Меркла є вузли та корінь (самий верхній вузол) (рис. 2.3) [37].

Для перевірки узгодженості даних на двох різних вузлах достатньо побудувати це дерево та порівняти отримані хеш-значення. Перевірка починається з кореневого значення. Якщо ці хеш-значення відрізняються, то відбувається перевірка хеш-значень

наступного рівня. Це дозволяє відносно легко та швидко локалізувати неузгоджені дані [37].

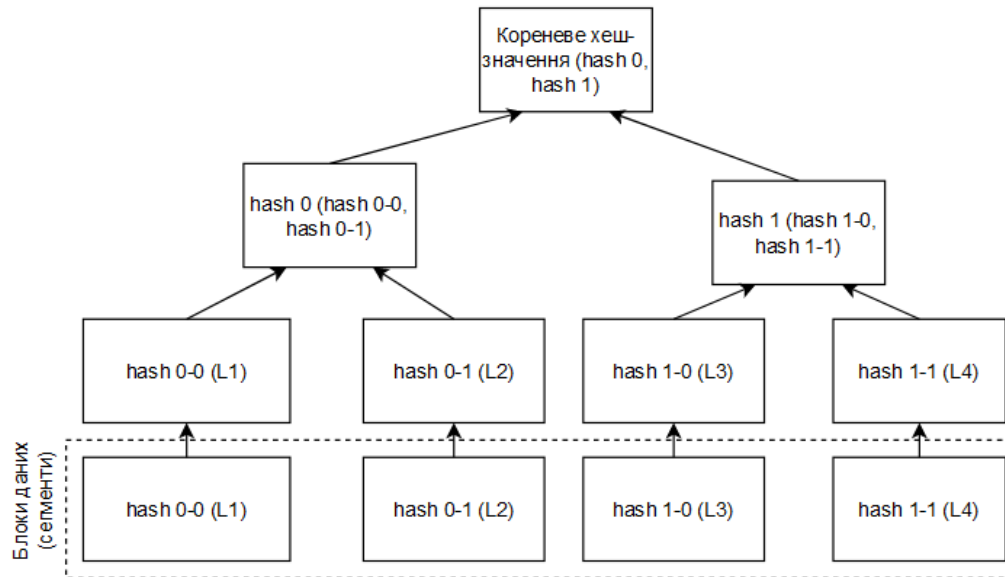


Рисунок 2.3 – Графічне зображення дерева Меркла

Переваги використання дерева Меркла для ідентифікації неузгодженості:

- безпечно, бо вузли обмінюються між собою знімками, а не фактичними даними;
- малий розмір даних, які необхідно перевірити, оскільки після пошуку можна точно ідентифікувати неузгоджені дані, локалізувати та виправити. Немає необхідності в перевірці даних, які відносяться до інших гілок [37].

Розглянемо можливу реалізацію пошуку неузгодженостей в розподіленій базі даних. Для початку створимо сегменти для зберігання набору ключів. Кожен ключ асоціюється з якимось значенням. Ключі мають роль ідентифікатора для перевірки даних на різних вузлах (рис. 2.4). Для кожного сегменту розраховується хеш-значення, які використовуються для побудови дерева.

Вузол 1	Вузол 2
1 -> "kpi"	1 -> "КПІ"
2 -> "fict"	2 -> "ФІОТ"
3 -> "abc"	3 -> "abc"
4 -> "bca"	4 -> "bca"
5 -> "kmn"	5 -> "kmn"
	6 -> "jkv"

Рисунок 2.4 – Два неузгоджених екземпляри баз даних

На рис. 2.4 видно, що вузли неузгоджені і дані відносяться до першого та третього сегментів. Кожен запис розміщується в певному сегменті. Наприклад, четвертий сегмент повністю пустий. Але для того, щоб системі це побачити необхідно виконати ідентифікацію та локалізацію неузгодженості.

Перший вузол будує дерево Меркла (рис. 2.5), а другий вузол будує своє (рис. 2.6).

Якщо є дерево Меркла для двох вузлів, то можна дізнатися, чи однакові дані на цих двох вузлах, просто порівнявши кореневий вузол. Якщо кореневий хеш однаковий, обидва вузли скоріш за все мають однакові дані. Якщо ні, то перейти на рівень нижче і повторити процес порівняння хешів дочірніх вузлів, щоб знайти, які частини дерева відрізняються. Це допоможе локалізувати сегменти, які не синхронізовані. Після цього можна запустити механізм, який узгодить дані. Він може використовувати часові мітки або логічні годинники для визначення тієї версії документа, яка є актуальною.

Таким чином, використовуючи дерево Меркла, кількість інформації, якою обмінюється для синхронізації, буде пропорційна відмінностям між двома репліками,

а не кількості даних, які вони містять. Це забезпечує ефективність під час синхронізації двох вузлів. Антиентропійний менеджер може запускатись як в певні проміжки часу, так і працювати безперервно.

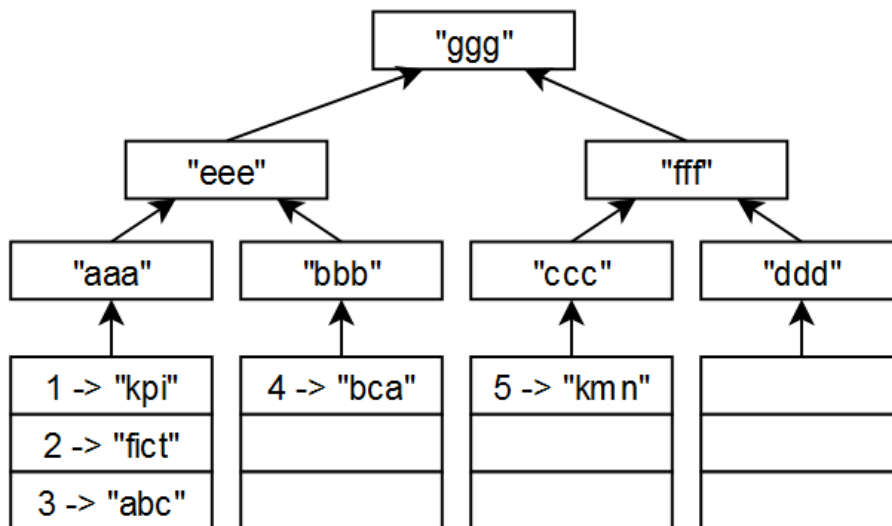


Рисунок 2.5 – Дерево Меркла побудоване на першому вузлі

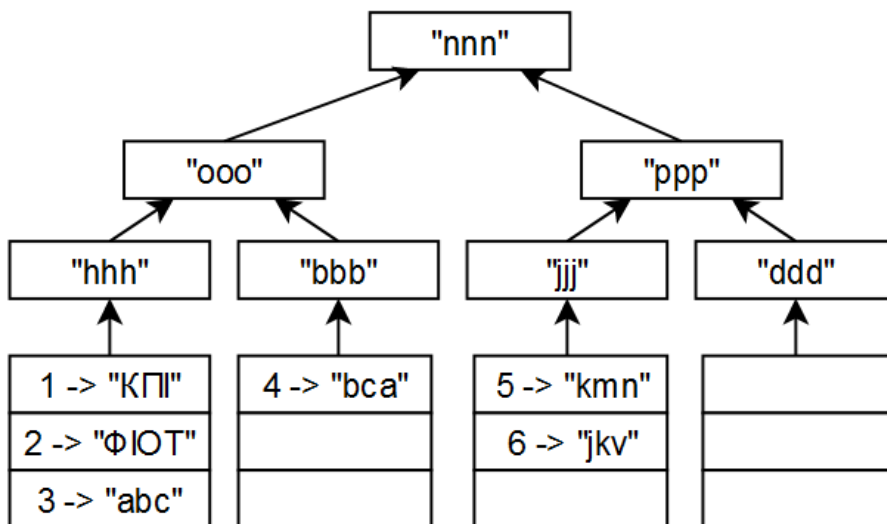
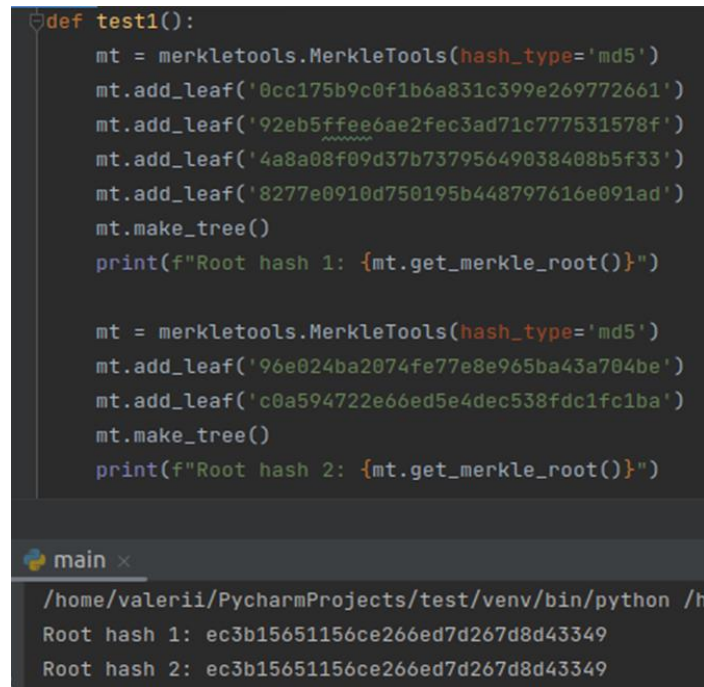


Рисунок 2.6 – Дерево Меркла побудоване на другому вузлі

Для хешування можуть використовуватися SHA-2 або CRC32, залежно від цілей. Слід зауважити, що використовуватись може будь-який алгоритм. Обмеження залежать від вимог системи.

Слід пам'ятати, що будь-який алгоритм хешування в якості недоліку має можливість колізій. Вони можуть виникати, коли при різних вхідних масивах даних хеш-функція повертає одне і те саме значення. Дерево Меркла активно використовує хешування і вірогідність колізій збільшується при збільшенні кількості сегментів. Приклад можна побачити на рис. 2.7.



```
def test1():
    mt = merkletools.MerkleTools(hash_type='md5')
    mt.add_leaf('0cc175b9c0f1b6a831c399e269772661')
    mt.add_leaf('92eb5ffee6ae2fec3ad71c777531578f')
    mt.add_leaf('4a8a08f09d37b73795649038408b5f33')
    mt.add_leaf('8277e0910d750195b448797616e091ad')
    mt.make_tree()
    print(f"Root hash 1: {mt.get_merkle_root()}")

    mt = merkletools.MerkleTools(hash_type='md5')
    mt.add_leaf('96e024ba2074fe77e8e965ba43a704be')
    mt.add_leaf('c0a594722e66ed5e4dec538fdc1fc1ba')
    mt.make_tree()
    print(f"Root hash 2: {mt.get_merkle_root()}")

main x
/home/valerii/PycharmProjects/test/venv/bin/python /h
Root hash 1: ec3b15651156ce266ed7d267d8d43349
Root hash 2: ec3b15651156ce266ed7d267d8d43349
```

Рисунок 2.7 – Приклад колізії кореневого хеш-значення при використанні MD5 алгоритму хешування

Спрогнозувати ту чи іншу колізію неможливо, оскільки існує багато вихідних умов, які впливають на результат. Криптографічні алгоритми взагалі унеможливають будь-яке прогнозування, оскільки це погіршувало б безпеку. Хешування використовується в багатьох напрямках і це було б не припустимо мати таку вразливість [38]. Приклади використання хешування:

- зберігання захешованих паролей у базах даних для забезпечити безпеки особистих даних користувачів при витоку даних. Багато користувачів мають

один і той самий пароль для багатьох сервісів, що може бути потенційною загрозою;

- використання для електронно-цифровий підпису для шифрування не самого повідомлення, а його хеш-значення. Це зменшує час обчислення, а також покращує криптостійкість.

Для розподіленої системи, яка використовує механізм активної антїентропії, це може негативно впливати, оскільки пошук неузгодженості може потребувати більше часу, ніж очікувалось.

Такий механізм мають такі розподілені бази даних, як Amazon DynamoDB, Cassandra, Riak.

Окрім дерева Меркла, існують ще фільтр Блума та його модифікації, але ця структура даних має певну специфіку і відрізняється від нього, хоча вирішує схожі задачі.

Фільтр Блума представляє собою імовірнісну структуру даних, яка використовується для перевірки того, чи належить певний елемент до множини значень. Наприклад, перевірка доступності імені користувача є проблемою членства в наборі, де набір є списком усіх зареєстрованих імен користувачів. Ціна, яку ми платимо за ефективність, полягає в тому, що вона носить імовірнісний характер, що означає, що можуть бути деякі хибнопозитивні результати. Помилкове спрацьовування означає, що це може означати, що дане ім'я користувача вже зайнято, але насправді це не так [39].

З іншого боку, ця структура даних гарантує, що хибно негативні результати неможливі. Іншими словами, якщо перевірка входження певного елементу дала негативний результат, то такого елементу дійсно немає.

На рис. 2.8 зображено пустий фільтр Блума.

0	0	0	0	0	0	0	0	0	0
1	2	3	4	5	6	7	8	9	10

Рисунок 2.8 – Пустий бітовий фільтр Блума

Фільтр Блума представляє собою бітовий масив, тобто кожен елемент якого може мати значення 1 або 0. Його заповнення відбувається з використанням хешування та операції ділення за модулем на розмір масиву.

Нехай у нас є 3 функції хешування та пустий фільтр Блума як на рис. 2.8. Отримуємо індекси масиву наступним чином: $h_1(\text{"Miller"}) \% 10 = 2$, $h_2(\text{"Miller"}) \% 10 = 5$, $h_3(\text{"Miller"}) \% 10 = 6$.

Після цього масив буде виглядати як зображено на рис. 2.9.

0	1	0	0	1	1	0	0	0	0
1	2	3	4	5	6	7	8	9	10

Рисунок 2.9 – Приклад заповнення фільтра Блума

Таким чином, якщо буде потрібна перевірка, чи використовується цей елемент чи ні, то процедура буде схожа: отримати індекси і перевірити їх значення. Якщо всі значення дорівнюють одиниці, то цей елемент вже є, якщо ні — то немає [39].

Додамо ще один елемент таким способом. Визначимо індекси так само, як робили до цього: $h_1(\text{"Andrii"}) \% 10 = 7$, $h_2(\text{"Andrii"}) \% 10 = 2$, $h_3(\text{"Andrii"}) \% 10 = 6$.

Тепер фільтр Блума виглядатиме так, як на рис. 2.10.

0	1	0	0	1	1	1	0	0	0
1	2	3	4	5	6	7	8	9	10

Рисунок 2.10 – Фільтр Блума після вставки ще одного елемента

Роглянемо хибно позитивний випадок. Візьмемо інше значення та розрахуємо індекси: $h_1(\text{“Hutnyk”}) \% 10 = 7$, $h_2(\text{“Hutnyk ”}) \% 10 = 5$, $h_3(\text{“Hutnyk ”}) \% 10 = 2$.

На рис. 2.10 видно, що значення дорівнюють одиниці, тож цей елемент нібито є у множині, хоча насправді його вставки не було.

Фільтр не дозволяє операцій видалення, оскільки встановити чи встановлений в одиницю біт має відношення до одного запису, а не декільком, неможливо.

Ймовірність хибного позитивного результату може бути розрахована за формулою:

$$P = (1 - [\frac{1}{m}]^{k \cdot n})^k, \quad (2.1)$$

де m – розмір бітового масиву;

k – кількість хеш-функцій;

n – кількість очікуваних елементів, які будуть вставлені у фільтр [39].

Розмір бітового масиву можна розрахувати за формулою, якщо відома очікувана кількість елементів і бажана ймовірність помилкового позитивного результату дорівнює, тоді розмір бітового масиву m можна обчислити як:

$$m = - \frac{n \cdot \ln P}{(\ln 2)^2}, \quad (2.2)$$

де n – кількість очікуваних елементів, які будуть вставлені у фільтр;

P – ймовірність помилкового позитивного результату [39].

Оптимальну кількість хеш-функцій для використання можна розрахувати за наступною формулою:

$$k = \frac{m}{n} \cdot \ln 2, \quad (2.3)$$

де m – розмір бітового масиву;

n – кількість очікуваних елементів, які будуть вставлені у фільтр [39].

Дана структура даних доволі поширена і може застосовуватись в розподілених базах даних. Одним із способів використання є алгоритм BloomJoin, мета якого полягає в тому, щоб зменшити не ефективне використання часу при передачі даних.

BloomJoin виявився більш ефективним для зниження витрат і підвищення продуктивності, ніж будь-яка інша техніка.

Є різні варіації даної структури даних: Distributed Bloom Filter, Spectral Bloom Filter, Space-Code Bloom Filter. Кожен з цих варіантів має свої відмінності та, відповідно, ситуації використання.

Distributed Bloom Filter представляє собою імовірнісну структуру даних, призначеною для розподілених систем, які потребують швидкої синхронізації в часі. Це досягається через надсилання фільтру кожним вузлом з його ідентифікатором до іншого вузла системи, що призводить до зміни ймовірності реплікації даних. Слід зауважити, що ймовірність хибнопозитивного результату зменшується при збільшенні загальної кількості вузлів [40].

Space-Code Bloom Filter досягає гарного компромісу між точністю підрахунку та кількістю бітів, що використовуються для підрахунку. Він представляє собою мультинабір, розширюючи можливості класичного фільтру Блума для представлення набору. Якщо задано елемент X , він не тільки дозволяє перевірити, чи є X у мультимножині, але також підраховує кількість входжень. SCBF має кілька важливих застосувань у вимірюванні мережі. Наприклад, вимірювання трафіку за потоком для проектування трафіку та виявлення аномалій [41].

Spectral Bloom Filter замінює вектор бітів на вектор лічильників. Лічильники представляють собою позитивні натуральні числа, які на самому початку мають нулі. При розрахунку позиції у векторі даних, збільшується відповідний лічильник на одиницю. Видалення елемента проходить схожим чином, але віднімаючи одиницю від відповідних лічильників [42].

2.3.2 Механізми з використанням консенсусних протоколів та централізованими операціями записів

Вузли такої розподіленої бази даних можуть мати такі ролі:

- головний вузол для виконання операцій читання-запису. При операціях запису відбувається реплікація на другорядні вузли;
- другорядні вузли, головною метою яких є зберігання даних, які були отримані від головного вузла. Один із таких вузлів може бути підвищено до головного вузла;
- допоміжний вузол, який приймає участь у досягненні консенсусу під час виборів головного вузла. Може не зберігати дані взагалі.

Загальна процедура представляє наступне:

- клієнтський додаток відправляє операцію запису;
- при виконанні операції запису на основному вузлі, вона спочатку записується в журнал операцій, а потім надсилається до вторинних вузлів;
- вторинні вузли неперервно отримують дані з основного журналу операцій головного вузла і застосовують ці зміни до власних журналів.

Всі вузли розподіленої бази даних відправляють повідомлення іншим вузлам для того, щоб показати що вузол в порядку. Якщо на головному вузлі стався збій, то інші вузли легко можуть це дізнатися, оскільки перестануть отримувати ці повідомлення у певні проміжки часу. Коли це сталося, то будь-який другорядний вузол може розпочати процес виборів.

За таким принципом працює MongoDB ReplicaSet. База даних може складатися з 50 вузлів і з них 7 членів можуть приймати участь у голосуванні [43].

2.3.3 Полегшені транзакції

Полегшені транзакції (LWT) — це функція Apache Cassandra, розподіленої бази даних NoSQL, яка забезпечує підтримку узгоджених і атомарних операцій. LWT дозволяє користувачам виконувати однорядкові транзакції, які зберігають властивості ACID (Atomicity, Consistency, Isolation, Durability), навіть у середовищі розподіленої та високодоступної бази даних.

Полегшені транзакції з лінеаризованою узгодженістю забезпечують рівень ізоляції транзакцій, подібний до серіалізованого рівня, що пропонується RDBMS. Вони також відомі як транзакції порівняння та встановлення. Використання таких транзакцій замість довготривалих транзакцій із можливою/регульованою узгодженістю для ситуацій, коли потрібні вузли в розподіленій системі для узгодження змін даних. Наприклад, два користувачі, які намагаються створити унікальний обліковий запис користувача в одному кластері, можуть перезаписати роботу один одного. Використовуючи легку транзакцію, вузли можуть погодитися створити лише один обліковий запис [44].

Cassandra реалізує легкі транзакції, розширюючи консенсусний протокол Paxos, який базується на алгоритмі на основі кворуму. Використовуючи цей протокол, розподілена система може узгодити запропоновані доповнення/модифікації даних без потреби в головній базі даних або двофазній фіксації.

До основних характеристик цього механізму можна віднести наступне:

- гарантування сильної узгодженості, що операції читання та запису в рамках транзакції слідує сильній узгодженості, яка забезпечує, що операції читання в транзакціях завжди повертатимуть останню версію або взагалі не повертатимуть значення, а операції запису є атомарними та ізольованими від інших транзакцій;
- однорядкові транзакції операцій, які включають один рядок або запис у таблиці Cassandra. Це дозволяє виконувати умовні оновлення або вставки на основі конкретних умов, що робить його придатним для випадків використання, коли узгодженість даних є критичною;
- семантика порівняння та встановлення при використанні для сценаріїв порівняння та встановлення. Іншими словами, можна вказати умову, яка має бути виконана, щоб транзакція була успішною. Якщо умова виконується, виконується операція оновлення або вставки; інакше це не вдається;

- механізм реалізовано за допомогою умови IF у операторах CQL (Cassandra Query Language). Наприклад, під час виконання оновлення можна включити умову IF для має бути істинною для запису, інакше оновлення відкочується;
- контроль паралелізму через використання протоколу Raft для координації та керування одночасними транзакціями. Raft гарантує, що тільки одна транзакція буде успішною, якщо кілька транзакцій намагаються оновити ті самі дані одночасно;
- наявність компромісів щодо продуктивності. Використання Raft для консенсусу може ввести додаткову затримку порівняно зі звичайними операціями без LWT. Таким чином, краще використовувати ретельно продумуючи для яких випадків слід використовувати [44].

Таким чином, LWT є цінною функцією в Cassandra для ситуацій, коли узгодженість і атомарність даних є вирішальними, навіть у середовищі розподіленої та високодоступної бази даних. Це дозволяє розробникам виконувати однорядкові транзакції і бути впевненими, що дані будуть оновлюватися атомарно та послідовно відповідно до принципів транзакцій ACID, який більше популярний в реляційних базах даних.

2.3.4 Потоки DynamoDB

Потік DynamoDB — це впорядкована за часом послідовність подій, що записує всі зміни для таблиць DynamoDB майже в реальному часі. Це дозволяє фіксувати зміни елементів у таблиці DynamoDB, а потім виконувати певні дії на основі цих змін. До одних з ключових особливостей можна віднести [45]:

- фіксує зміни в елементах таблиці DynamoDB практично в реальному часі, які можуть включати створення нових записів, оновлення існуючих і видалення;

- впорядкування за часом дозволяє виконувати обробку саме у тій послідовності, в якій відбулися події;
- широка масштабованість та надійність вимагають гнучкого підходу для реагування на зміни в системі. Використання подіє-орієнтованого підходу дозволяє зняти навантаження, оскільки немає потреби в неперервному опитуванні або моніторингу таблиць.

Слід зазначити, що надійні та узгоджені операції мають відбуватися в одній репліці регіону, оскільки глобальні таблиці підтримують послідовне читання лише для міжрегіональних операцій. При читанні з репліки, відмінної від тієї, яка відповідальна за певний регіон, можна отримати застарілі дані. У разі конфлікту для різних реплік використовуватиметься політика перемоги останнього автора [45].

2.4 Метод підвищення ефективності узгодженості даних за допомогою транзакційного годинника з пріоритизацією

Однією із проблем, яка може виникнути в інформаційній системі, може бути надсилання великої кількості поодиноких транзакцій, які відносяться до одного і того самого документу. Це можуть бути оновлення якихось статистичних даних, які зберігаються в одному документі. В якості прикладу можна навести системи позиціонування в режимі реального часу, які працюють наживо та займаються розпізнаванням та відстеженням об'єктів, та визначенням координат. Метою таких систем є накопичення, обробка та зберігання даних. Для цього розгортається спеціальна інфраструктура з усім необхідним обладнанням. Такі інформаційні системи можуть мінімізувати час для пошуку постраждалих внаслідок надзвичайних ситуацій або контролювати технологічні процеси на виробництвах задля зменшення ризику людських або матеріальних втрат.

2.4.1 Логіка роботи та математичний опис транзакційного годинника з пріоритезацією

Процес узгодження даних в розподілених системах виконується зазвичай наступним чином. На різних вузлах системи в базах даних відбуваються зміни. В певний момент часу, коли ці зміни потрібно узгодити виконується процедура синхронізації даних двох залежних вузлів. Для цього використовуються алгоритми PUSH, PULL та PULL and PUSH. Недоліками є те, що [46]:

- вони починають працювати у чітко визначений час, а не при виникненні події;
- обмін даними виконується без врахування критично важливих даних інформаційної системи;
- вони не враховують тип операцій, які відбуваються з даними, які зберігаються на різних вузлах залежних баз даних.

Існують задачі, для яких узгодженість даних є критично важливою. Наприклад, платіжна система. Якщо клієнт знімає кошти зі свого рахунку на сервері в Києві, то ця інформація повинна максимально швидко розповсюджуватися на інші вузли, які знаходяться у Варшаві або Вашингтоні. З іншого боку, якщо клієнт змінює свою адресу, то є можливість трохи затримати узгодженість цієї інформації. Другий приклад, система оренди номерів в готелі. Процедура бронювання номеру є критично важливою з точки зору узгодженості, а зміна внутрішньої службової інформації готелю не має такої критичності [46].

Тому, задачу пришвидшення швидкодії процесу узгодження даних в розподілених базах даних треба розглядати як завдання максимально швидкого узгодження критично важливих даних в залежних базах даних. Для цього необхідно визначити критично важливі дані в системі та розробити алгоритм, який дозволяє максимально швидко їх узгоджувати [46].

Кожна інформаційна система оперує своїм набором даних, але в будь-якій з них можна виділити такі дані, які повинні бути опрацьовані в першу чергу. Наприклад, є фінансова інформаційна система в якій є різноманітна інформація: персональні дані

користувачів, користувацькі налаштування, транзакції і тому подібне. Першочергова обробка фінансових транзакцій має ключове значення для такої системи, оскільки напряму впливає на здатність виконання визначених функцій та процедур. Персональні дані користувача також важливі, але їх опрацювання трохи можна відтермінувати і вплив затримки буде не критичним. Користувацькі налаштування можна віднести до службових даних, оскільки прямого впливу вони не мають на поставлені задачі і вони можуть бути відтерміновані.

Транзакційний годинник представляє собою посередника між клієнтським інтерфейсом та підсистемою зберігання даних. До його головних задач можна віднести об'єднання даних, які відносяться до одного об'єкту, та керування чергою транзакцій з метою пришвидшення операцій пов'язаних з критично важливими даними.

Узгодження даних між різними вузлами системи виконується за наступним алгоритмом:

- всі операції зміни даних (Update, Delete) на залежних базах даних оформлюються в транзакції та максимально швидко передаються у чергу транзакційного годинника;
- всі транзакції, які змінюють репліковані дані, зберігаються у відповідній черзі та обробляються з використанням транзакційної логіки. Після цього отримується результуюча транзакція, яка передається на всі залежні вузли;
- для прискорення процесу узгодження даних критичних даних та враховуючи можливий стан гонки, кожна запис бази даних отримує свій пріоритет. В цьому випадку, критичні дані завжди будуть мати максимальний пріоритет. Послідовність обробки складних транзакцій буде виконуватись відповідно до пріоритетів змінених записів [46].

Слабким місцем є вузол, на якому працюватиме транзакційний годинник, через централізовані операції записів та видалення. При його недоступності втрачається можливість створення результуючої транзакції. На операції читання це не впливає,

оскільки вони виконуються на залежних базах даних, а транзакції залишаються у локальній черзі на кожному вузлі до моменту відновлення вузла з транзакційним годинником або переключенням на резервний з використання протоколів Paxos або Raft [46].

В складній розподіленій системі можуть бути залежні та незалежні бази даних. Бази даних будуть залежними, якщо вони зберігають репліковані дані. Наприклад, дані про банківський рахунок однієї людини чи дані про бронювання номерів готелю, які можна продивлятися та змінювати на різних вузлах розподіленої системи.

Залежні бази даних потребують узгодження, а незалежні – ні. Для їх розмежування необхідно ввести поняття матриці залежності MS , яка описується наступним чином [46]:

$$MS_{i,j} = \begin{cases} 1, \text{якщо } i \text{ та } j \text{ БД залежні} \\ 0, \text{якщо } i \text{ та } j \text{ БД незалежні} \end{cases} \quad (2.4)$$

де i – номер вузла,

j – номер запису.

Логіка роботи транзакційного годинника полягає у тому, щоб з певного набору транзакцій утворити одну єдину. Якщо серед транзакцій присутня операція видалення даних, то всі інші ігноруються і залишається вона. Якщо набір складається з операцій оновлення, то вони зливаються в результуючу транзакцію. У контексті документо-орієнтованих баз даних це означає, що буде відбуватися рекурсивне копіювання ключів та значень. Об'єднання відбувається від старішої транзакції до новішої. Графічне зображення цього процесу можна побачити на рис. 2.11 [46].

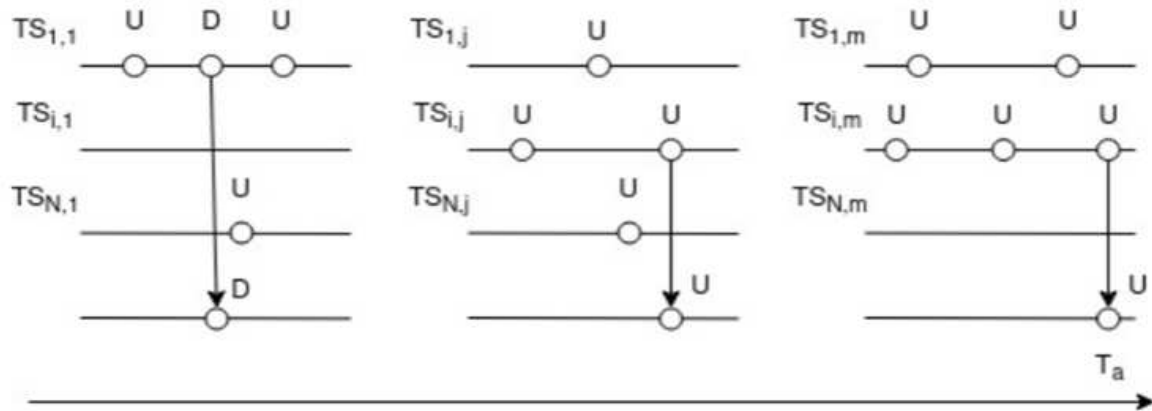


Рисунок 2.11 – Логіка роботи транзакційного годинника [46]

На цьому рисунку, $TS_{i,j}$ представляє собою транзакцію, яка відбувалась на i -му вузлі, j -го запису. Під час обробки в транзакційному годиннику їх може бути декілька. Наприклад, для транзакції $TS_{1,1}$ – відбулися послідовно операції Update, Delete, Update. Транзакційна логіка повинна проаналізувати всі складні транзакції різних вузлів для одного запису та підготувати результуючу транзакцію. Результуючу транзакцію можна визначити наступним чином:

$$U_1 \rightarrow U_2 \rightarrow \dots \rightarrow U_N = U_N$$

$$U_1 \rightarrow D \rightarrow \dots \rightarrow U_N = D$$

Слід зазначити, що транзакції накопичуються на вузлі по мірі надходження і передаються на головний вузол, якщо значення пріоритету транзакції найбільший. Особливу увагу має операція Delete, яка має більший пріоритет, оскільки видалення відбувається на всіх вузлах системи і через це, її пріоритет більше, ніж у транзакцій з іншими операціями.

Розподілена база даних складається вузлів, які активно взаємодіють з клієнтами і опрацьовують операції запису, та вузлів, які використовуються для зберігання копій. В такому контексті, ми можемо розділити вузли на дві категорії: залежні та незалежні. База даних є залежною, якщо вона використовується для зберігання копії даних у процесі реплікації.

Математично транзакційний годинник можна описати наступним чином:

$$TS_{i,j} = f(P_{g_n}, N, T_a, O, SZ, NZ), \quad (2.5)$$

де P_{g_n} – динамічний пріоритет n-транзакції;

N – номер запису;

T_a – вектор абсолютного часу транзакції;

O – операція транзакції;

SZ – вектор старих значень;

NZ – вектор нових значень.

Вектор абсолютного часу транзакції певного вузла можна розрахувати за формулою:

$$T_{a_i} = T_{S_i} - T_{O_i}, \quad (2.6)$$

де T_{S_i} – час транзакції i-го вузла;

T_{O_i} – відносна похибка i-го вузла.

Динамічний пріоритет транзакції можна представити наступною формулою:

$$P_{g_n} = f(P_{S_n}, T_{a_n}), \quad (2.7)$$

де P_{S_n} – статичний пріоритет n-го запису;

T_{a_n} – вектор абсолютного часу n-ої транзакції [46].

2.4.2 Розробка транзакційного годинника з пріоритезацією

При розробці транзакційного годинника потрібно пам'ятати, що транзакції можуть надходити від різних додатків, які мають різне географічне розташування, що може впливати на час передачі даних. Це може призвести до ситуації, коли оновлення від найвіддаленішого вузла надійде пізніше і перезапише дані, які були створені пізніше з зору логіки (рис. 2.12).

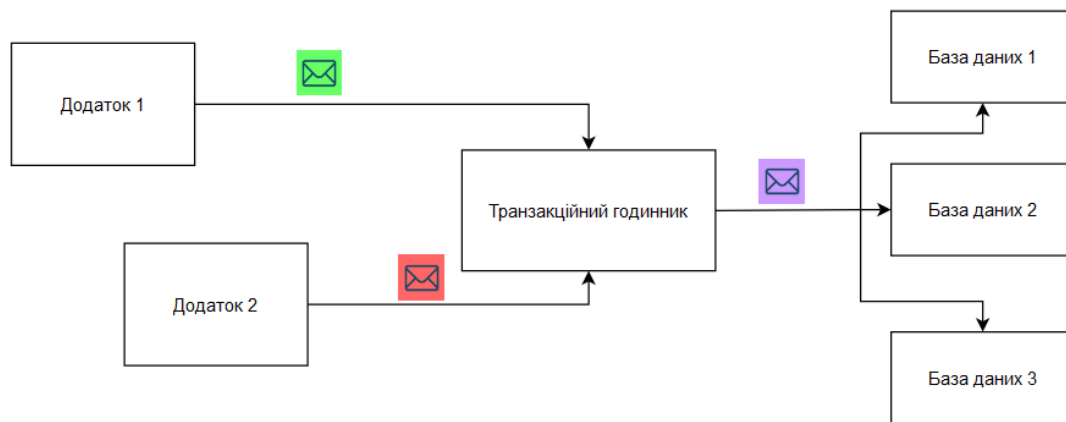


Рисунок 2.12 – Приклад загальної схеми роботи

На рис. 2.12 показані повідомлення, які відправляються різними клієнтами до транзакційного годинника. При цьому, час створення повідомлень червоного кольору раніше, ніж повідомлень зеленого кольору. Але час транспортування цих повідомлень мережею різний: зелені повідомлення надходять швидше у порівнянні з червоними.

Для впорядковування транзакцій та, по-можливості, уникнення таких ситуацій, з кожною транзакцією передається її часова мітка створення. Ця часова мітка створюється за стандартом UTC, що дозволяє уникнути проблем із різними часовими поясами. Коли відбувається процес об'єднання транзакцій, то вони впорядковуються за збільшенням та утворюється результуюча транзакція.

Всі транзакції діляться на 3и типи: CREATE, UPDATE, DELETE.

- якщо у множині транзакцій є транзакція з типом DELETE, то всі транзакції відкидаються і залишається лише вона;
- якщо множина транзакцій містить тільки транзакції з типом UPDATE, то більш нова транзакція зливається у попередню;
- якщо множина транзакцій містить CREATE, то транзакції інших типів будуть відсутні.

Після злиття транзакцій в результуючу транзакцію, остання надсилається всім вузлам розподіленої бази даних. На рис. 2.13 зображено загальну схему роботи.

Слід зазначити, що операції читання відбуваються напряму до реплік, що дозволяє розподілити навантаження між вузлами. Це особливо корисно при різному географічному розташуванні структурних компонентів. Якщо вузол знаходиться поруч із відповідним сервісом, то значно зменшується час відповіді та зменшується вірогідність помилок мережі, через які можуть відбуватися затримки. Також, це позитивно впливає на безпеку, оскільки транзакції проходять через меншу кількість мереж.

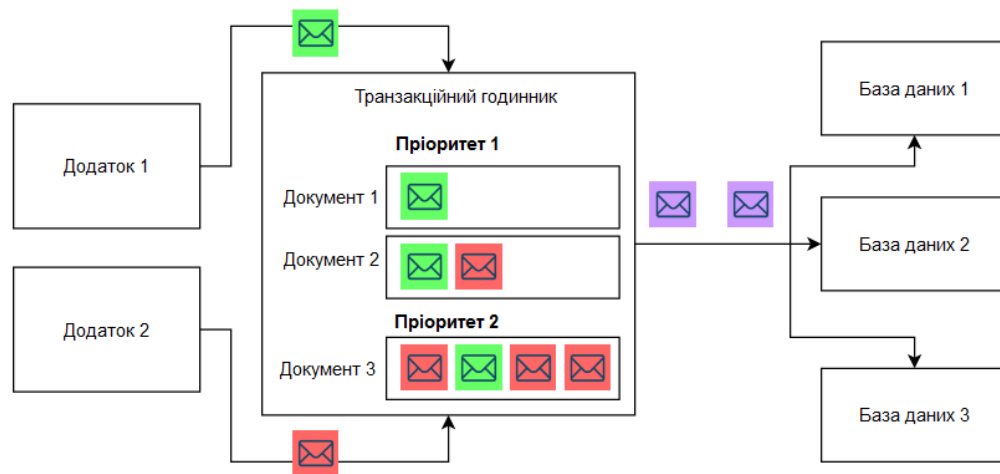


Рисунок 2.13 – Приклад загальної схеми роботи підсистеми транзакційного годинника

2.5 Модернізація процесу узгодженості даних за допомогою механізму активної антиентропії

При горизонтальному масштабуванні збільшується кількість реплік бази даних, що створює необхідність в узгодженні даних, оскільки операції запису до різних вузлів збільшує ентропію. Є різні технології, які направлені на її зменшення, серед яких активна ентиентропія. Суть її полягає у тому, щоб виявити неконсистентність та розпочати процес узгодження між репліками. Вона активно використовується у такій базі даних, як Riak та використовує структуру даних дерева Меркла, яка базується на використанні алгоритмів хешування. Швидкість ідентифікування неузгодженості

залежить від обраних алгоритмів хешування та кількості документів в колекції. Збільшення кількості документів або навіть їх розмір може погіршувати рівномірний розподіл та призводити до збільшення кількості колізій. Виникнення колізій збільшує проміжок часу неузгодженості даних, оскільки система не може вчасно виявити неконсистентність [47].

Окрім колізій, які можуть виникати, потрібно враховувати затримку через передачу даних мережею при взаємодії вузлів та пам'ятати, що така перевірка не є поодиноким операцією, а вимагає постійного обчислення на репліках та відправки для перевірки. Мінімізація часу виконання цих операцій дозволить пришвидшити процес узгодження даних.

Критично важливі дані повинні бути узгоджені з мінімальною затримкою, оскільки невчасно або неправильно прийняте рішення може призвести до матеріальних, або навіть людських втрат.

Широко поширений механізм активної антиентропії працює на основі дерева Меркла, що представляє собою деревоподібну структуру та використовує алгоритми хешування для розрахунку вузлових значень. Кореневе хеш-значення використовується для першого співставлення між двома вузлами. Потім використовуються вузлові хеш-значення і так до тих пір, поки місце неузгодженості не буде локалізовано. Після цього запускається процедура для узгодження відповідних ділянок пам'яті [47].

Цей механізм зручний та простий, що і обумовлює його широке використання не тільки в розподілених базах даних, але і в інших системах, таких як блокчейн.

Оскільки він цілковито базується на хешуванні, то однією з головних проблем є утворення колізій, що представляє собою отримання однакових хеш-значень з різних вхідних даних.

Іншою проблемою є те, що під час розрахунку дерева Меркла можуть відбуватися операції запису, які вимагають почати розрахунок дерева заново, оскільки зміна однієї складової впливає на структуру загалом. Навіть якщо в якості оптимізації

зробити так, що розраховуватись будуть тільки ті вузли, які знаходяться під впливом зміни, то використання хеш-функцій може створювати стабільне та небажане навантаження на сервер, що може погано впливати на продуктивність.

2.5.1 Модифікований механізм активної антиентропії

Спосіб узгодження у розподіленій базі даних з використанням класичного механізму активної антиентропії було розглянуто у другому розділі.

Процес узгодження даних з використанням активної антиентропії можна умовно поділити на такі кроки:

- ідентифікація неузгодженості – встановлення того факту, що на вузлах є певні об'єкти, які відрізняються один від одного;
- локалізація – визначення які саме об'єкти на вузлах відрізняються;
- виправлення – узгодження даних за певною процедурою, яка повинна вирішити, які дані мають більший пріоритет. Одним із варіантів може бути використання реальних часових міток або логічних годинників.

Для зменшення впливу недоліків хешування можна проводити процес узгодження для окремо взятих конкретних записів на різних вузлах. В цьому випадку, не потрібно займатися локалізацією, оскільки ми на самому початку знаємо дані, які перевіряються. Також, зникає ризик того, що відбудеться взаємокомпенсування при хешуванні наборів даних, що призведе до колізійної ситуації.

Для ідентифікації неузгодженості на різних вузлах необхідно використовувати спеціально розрахований знімок, який мінімізовано за розміром. Це необхідно, щоб мінімізувати час його генерації та передачі мережею.

Комунікація між вузлами буде працювати за протоколом пліток (опис та основні принципи було розглянуто у першому розділі), підтримуючи операції push та pull. Це дозволяє пришвидшити процес, хоча і може створювати певні складнощі в плані реалізації та потребувати додаткове навантаження на апаратну частину серверу. Саму систему в цьому випадку можна представити у вигляді графу. Кожен вузол системи

неперервно відправляє знімок певного запису на перевірку сусіднім вузлам. Якщо вузол, який отримав знімок, бачить різницю в результаті порівняння з локальною версією, то він надсилає назад часову мітку. Вузол, який отримав часову мітку, порівнює з локальною часовою міткою. Якщо поточний вузол має більш нову версію, то він відправляє запис. В іншому випадку, вузол нічого не робить. На рис. 2.14 зображено схему роботи механізму [47].

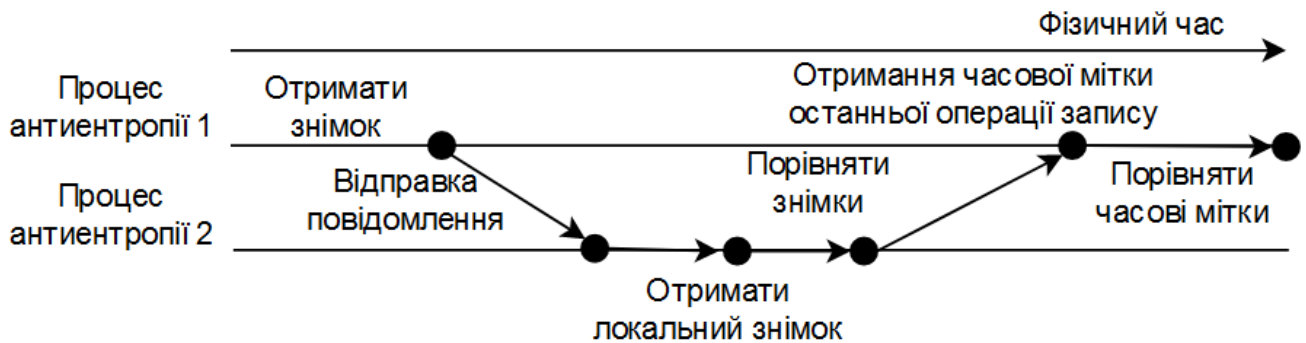


Рисунок 2.14 – Приклад роботи механізму активної антиентропії

Оскільки обмін повідомленнями відбувається неперервно, то виникають такі вимоги до повідомлень:

- знімок повинен бути стійким до колізій;
- швидкість обчислення знімку повинна бути максимальною;
- розмір знімку повинен бути мінімальним, щоб мінімізувати час необхідний для передачі між вузлами комп'ютерною мережею.

Стійкість до колізій є критично важливим показником, оскільки від нього залежить як швидко механізм зможе виявити неконсистентність і розпочати процес узгодження даних.

Як варіант, можна було б застосувати криптографічні алгоритми хешування, але вони мають властивості, які більше стосуються безпеки. Для підтримки узгодженості це не має відношення, але може негативно впливати на потреби швидкого обчислення та розміру. Інше кажучи, вони не відповідають другій та третій вимогам. У цьому

випадку, можуть бути використані некриптографічні алгоритми хешування, але для них залишається відкритим питання колізійної стійкості.

Слід зауважити, що описаний процес відбувається паралельно на всіх вузлах без виключення. Тобто, якщо вузол 1 відправляє знімок на перевірку, то і вузол 2 може в той самий момент відправляти знімок на перевірку, але не обов'язково перевірка може відноситись до того самого запису (рис. 2.15). Кожен вузол спілкується із сусідніми.

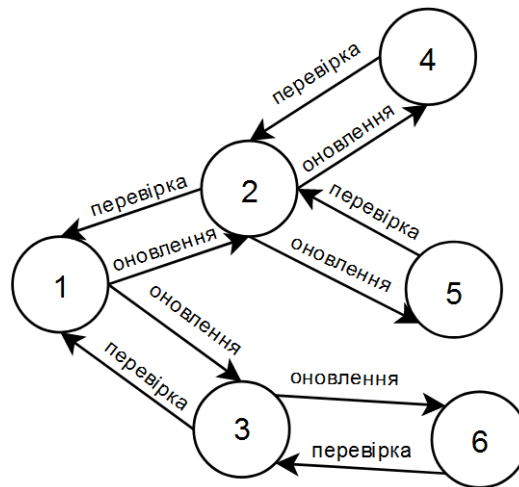


Рисунок 2.15 – Приклад комунікації вузлів при активній антиентропії

Одним з можливих варіантів для створення знімку може бути використання спектрального фільтру Блума. У класичному вигляді, він представляє собою вектор лічильників, який формується з вхідних даних. Лічильник представляє собою ціле число без знаку. Кожний вхідний блок даних хешується, отримується дайджест та з його використанням розраховується позиція у векторі. При звертанні до певного елемента цього вектора, значення лічильника збільшується на одиницю [47].

Використання спектрального фільтру Блума обумовлено тим, що ця структура даних дозволяє вставку та видалення елементів, що може надати гнучкості під час реалізації для різних типів нереляційних баз даних. Наприклад, в документо-орієнтованих базах даних це дозволяє робити вставку в фільтр тільки значень документа, ігноруючи тим часом статичні ключі (рис. 2.16).

```
{
  "firstname": "Valerii",
  "lastname": "Nikitin"
},
{
  "firstname": "Victoria",
  "lastname": "Dubinina"
}
```

Рисунок 2.16 – Приклад документів з ключами

Можливість видалення елементу може бути корисна у випадках використання для баз даних типу “ключ-значення” для перевірки та виключення узгоджених даних.

Знімок представляє собою набір байтів, які містять метадані про запис (наприклад, ідентифікатор запису) та розраховані значення спектрального фільтру Блума та алгоритму хешування. В структуру можуть входити також інші поля в залежності від типу нереляційної бази даних та якоїсь специфіки задач. Наприклад, документо-орієнтовані бази даних оперують колекціями і в структуру може бути включено додаткове поле з назвою колекції. Якщо назва колекції не фіксованого розміру, то може додаватися розмір назви колекції для правильної десеріалізації. Приклад структури зображено на рис. 2.17 [47].

OpCode	Object ID	Spectral Bloom filter	Hash
1 байт	X байтів	Y байтів	Z байтів

Рисунок 2.17 – Приклад повідомлення для перевірки записів

Один із вузлів, отримавши знімок і порівнявши їх, відправляє у разі потреби відповідь повідомлення зі структурою, як на рис. 2.18.

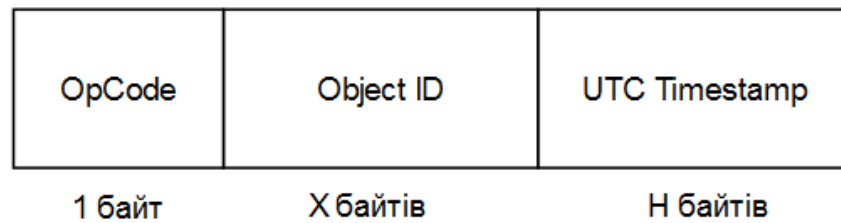


Рисунок 2.18 – Приклад структури повідомлення-відповіді після перевірки знімків

Ідентифікатор операції за розміром становить 1 байт і може приймати три значення:

- якщо надсилається повідомлення для термінування процесу, то поле приймає значення 0;
- якщо надсилається повідомлення зі знімком, то поле приймає значення 1;
- якщо надсилається повідомлення із таймстампом, то поле приймає значення 2 [47].

Ці ідентифікатори дозволяють визначити тип повідомлення та опрацювати його потрібним чином.

На рисунках присутній ObjectID, який ідентифікує певний запис у базі даних. Його розмір та формат залежить від конкретно взятої бази даних.

На рис. 2.18 присутній UTC Timestamp, який представляє собою часову мітку, коли були внесені останні зміни в документ. Він потрібен для визначення документа, який більш новий на репліках. Його розмір та формат також може бути по-різному представлений і залежить від точності, з якою час описується. Наприклад, можна використовувати таймстемп тільки з секундами, а можна також включати мікро- або наносекунди. Оскільки не всі бази даних мають спеціальний API для того, щоб отримати це значення, механізм повинен зберігати це окремо або розробники повинні додавати його в ті документи, які узгоджуються [47].

Останнє поле використовується для передачі хеш-значення. Передача хеш-значення потрібна для того, щоб уникнути колізійних ситуацій при використанні фільтру Блума.

Слід зазначити, що надсилається повідомлення з операцією термінування потреби узгодження, але повідомлення буде складатися лише з коду операції, що за розміром еквівалентно 1 байту.

Механізм використовує протоколи транспортного рівня UDP та TCP для комунікації, що зменшує час обробки повідомлень на стороні відправника та отримувача. Використання таких протоколів як HTTP, HTTPS є недоцільним, оскільки створює непотрібне навантаження при інкапсуляції та декапсуляції пакетів.

Протокол UDP використовується для всіх операцій. Протокол TCP забезпечує надійну передачу документів при виявленні неконсистентності. Для роботи механізму потрібно прослуховувати два порти одночасно для забезпечення роботи [47].

2.5.2 Математична модель реплікації механізму активної антиентропії

Для математичного опису механізму можна застосувати клітинний автомат, який нагадує поширення оновлень в системі з використанням активної антиентропії.

Клітинний автомат представляє собою просту модель обчислень, що складається з елементів пам'яті. Вона має визначену топологію, яку можна описати з використанням графу комплексної мережі $G = (V, E)$, в якій V представляє собою множину вузлів, а E – множину ребер. Таким чином, кожне входження представляє собою вузол $G: k_i \in V$ з $1 \leq i \leq n$, де n – кількість елементів множини V і локальні взаємодії між цими клітинами моделюються за допомогою ребер: існує зв'язок між k_i та k_j , якщо $(k_i, k_j) \in E$.

Кожна клітина може мати певний стан на кожному проміжку часу $t: s \in S$, де S є кінцевою множиною станів. Цей стан змінюється за допомогою локальної функції переходу, змінними якої є стани основного елементу та сусідніх елементів на попередньому кроці часу.

Сприйнятливі вузли – це ті, які ще не отримали оновлення. Оновлені вузли характеризуються тим, що вони отримали оновлення і можуть передавати його на сусідні вузли.

Коефіцієнтом клітинкового апарату є параметр передачі $0 \leq h \leq 1$, це ймовірність того, що вузол стане оновленим та матиме можливість передати оновлення іншим поєднаним вузлам. На цей параметр впливає стан мережі. Якщо мережа працює стабільно та відкинути вірогідність втрати UDP пакетів, які використовуються для передачі знімку документа, то цей параметр дорівнює 1.

Основні характеристики моделі розповсюдження:

- кожен вузол $k_i \in V$ графу G являє собою клітинку та відноситься до клітинкового апарату;
- набір станів формується двома станами: сприйнятливим (S), оновленим (U): $S = \{S, U\}$;
- клітинка, стан якої $s = U$, може передати лише пов'язаним з нею вузлам $\{k_j \mid (k_i, k_j) \in E \wedge s = S\}$;
- локальною перехідною функцією між станами є перехід від сприйнятливого до оновленого: якщо $s_t = S$, то $s_{t+1} = U$ з ймовірністю h .

Приблизний час для переходу до узгодженого стану системи можна розрахувати за формулою:

$$T = N \cdot t_s, \quad (2.8)$$

де N – кількість ітерацій, які необхідні для узгодженості системи;

t_s – час, який необхідний для поширення оновлення від вузла до вузла.

2.5.3 Модифікований метод генерації спектрального фільтру Блума

Одним із різновидів фільтру Блума є спектральний фільтр Блума. Ідея полягає в тому, щоб надати можливість не тільки для вставки, але і для видалення елементу з фільтру. Також, стає можливим підрахувати кількість елементів, які були вставлені.

Головною відмінністю є те, що масив містить не бітові значення, а лічильники у вигляді цілих невід’ємних чисел. Індеси масиву розраховуються таким самим чином, як і в класичному алгоритмі. Після отримання індесів, відповідні лічильники збільшуються на одиницю [48].

Видалення елемента відбувається так само, як і вставка, але значення відповідних лічильників зменшуються на одиницю.

На рис. 2.19 зображено фільтр Блума з лічильниками та його відмінність від класичного.

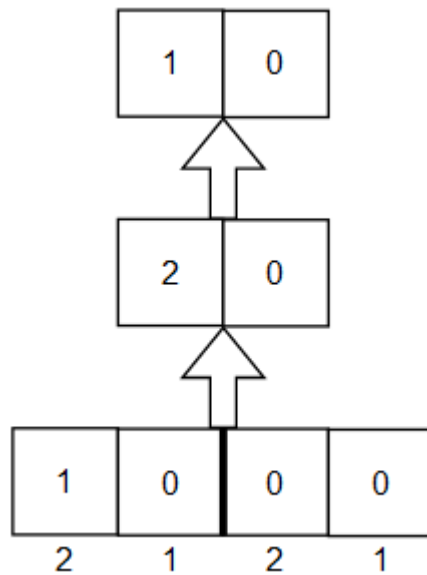


Рисунок 2.19 – Приклад спектрального фільтра Блума

Розглянемо приклад формування спектрального фільтра Блума з використанням алгоритмів Minimal Increase та Minimal Selection. Пустий фільтр зображено на рис. 2.20.

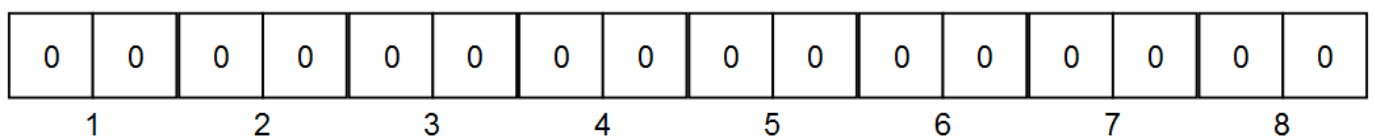


Рисунок 2.20 – Пустий спектральний фільтр Блума

Слід зауважити, що під індексом знаходиться число i в даному випадку воно кодується двома бітами. Тобто, максимально можливе значення дорівнює 3.

Нехай у нас є декілька фраз, які ми хочемо додати до фільтру: “pears are pears”, “carrots are carrots”. Розіб’ємо фрази на токени та отримаємо масив [“pears”, “are”, “pears”, “carrots”, “are”, “carrots”].

Вставляємо перші два токени: “pears”, “are”. Для цього хешуємо з використанням двох хеш-функцій та отримуємо позицію в масиві:

$$h_1(\text{“pears ”}) \% 8 = 7, h_2(\text{“pears ”}) \% 8 = 2$$

$$h_1(\text{“are”}) \% 8 = 3, h_2(\text{“are”}) \% 8 = 6$$

Робимо вставку та отримуємо фільтр як на рис. 2.21.

0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0
1		2		3		4		5		6		7		8	

Рисунок 2.21 – Фільтр після вставки перших двох tokenів

Розраховуємо індекси для наступних tokenів “pears” та “carrots”:

$$h_1(\text{“pears”}) \% 8 = 7, h_2(\text{“pears”}) \% 8 = 2$$

$$h_1(\text{“carrots”}) \% 8 = 2, h_2(\text{“carrots ”}) \% 8 = 5$$

Робимо вставку та отримуємо фільтр як на рис. 2.22.

0	0	1	0	0	1	0	0	0	1	0	1	1	0	0	0
1		2		3		4		5		6		7		8	

Рисунок 2.22 – Вставка наступних двох tokenів

Розраховуємо індекси у фільтрі для останніх двох елементів: “are”, “pears”:

$$h_1(\text{“are”}) \% 8 = 3, h_2(\text{“are”}) \% 8 = 6$$

$$h_1(\text{"pears"}) \% 8 = 7, h_2(\text{"pears"}) \% 8 = 2$$

Результат вставки зображено на рис. 2.23.

0	0	1	1	1	0	0	0	0	1	1	0	1	1	0	0
1		2		3		4		5		6		7		8	

Рисунок 2.23 – Фільтр після вставки двох останніх tokenів

Отримати кількість елементів з фільтру можна з використанням алгоритму Minimum Selection. Суть алгоритму полягає в тому, щоб обрати мінімальне значення з лічильників масиву, які відповідають певному елементу [42].

Нехай ми хочемо порахувати, яка кількість токена “pears” була додана до фільтру. Для цього потрібно отримати позиції в масиві шляхом хешування функціями h_1 та h_2 , отримати значення з них та обрати мінімальне: $\text{Кількість}(\text{"carrots"}) = \min([C(h_1(\text{"carrots"})), C(h_2(\text{"carrots"}))]) = \min([C(2), C(5)]) = \min([3, 1]) = 1$.

У контексті механізму активної антиентропії, потрібно мінімізувати кількість хешування, оскільки цей процес впливає на навантаження апаратної частини вузла. Таким чином, спектральний фільтр Блума представляє собою набір з восьми цілих невід’ємних чисел. Кожне число представляє собою лічильник, яке збільшується на одиницю при кожному зверненні до нього. На самому початку, значення всіх лічильників дорівнюють нулям [48].

Звернення до певного лічильника відбувається наступним чином:

- представити документ у вигляді байтів;
- кожен байт конвертується у ціле число без знаку;
- ділення за модулем кожного числа на набір з простих чисел від 2 до 17 включно. Якщо остача при діленні дорівнює нулю, то відповідний лічильник у фільтрі Блума збільшується на одиницю. Якщо число не було поділене на

будь-яке просте число із набору, то збільшується на одиницю останній елемент фільтру [48].

На рис. 2.24 зображено алгоритм формування фільтру для документу, який складається з трьох байтів. Даний розмір документу був обраний із метою зручності візуалізації.

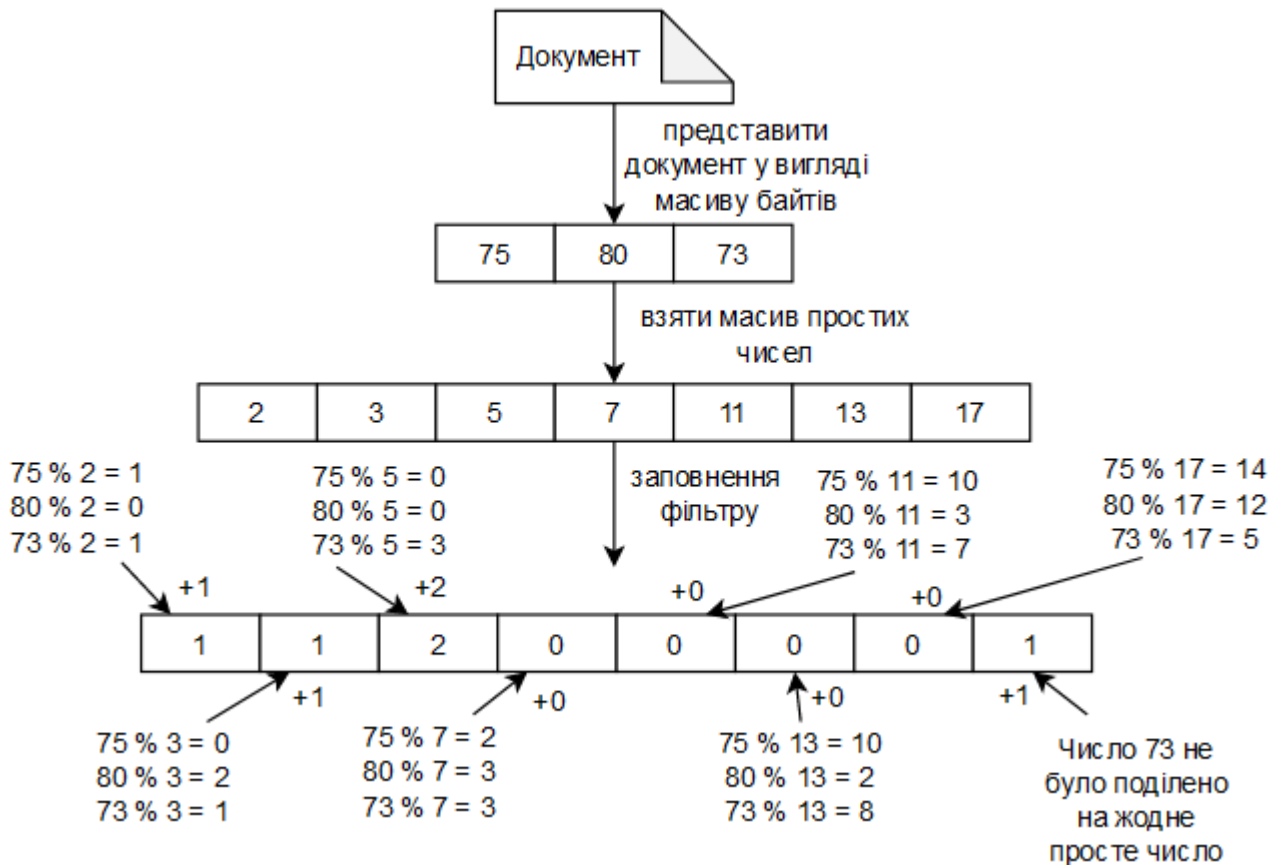


Рисунок 2.24 – Алгоритм заповнення спектрального фільтру Блума з використанням простих чисел [48]

2.5.4 Стійкий до колізій алгоритм хешування

Хешування – це отримання бітової послідовності з масиву вхідних даних довільної довжини, яке виконується за певним алгоритмом. Даний процес широко використовується у багатьох процесах різних інформаційних систем. Наприклад, при передачі інформації по мережі використовуються так звані контрольні суми, які

обчислюються з відправляємих даних та дозволяють пересвідчитись у цілісності пакетів. Іншим прикладом може бути отримання файлу по мережі. У даному випадку перевірка хеша буде відбуватися на стороні отримувача та буде звірятися з хеш-сумою, яку зазначив відправник. У свою чергу, отримувач генерує хеш з отриманого файлу та зіставляє з хешем відправника. Бази даних не є виключенням та можуть використовувати хешування для швидкого прямого доступу до зберігаємого запису на основі хеш-таблиць.

Головним недоліком алгоритмів хешування є колізії, які можуть виникати. Колізія представляє собою ситуацію, коли хеш-функція повертає однакові хеш-значення при різних вхідних масивах даних. Криптографічні алгоритми максимально спроектовані таким чином, щоб мінімізувати це в цілях безпеки. При забезпеченні цього, алгоритми хешування можуть генерувати хеш-значення відносно великого розміру: 128, 256, 512 бітів. Це не підходить для механізму активної антиентропії через те, що розрахунок хеш-значення має бути швидким, а саме хеш-значення мінімального розміру для швидкої передачі.

Хеш РН2 алгоритму складається з 6 частин. Кожна частина приймає відповідне значення залежно від масиву вхідних даних (рис. 2.25).

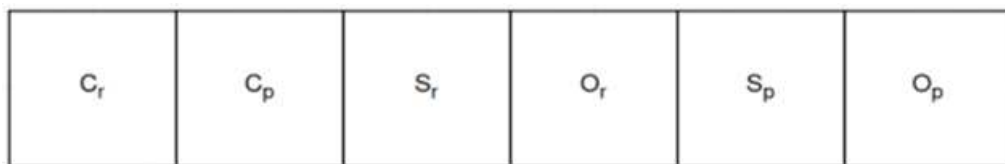


Рисунок 2.25 – Структура хеш-значення алгоритму РН2

У першій частині закодовано кількість непростих чисел. У другій частині закодовано кількість простих чисел. Третя і четверта частини являють собою суму непростих чисел і кількість переповнень через обмеження розрядності обчислювальної машини. Під переповненням ми маємо на увазі кількість обнулень через максимально допустиме значення цілого числа, оскільки фізично неможливо

описати будь-яке велике ціле число. П'ята і шоста частини такі ж, як третя і четверта частини, але з використанням простих чисел. Якщо розмір блоку буде дорівнювати 32 бітам, то розмір результуючого хеш-значення буде дорівнює 192 біта [49].

Введемо параметри, які необхідні для розрахунку хеш-значення:

- нехай існує невід'ємне число S , що характеризує розмір блоку в бітах;
- нехай існує множина простих невід'ємних чисел $P = \{2, 3, 5, \dots, p_k\}$, де кожне $p_i \leq 2^S$;
- нехай існує множина цілих невід'ємних чисел $N = \{n_1, n_2, \dots, 2^S\}$;
- нехай існує вектор цілих невід'ємних чисел $V = [b_1, b_2, \dots, b_n]$, які були отримані внаслідок перетворення блоків з вихідної бітової послідовності в числа.

Розрахувати кількість непростих чисел можна за формулою:

$$C_r = \sum_{i=1}^N 1, \text{ при умові } b_i \notin P, \quad (2.8)$$

де b_i – належить набору V .

Розрахувати кількість простих чисел можна за формулою:

$$C_p = \sum_{i=1}^N 1, \text{ при умові } b_i \in P, \quad (2.9)$$

де b_i – належить набору V .

Розраховуємо суму всіх непростих чисел:

$$S_r = \sum_{i=1}^{C_N} b_i, \text{ при умові } b_i \notin P \quad (2.10)$$

де C_N – кількість елементів множини N .

Розраховуємо суму всіх простих чисел:

$$S_p = \sum_{i=1}^{C_N} b_i, \text{ при умові } b_i \in P \quad (2.11)$$

де C_N – кількість елементів множини N .

Розрахувати кількість переповнень в результаті знаходження суми. Операцію ділення можна представити у наступному вигляді:

$$a = b \cdot c + r, \quad (2.12)$$

де a – ділене;

b – дільник;

c - неповна частка;

r - остача ($0 \leq r \leq |b|$) [49].

Кількість переповнень при знаходженні суми непростих чисел можна за формулою:

$$O_r = \sum_{i=1}^s 1, \text{ що } \forall i \in \mathbb{N} \text{ при умові } S_r - 2^s \cdot i \geq 0, \quad (2.13)$$

де S_r – розрахована сума непростих чисел.

Кількість переповнень при знаходженні суми непростих чисел можна за формулою:

$$O_p = \sum_{i=1}^s 1, \text{ що } \forall i \in \mathbb{N} \text{ при умові } S_p - 2^s \cdot i \geq 0, \quad (2.14)$$

де S_p - розрахована сума простих чисел.

Отримавши всі складові, необхідно провести конкатенацію байтів у порядку зазначеному на рис. 2.25 та отримати результуючий хеш [49].

Розглянемо приклад хешування РН2-48. Даний варіант алгоритму обумовлює використання 8 біт на блок. Це означає, що максимально можливе значення блоку може бути $2^8 = 255$ та в результаті буде отримано хеш-значення довжиною 48 біт.

Нехай в якості вхідного масиву даних буде строчка “Hello World!!”. Переведемо кожен символ у код у десятковій системі числення (табл. 2.1):

Таблиця 2.1 – Кодування символів строчки

Символ	Н	е	l	о	‘ ‘ (пробіл)	W	r	d	!
Значення	72	101	108	111	32	87	114	100	33

Отримуємо вектор чисел у десятковій системі числення: $B = [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33, 33]$.

Розіб'ємо послідовність B на два вектора P та R , в яких будуть знаходитись прості та непрості числа відповідно: $P = [101]$, $R = [72, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33, 33]$.

Тепер ми можемо розрахувати всі необхідні частини для формування хеш значення:

- кількість непростих чисел $C_r = 12$ за формулою 2.8;
- кількість простих чисел $C_p = 1$ за формулою 2.9;
- сума непростих чисел $S_r = 1017$ за формулою 2.10;
- сума простих чисел $S_p = 101$ за формулою 2.11;
- кількість переповнень непростих чисел $O_r = 3$ за формулою 2.13;
- кількість переповнень простих чисел $O_p = 0$ за формулою 2.14.

Ми розрахували всі необхідні частини і це дає можливість отримати результуючий хеш використовуючи конкатенацію та у якості роздільника між частинами використаємо символ ‘.’. Слід зазначити, що для реалізації ще потрібно слідкувати за розміром даних. Якщо розмір блоку становить 8 біт, то ми можемо покласти туди тільки числа до 255 включно. У нашому випадку, сума непростих чисел складає 1017. Для вирішення цього слід використати ділення за модулем, щоб відкинути зайве [49].

Результуючий хеш дорівнює:

$$h = C_r \cdot C_p \cdot S_r \cdot O_r \cdot S_p \cdot O_p = 12.1.252.3.101.0 =$$

$$00000100.00000001.11111100.00000011.01100101.00000000$$

Алгоритм той самий при хешуванні PH2-96, але з тою різницею, що у цьому випадку буде менша кількість блоків. Використовувати будемо ту саму строчку “Hello, World!!”.

Розмір блоку в даному випадку дорівнює 2 байтам. Вихідна послідовність має 13 байтів. Для того щоб розбити на блоки, потрібно доповнити вихідну послідовність нульовими байтами.

Отримаємо послідовність чисел у десятковій системі числення:

$$B = [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33, 33, 0]$$

Об’єднуючі сусідні байти отримуємо 7 блоків (рис. 2.26).

72, 101	01001000 01100101
108, 108	01101100 01101100
111, 32	01101111 00100000
87, 111	01010111 01101111
114, 108	01110010 01101100
100, 33	01100100 00100001
33, 0	00100001 00000000

Рисунок 2.26 – Результат розбиття на блоки

Переводимо у числа десяткової системи числення та отримуємо вектор:

$$B = [18533, 27756, 28448, 22383, 29292, 25633, 8448]$$

Розіб'ємо послідовність B на дві послідовності P та R , в яких будуть знаходитись прості та непрості числа відповідно:

$$P = [25633]$$

$$R = [18533, 27756, 28448, 22383, 29292, 8448]$$

Проведемо такі самі розрахунки, як і в прикладі для РН2-48:

- кількість непростих чисел $C_r = 6$ за формулою 2.8;
- кількість простих чисел $C_p = 1$ за формулою 2.9;
- сума непростих чисел $S_r = 134860$ за формулою 2.10;
- сума простих чисел $S_p = 25633$ за формулою 2.11;
- кількість переповнень непростих чисел $O_r = 2$ за формулою 2.13;
- кількість переповнень простих чисел $O_p = 0$ за формулою 2.14.

Ми розрахували всі необхідні частини, застосовуємо ділення за модулем для того, щоб вмістити в блок та отримуємо результуючий хеш використовуючи конкатенацію:

$$h = C_r \cdot C_p \cdot S_r \cdot O_r \cdot S_p \cdot O_p = 6 \cdot 1 \cdot 13790 \cdot 2 \cdot 25633 \cdot 0$$

Колізійних ситуацій уникнути не вдасться, але із самого алгоритму видно, що він стійкий до колізій при зміні розміру вхідного масиву даних.

Нехай є послідовність байт:

$$B_1 = [140, 230, 250, 41, 7, 250, 250, 179]$$

Результуючий хеш при використанні алгоритму PH2-48 буде дорівнювати:

$$h_1 = 5.3.100.4.227.0$$

З самого алгоритму можна побачити, що порядок байтів не має впливу на формування хеш значення. Нехай є послідовність та сама послідовність, але елементи попарно змінені місцями:

$$B_2 = [230, 140, 41, 250, 250, 7, 179, 250]$$

B_1 складається з тих самих елементів, що і B_2 . Результуючий хеш для B_1 буде таким самим, як і для B_2 :

$$h_2 = 5.3.100.4.227.0$$

Іншим варіантом колізії є переповнення, яке відбувається внаслідок перевищення допустимої місткості блоку.

Нехай існує послідовність B_3 , яка складається з:

- блоку 255 у кількості 259 блоків;
- блок 100 у кількості одного блоку;
- блок 41 у кількості одного блоку;
- блок 7 у кількості одного блоку;
- блок 179 у кількості одного блоку.

Хеш значення цієї послідовності буде виглядати як:

$$h_3 = 5.3.100.4.227.0$$

Можно побачити, що хеш значення h_3 співпадає з хешами h_1 та h_2 , які були отримані з послідовностей B_1 та B_2 . Слід зауважити, що у контексті активної антиентропії для даних бази даних вірогідність такої різкої зміни мало ймовірна [49].

Висновки до розділу 2

Даний розділ присвячено розробці нового методу узгодження даних з використанням транзакційного годинника та модернізації методу активної антиентропії.

Під час удосконалення методу активної антиентропії було вперше замінено дерево Меркла на алгоритм хешування та використано спектральний фільтр Блума для задачі забезпечення узгодженості даних. Для його ефективного використання було модифіковано алгоритм формування фільтру, що дозволило покращити швидкість його формування і зберегти на достатньому рівні колізійну стійкість.

Також, було розроблено спеціальний алгоритм хешування, який стійкий до колізій при різних розмірах вхідних масивів даних та має близьку швидкість формування до некриптографічних алгоритмів. Слабким місцем є колізійна стійкість при хешуванні даних однакового розміру, але це не є критичним в контексті роботи в механізмі активної антиентропії через використання в поєднанні зі спектральним фільтром Блума.

Розроблений метод активної антиентропії базується на протоколі пліток для комунікації з іншими вузлами, що забезпечує розподіл навантаження під час процесу узгодження даних рівномірно серед всіх вузлів розподіленої бази даних.

З ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНИХ МЕТОДІВ ТА ТЕХНОЛОГІЙ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ УЗГОДЖЕНОСТІ ДАНИХ В ІНФОРМАЦІЙНИХ СИСТЕМАХ З РОЗПОДІЛЕНОЮ БАЗОЮ ДАНИХ

3.1 Розробка інтегрального середовища фінансової інформаційної системи для проведення експериментального дослідження

Проведення досліджень з використанням фінансової інформаційної системи обумовлено тим, що вона має максимально ефективно підтримувати узгодженість даних, оскільки це впливає на час критично важливих процесів на рівні держави та окремо взятої особи. Податкова система розподілена по різних областях та містах країни, що вимагає спеціалізованих методів та способів, які дозволять оптимально отримувати дані та коректно оцінювати ситуацію. Сплата податків юридичними особами може відбуватися частинами та через різні банківські установи, які географічно розташовані в різних місцях, але це не повинно впливати на кінцеву узгодженість самої системи. Закриття або відкриття рахунків можуть займати велику кількість часу через необхідність проведення спеціального бухгалтерського обліку. Цей процес має бути максимально узгодженим та надійними задля уникнення конфліктів.

Як зазначалося раніше, фінансові системи повинні відповідати високим вимогам. Це обумовлено не тільки задачею користування кінцевими користувачами, але й певними обмеженнями зі сторони держави. Фінансова установа повинна слідувати розробленими державою правовим нормам та актам, і однією з потреб може бути проведення внутрішнього фінансового моніторингу з метою уникнення та виявлення протиправних дій. Розподіл даних та реплікація дозволяють мати окремі вузли з копіями для розподілу навантаження. В даному випадку, окремі вузли можуть використовуватись для відділу фінансового моніторингу з можливістю відслідковування підозрілої активності та навіть блокуванням облікових записів за необхідністю.

Одним із варіантів такої системи може бути система, яка складається з вузла з програмним забезпеченням для безпосередньої взаємодії клієнтів, вузла з програмним забезпеченням для проведення моніторингу та восьми вузлів розподіленої бази даних. Перший додаток взаємодіє з однією з реплік, але у разі необхідності може переключитись на будь-який інший. Це забезпечує надійність системи та покращення швидкодії.

3.1.1 Бізнес-логіка інтегрального середовища фінансової інформаційної системи

Бізнес-логіка починається з того, що людина повинна бути зареєстрованою в системі. Для цього вона передає деякі персональні дані, такі як ім'я, прізвище, дата народження та інше. Після цього, створюється обліковий запис, до якого прив'язується ця інформація. Обліковий запис потрібен для прив'язки рахунків, активації або деактивації послуг з обслуговування. Це дає можливість створювати кредитні картки з їх подальшою прив'язкою до облікового запису. Для реєстрації картки потрібно передати наступні дані: номер картки, cvv2 код, дата просрочення. Баланс нової картки становить нуль і буде збільшуватись або зменшуватись в залежності від типу транзакції. Загалом, їх може бути дві: кредитування при знятті коштів та переказі на інший рахунок або дебетування, якщо кошти надходять на рахунок. При використанні картки, незалежно від типу операції, у системі створюється транзакція яка не підлягає ніякими змінам. Тобто, транзакція є цілісною одиницею, яка зберігається протягом часу. Це потрібно для забезпечення максимальної надійності та лінійності. При виникненні якоїсь аварійної ситуації на стороні користувачів, буде створена компенсаційна транзакція, яка дозволить відкотити стан до проведення транзакції. Це може виникати при наявності певних мережевих проблем або при збоях під час взаємодії відповідних підсистем.

Ще в якості службової інформації є дані про термінали самообслуговування та відділення фінансової установи. Термінал самообслуговування прив'язується до відділення та в якості службової інформації слугує статус працездатності та кількість

готівки. Відділення має дані про графік роботи, адресу розташування та які системи самообслуговування в ньому знаходяться.

Короткий варіант функціональної схем зображено на рис. 3.1.

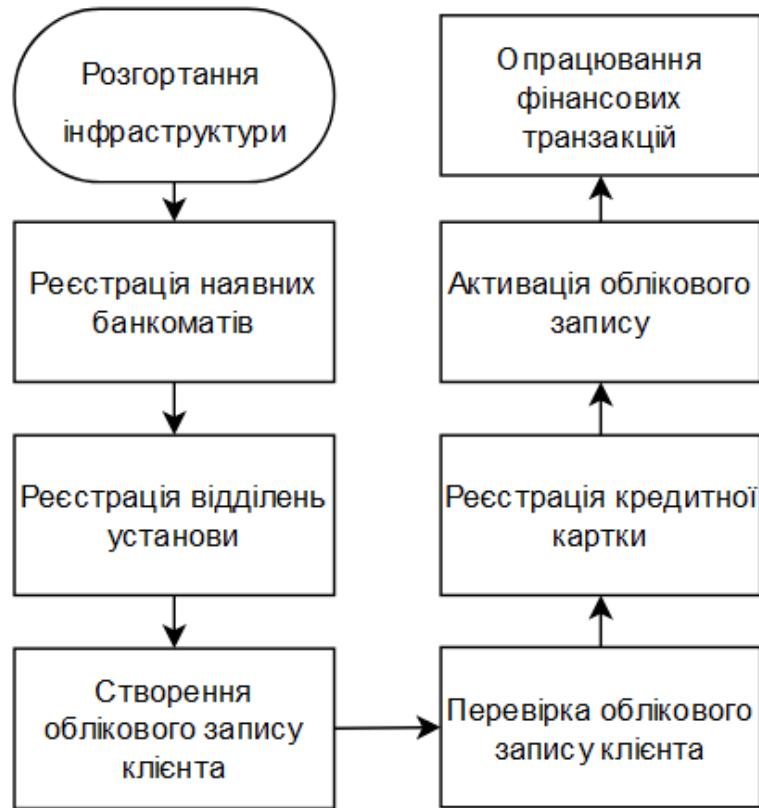


Рисунок 3.1 – Функціональна схема роботи системи

3.1.2 Опис структурних складових системи

Інформаційна система складатиметься з додатку для потенційних користувачів, додатку для персоналу внутрішнього відділу та розподіленої бази даних, яка складається з восьми вузлів. Слід зазначити, що структура розподіленої бази даних та зв'язків буде змінюватись в залежності від тих методів, які будуть використовуватись.

При використанні централізованого підходу буде головний вузол з яким буде відбуватися взаємодія. Всі операції запису будуть йти через нього, а операції читання – через репліки. На рис. 3.2 можна побачити структурну схему.

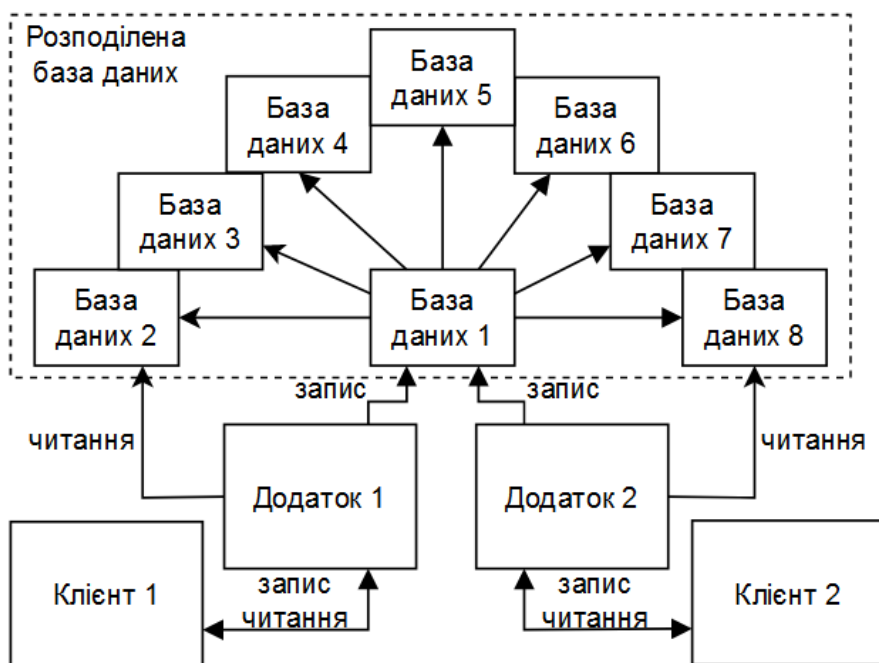


Рисунок 3.2 – Структурна схема з використанням MongoDB ReplicaSet

Структурна схема буде трохи відрізнятися при використанні транзакційного годинника. Обидва додатки будуть взаємодіяти з транзакційним годинником та зчитувати дані з різних вузлів розподіленої бази даних (рис. 3.3).

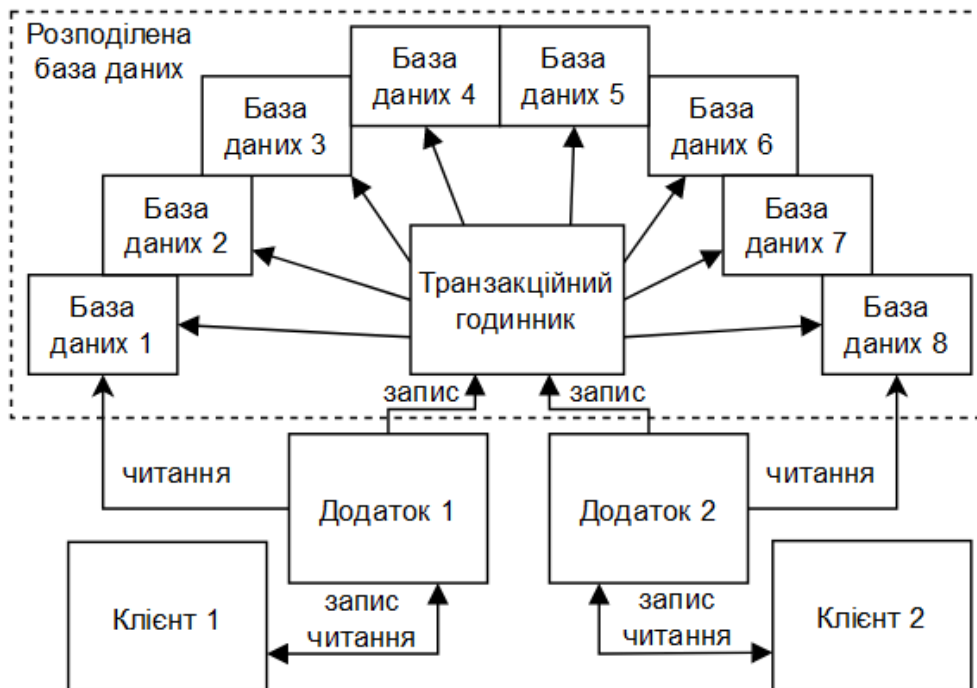


Рисунок 3.3 – Структурна схема з використанням транзакційного годинника

Остання схема буде суттєво відрізнятися від двох попередніх, оскільки тут використовуватимуться метод активної антиентропії. Додатки підключаються до різних реплік, а репліки взаємодіють одна з одною (рис. 3.4).

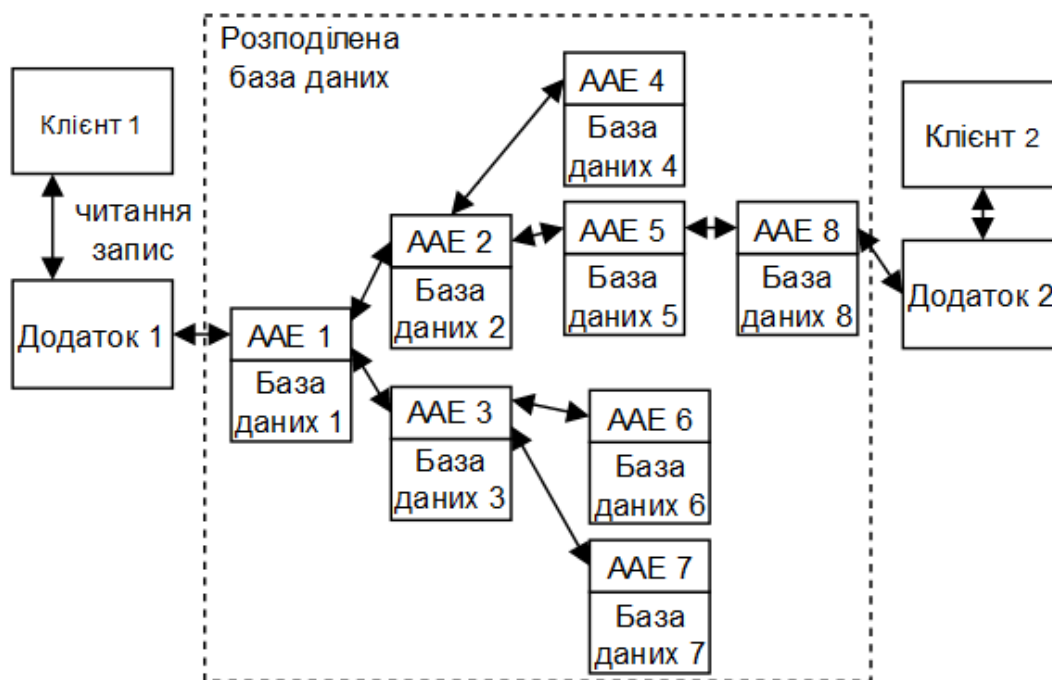


Рисунок 3.4 – Структурна схема з використанням активної антиентропії

3.1.3 Проектування інтегрального середовища фінансової інформаційної системи

Для порівняння розроблених методів та способів необхідно спроектувати та реалізувати релевантну інформаційну систему, яка дозволить якісно змоделювати процес узгодження даних. Для цього було створено UML діаграму для більш глибокого розуміння складності впровадження механізмів узгодження (рис. 3.5).

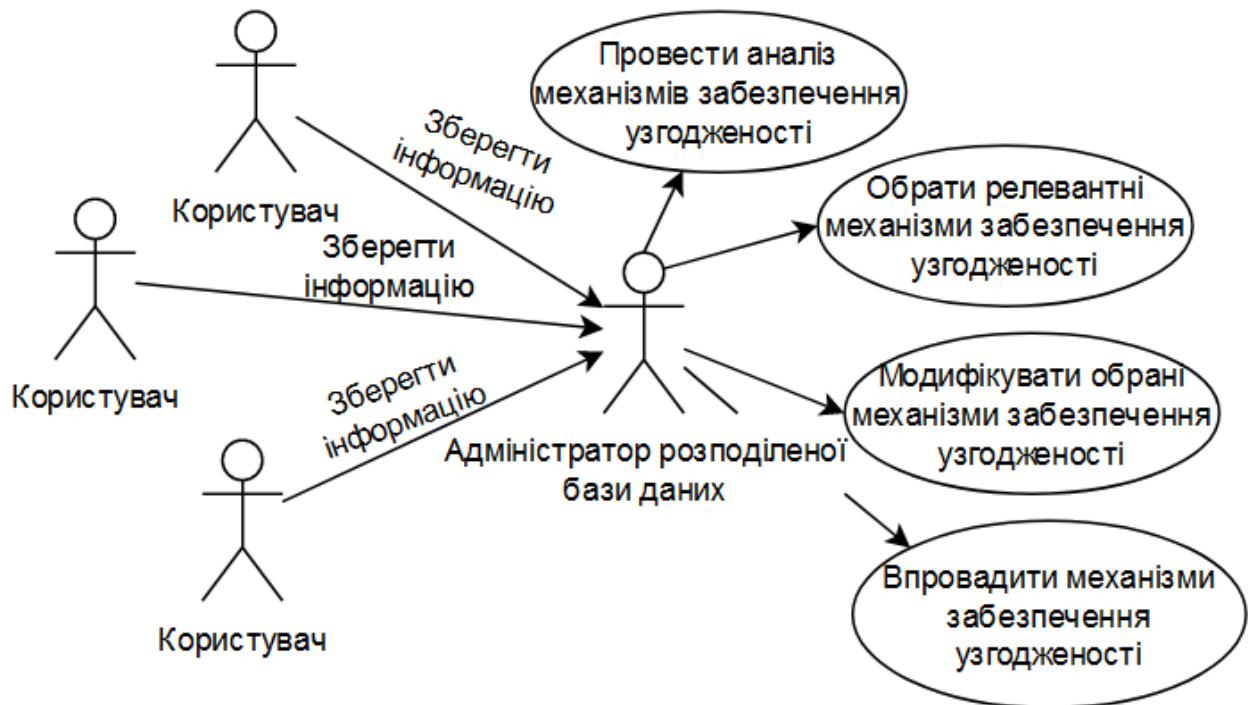


Рисунок 3.5 – UML діаграма варіантів використання при впровадженні механізмів підтримки узгодженості

Як показано на рис. 3.5, користувачі інформаційної системи, які побудовані на клієнт-серверній архітектурі, зберігають різноманітні дані. Розподілена база повинна зберегти дані на локальному вузлі та передати на інші. Під час передачі даних працюють механізми підтримки узгодженості, які виконують пошук неузгодженостей, локалізацію та виправлення.

У цьому випадку, архітектуру розподіленої бази даних можна представити у вигляді чотирьох шарів (рис. 3.6):

- рівень розподіленої бази даних складається з певного набору екземплярів бази даних, які будуть зберігати користувачську інформацію;
- рівень сервісу механізму підтримки узгодженості представляє собою шар між розподіленою базою даних та серверним додатком. Він реалізує той чи інший метод забезпечення узгодженості розподілених даних. Якщо це транзакційний годинник, то він буде напряму робити операції запису в

репліки. Якщо це механізм активної антиентропії, то він буде з'єднаний з чітко визначеною базою даних, в яку буде робити записи. Запис в інші бази даних буде реалізовуватись з використанням в результаті спілкування цього сервісу з іншим;

- шар серверного додатку реалізує всю необхідну бізнес-логіку інформаційної системи (збереження даних, читання та їх видалення);
- шар інтерфейсу користувача дозволяє кінцевим користувачам взаємодіяти з інформаційною системою з використанням стандартних засобів (наприклад, браузер).

Шар серверного додатку включає в себе наступні модулі:

- модуль вебсервісу – забезпечує взаємодію користувачів з інформаційною системою шляхом реалізації необхідного чітко визначено програмного інтерфейсу і працюючого з використанням протоколів HTTP/HTTPS на прикладному рівні еталонної моделі OSI. Можна сказати, що це вхідна точка при взаємодії із системою;
- модуль реалізації бізнес-логіки – цей модуль програмно реалізує всі необхідні операції бізнес-логіки інформаційної системи. Оскільки це фінансова інформаційна система, то тут виконуються операції отримання квартальних виписок, реєстрація кредитних карток, блокування облікових записів, опрацювання фінансових транзакцій;
- модуль зберігання даних – реалізує чітко визначений інтерфейс для взаємодії з програмним інтерфейсом конкретного сервісу забезпечення узгодженості даних;
- модуль читання даних з репліки – реалізує програмний інтерфейс для роботи з екземпляром бази даних для виконання операцій читання. Ці операції виконуються напряму, що дозволяє зменшити витрати часу на мережу та проміжкові сервіси.

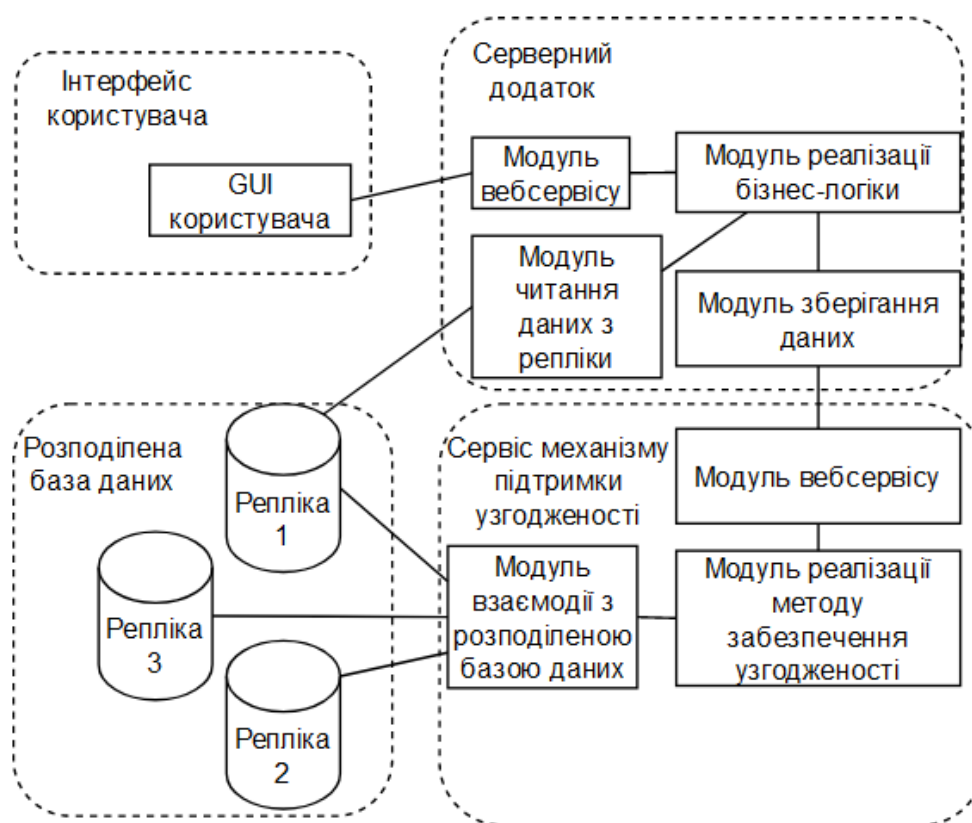


Рисунок 3.6 – Архітектура впровадження методів забезпечення узгодженості

Шар сервісу механізму підтримки узгодженості складається з наступних модулів:

- модуль вебсервісу – надає програмний інтерфейс для взаємодії із сервісом, виконує перевірку корисного навантаження, яке передається разом із запитом. При успішній десеріалізації, передає програмні об'єкти до модулю з реалізацією;
- модуль реалізації методу забезпечення узгодженості – у випадку транзакційного годинника з пріоритезацією, відповідає за додавання елементів у черги, їх опрацювання згідно пріоритетів, об'єднання транзакцій в результуючі транзакції та передачу останніх модулю взаємодії з розподіленою базою даних;

- модуль взаємодії з розподіленою базою даних – реалізує програмний інтерфейс для роботи з екземпляром бази даних і відповідає за транспортування цих даних.

Слід зауважити, розподілена база даних не є гетерогенною і містить тільки екземпляри однієї бази даних.

На рис. 3.7 можна побачити UML діаграму головної частини класів, які необхідно реалізувати для функціонування інформаційної системи.

До модулю вебсервісу відносяться наступні класи:

- TransactionsController – містить реалізацію обробки запитів, які пов’язані з фінансовими транзакціями. До них відносять функції, які дозволяють зафіксувати транзакцію та отримати деталі транзакції;
- AccountController – містить реалізацію обробки запитів для виконання операцій реєстрації, читання та модифікування сутностей, які пов’язані з обліковим записом;
- BankController – реалізує дії, які пов’язані з відділеннями банків та банкоматами;
- TransactionJSON – один з представників класів, які використовуються для серіалізації та десеріалізації. У конкретному випадку цього класу, то він описує модель даних, які передаються під час запиту та досеріалізуються в об’єкт цього класу.

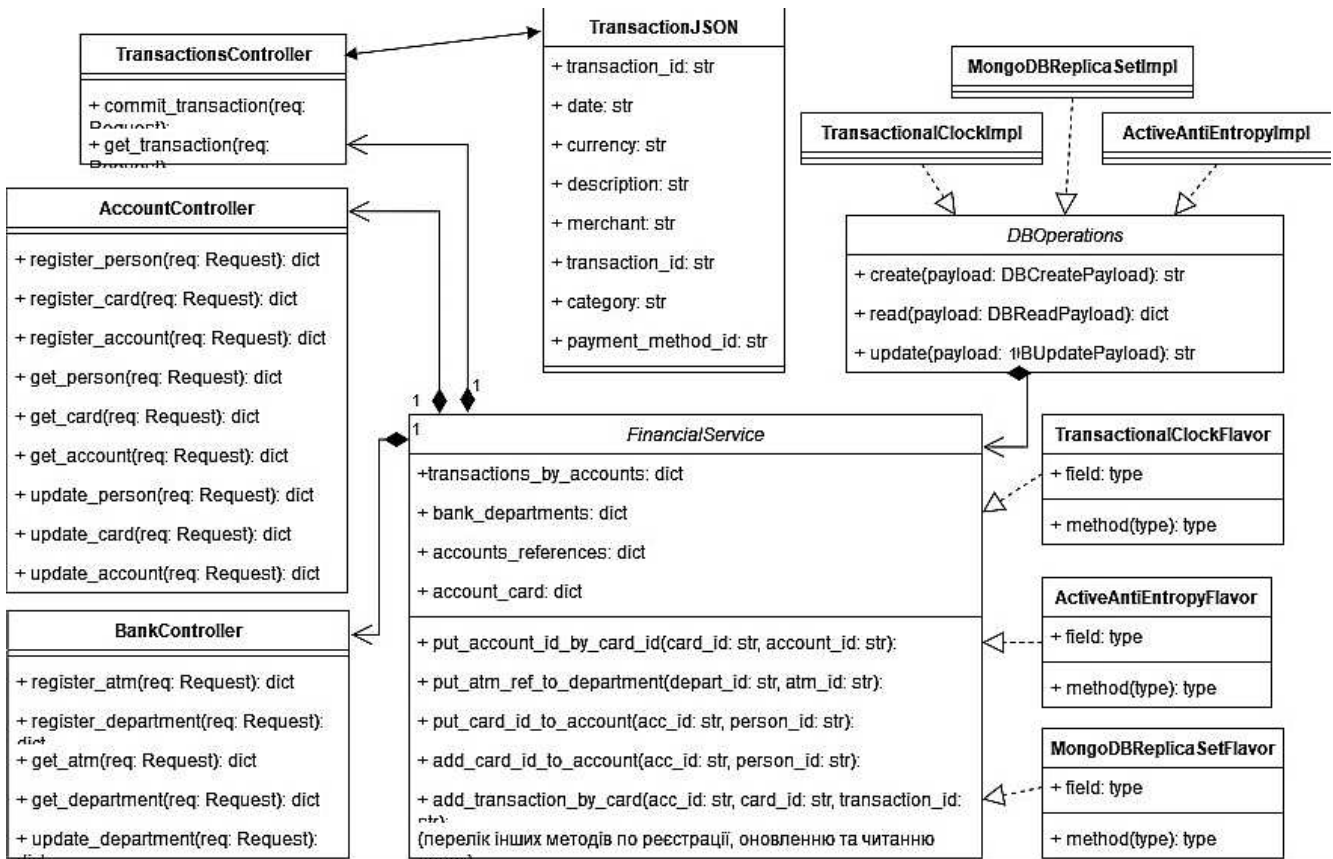


Рисунок 3.7 – UML діаграма головних класів інформаційної системи

До модулю реалізації бізнес-логіки відносяться наступні класи:

- Абстрактний клас *FinancialService* – містить реалізацію основної логіки роботи серверного додатку та абстрактні методи для реалізації класами, які будуть наслідуватись;
- *TransactionalClockFlavor* – реалізує абстрактні методи класу *FinancialService* з врахуванням особливостей використання транзакційного годинника для роботи з розподіленою базою даних. Використовує об'єкт класу *TransactionalClockImpl* в якості драйверу, який реалізує комунікацію через протокол HTTP прикладного рівня моделі OSI;
- *ActiveAntiEntropyFlavor* – реалізує абстрактні методи класу *FinancialService* з врахуванням особливостей використання активної антиентропії для роботи з розподіленою базою даних. Використовує об'єкт класу

ActiveAntiEntropyImpl в якості драйверу, який реалізує комунікацію через протокол TCP транспортного рівня моделі OSI;

- MongoDBReplicaSetFlavor – реалізує абстрактні методи класу FinancialService з врахуванням особливостей використання активної антиентропії для роботи з розподіленою базою даних. Використовує об'єкт класу MongoDBReplicaSetImpl в якості драйверу.

До модулів зберігання даних та читання даних з реплік відносяться:

- DBOperations – представляє собою абстрактний клас, який надає інтерфейс для уніфікації операцій з розподіленою базою даних;
- TransactionalClockImpl – реалізує виконання запитів до програмного інтерфейсу сервісу транзакційного годинника для операцій запису. Операції читання здійснюються напряду до репліки розподіленої бази даних;
- MongoDBReplicaSetImpl – реалізує виконання запитів до MongoDB ReplicaSet з використанням бібліотеки pymongo;
- ActiveAntiEntropyImpl – реалізує виконання запитів до програмного інтерфейсу сервісу активної антиентропії для операцій запису. Операції читання здійснюються напряду до репліки розподіленої бази даних.

3.1.4 Структура проєкту та реалізація серверного додатку

Для прототипування запропонованої інформаційної системи було вирішено використовувати мову програмування пайтон, яка надає необхідні можливості для цього, у вигляді широкого спектру бібліотек та фреймворків, простий синтаксис та широкий спектр можливостей для проєктування.

Серверний додаток умовно можна розділити на шар вебсервісу для взаємодії, шар сервісу та шар взаємодії із розподіленою базою даних. Шар вебсервісу представляє собою інтерфейс для взаємодії із системою та забезпечує клієнт-серверну взаємодію з використанням протоколу прикладного рівня HTTP. При

передачі корисного навантаження використовується формат JSON, який дозволяє зручно та просто оперувати даними.

До програмних кінцевих точок можна віднести головні:

- `atms`

Цей маршрут використовується для оперування інформацією банкоматів у системі. До ключової інформації відноситься стан, розташування, підтримувані операції, валюти та мови. При використанні методів PUT та GET необхідно передавати ID об'єкту в заголовках HTTP запиту. Підтримує HTTP методи POST, GET, PUT;

- `departments`

Використовується для оперування інформацією про відділення фінансової установи. До ключової інформації відносяться адреса, робочі години та ідентифікатори пов'язаних банкоматів. При використанні методів PUT та GET необхідно передавати ID об'єкту в заголовках HTTP запиту. Підтримує HTTP методи POST, GET, PUT;

- `persons`

До основної інформації, яку можна змінювати з використанням цієї кінцевої точки, відносяться ім'я, прізвище, дата народження, стать, електронна адреса, номер телефона. При використанні методів PUT та GET необхідно передавати ID об'єкту в заголовках HTTP запиту. Є підтримка HTTP методів POST, GET, PUT;

- `Accounts`

До основної інформації відносяться інформація про кредитні картки, статус та тип. Дозволяє реєструвати акаунти та змінювати інформацію про них. При використанні методів PUT та GET необхідно передавати ID об'єкту в заголовках HTTP запиту. Підтримуються HTTP методи POST, GET, PUT;

- `transactions`

Дає змогу працювати з транзакціями і оперувати такими даними, як дата створення, сума, валюта, коментар, платіжний спосіб та тип транзакції. При використанні методу GET необхідно передавати ID об'єкту в заголовках HTTP запиту. Підтримуються HTTP методи POST та GET.

Шар взаємодії з розподіленою базою даних також реалізовано у трьох версіх для кожного методу узгодження даних. Це є необхідністю, оскільки кожен метод має свій інтерфейс для взаємодії.

Шар сервісу представляє собою інтерфейс для взаємодії всередині додатку для виконання бізнес-логіки системи. Його було спроектовано таким чином, щоб забезпечити гнучкість використання того чи іншого способу взаємодії з розподіленою базою даних, оскільки сама система необхідна для порівняння трьох методів забезпечення узгодженості даних: підхід у MongoDB ReplicaSet, забезпечення узгодженості з використанням транзакційного годинника та використання активної антиентропії на основі протоколу пліток.

3.1.5 Створення бібліотеки для взаємодії із програмним інтерфейсом реалізованої інформаційної системи

Оскільки інформаційна система має власний програмний інтерфейс, то для зручності автоматизації виконання експериментів потрібно мати спеціальну бібліотеку, яка має доступ до бізнес-логіки інформаційної системи.

Бібліотека реалізовувалась мовою програмування Python3 із застосуванням функціонального програмування та використанням бібліотеки requests, яка дозволяє виконувати HTTP-запити та отримувати відповіді. Ця бібліотека має зручний та простий інтерфейс з можливістю налаштування всіх необхідних параметрів, таких як заголовки, тіло запиту, HTTP-методу (рис. 3.8).

```

└─ vnikitin
def create_atm(atm: ATMJSON) -> str:
    data = to_dict_recursive(atm.__dict__)
    payload = json.dumps(data)

    resp = requests.post(f"{url_prefix}atms", data=payload).json()

    return resp['id']

└─ vnikitin
def update_atm(_id: str, atm: ATMJSON) -> str:
    data = to_dict_recursive(atm.__dict__)
    payload = json.dumps(data)

    headers = {
        'Id': _id
    }

    resp = requests.put(f"{url_prefix}atms", headers=headers, data=payload).json()

    return resp['id']

```

Рисунок 3.8 – Частина коду реалізації драйверу до програмного інтерфейсу розробленої системи

3.1.6 Графічний інтерфейс реалізованого середовища

Розроблена інформаційна система дозволяє взаємодіяти з використанням спеціальних засобів, таких як програма Postman (рис. 3.9).

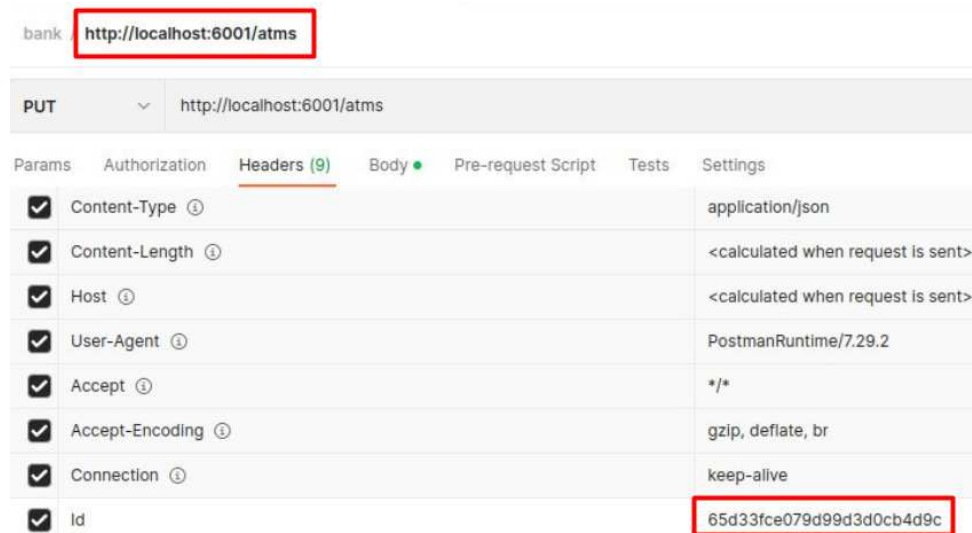


Рисунок 3.9 – Виконання запиту до інформаційної системи з використанням програми Postman

Для зручної взаємодії із системою було вирішено створити веб-інтерфейс, який дозволяє виконувати всі основні операції при взаємодії із системою. Головне вікно зображено на рис. 3.10.

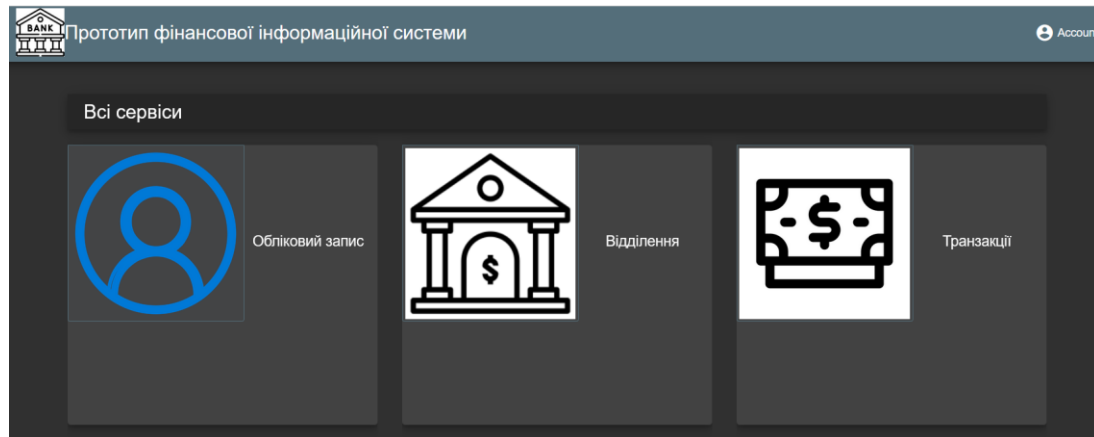


Рисунок 3.10 – Головне вікно веб-інтерфейсу для взаємодії із системою

Головні операції з системою виконуються при натисканні на відповідну кнопку. При натисканні на кнопку “Обліковий запис”, відкривається спеціальне вікно з можливістю переглядати та редагувати основну інформацію щодо акаунту. Теж саме стосується і кнопки “Відділення”.

При натисканні на кнопку “Транзакції” виводиться список останніх 10 транзакцій та можливість здійснити фінансову транзакцію. Для цього необхідно ввести необхідну інформацію (рис. 3.11).

Метод сплати



Card



PayPal

Інформація про кредитну картку

1234 1234 1234 1234






MM / YY

CVC



Власне ім'я платника

Full name on card

Країна

Ukraine

▼

Pay

Рисунок 3.11 – Створення фінансової транзакції

3.2 Програмна реалізація транзакційного годинника з пріоритезацією

3.2.1 Архітектура транзакційного годинника

Сама архітектура може бути використана і для створення транзакційного годинника для інших баз даних, оскільки представляє собою загальну схему. Одним із можливих рішень могло б бути безпосереднє впровадження транзакційного годинника у вихідний код MongoDB, але даний підхід вимагає детального розуміння вихідного коду, зосередженість тільки на даній базі даних, неможливість створення рішення для гетерогенних систем. Виходячи з цього, було вирішено створити окремий сервіс із власним програмним інтерфейсом. Схема зображена на рис. 3.12.

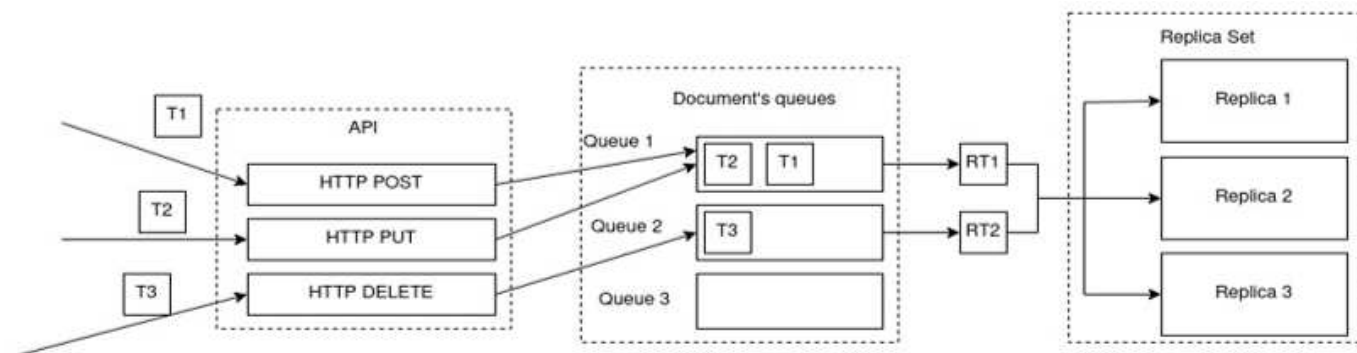


Рисунок 3.12 – Схема реалізації транзакційного годинника у вигляді сервісу

Сам сервіс можна логічно розділити на три шари: endpoints, processing та data access.

Шар endpoints містить вхідні точки доступу до сервісу. Для стандартизації та більш зручного використання, вони реалізовані у вигляді REST API поверх протоколу прикладного рівня HTTP (табл. 3.1).

Таблиця 3.1 – Програмний інтерфейс для взаємодії із транзакційним годинником

Назва	Метод	Path	Опис
create	POST	/mongodb	Використовується для створення запису
update	PUT	/mongodb	Використовується для оновлення полів існуючого запису
delete	DELETE	/mongodb	Використовується для видалення запису

Для зручності, деякі ключові параметри були винесені в хедери HTTP запитів. Хедери можуть відрізнятися один від одного в залежності від типу операції:

- **Database** – назва бази даних, до якої відноситься операція. Це представляє собою строковий тип даних. Наприклад, при роботі з базою даних якогось магазину, потрібно буде передати значення “shop”;
- **Collection** – назва колекції, до якої відноситься операція. Значення передається строкового типу даних. Якщо операція пов’язана з даними користувача, то може передаватися значення “users”;
- **Id** – ідентифікатор запису, до якого відноситься операція. Значення залежить від бази даних, оскільки кожна з них може використовувати різні формати. MongoDB використовує 12 байтовий ідентифікатор строкового типу даних. Наприклад, у контексті використання MongoDB, може бути потрібно передати “39aa9936da9c5d99f870ad38”;
- **CreatedAt** – час створення операції за стандартом UTC. Передається як строкове значення. В якості прикладу, можна навести часову мітку “2023-04-04T05:00:30.001000+09:00”;
- **Priority** – значення пріоритету операції. Може приймати будь-які додатні значення. Представляє собою цілочисельний тип даних. Наприклад, можна передати пріоритет зі значенням 50. Слід зазначити, що операція із пріоритетом 1 є найбільш пріоритетною. Чим більше значення, тим менше пріоритет операції. Значення за замовчуванням дорівнює 50.

Кожна операція має свій набір необхідних та опціональних хедерів. В табл. 3.2 наведено співставлення того чи потрібно передавати хедер для певної операції.

Таблиця 3.2 – Обов’язковість передачі хедера у запиті

Хедер/операція	create	update	delete
Database	Потрібно	Потрібно	Потрібно
Collection	Потрібно	Потрібно	Потрібно
Id	Не потрібно	Потрібно	Потрібно
CreatedAt	Не потрібно	Потрібно	Не потрібно
Priority	Опціонально	Опціонально	Опціонально

Шар processing відповідає за реєстрацію операцій, їх обробку, злиття та відправку результуючої транзакції. Загалом, в реалізації є три черги: необроблені транзакції, оброблені та результуючі транзакції. Черга з необроблених транзакцій містить транзакції, які відносяться до будь-якої бази даних, колекції, сутності. Сховище із обробленими транзакціями представляє собою асоціативний контейнер, який має індексацію за ключами, які представляють собою незмінні типи даних. Приклад ієрархії можна побачити на рис. 3.13.

З кожним ідентифікатором (ID) пов’язан певний список із транзакціями, які об’єднуються у результуючу транзакцію, яка після цього переходить до черги результуючих транзакцій. Опрацювання йде від найбільш пріоритетних записів до найменш пріоритетних.

Шар data access забезпечує зв’язок між годинником та репліками розподіленої бази даних. Його задача полягає у тому, щоб проводити операції запису, читання та видалення результуючих транзакцій без прив’язки до роботи годинника.

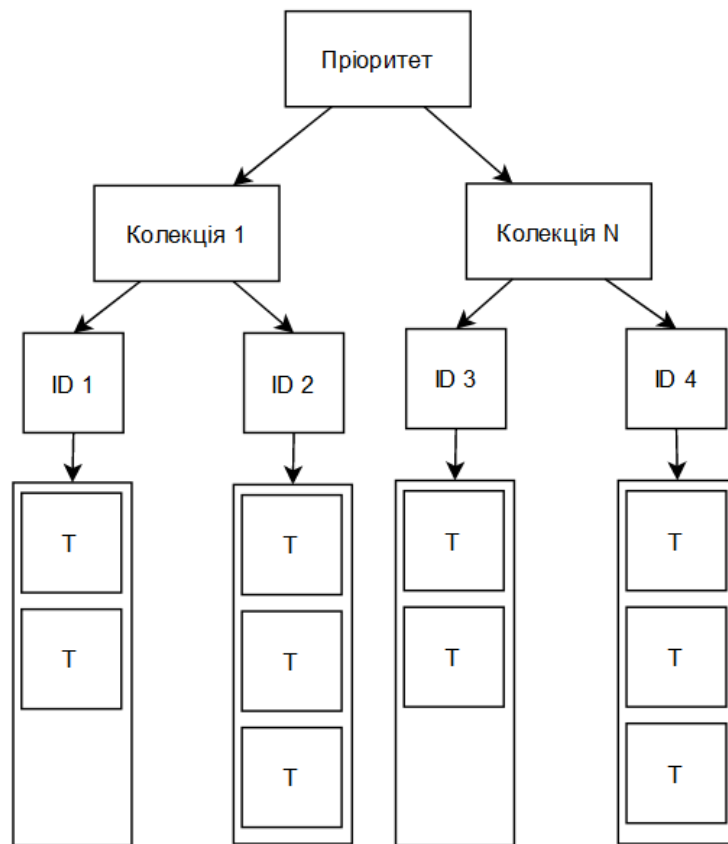


Рисунок 3.13 – Приклад ієрархії зберігання транзакцій

3.2.2 Вибір необхідних засобів для реалізації транзакційного годинника

Для реалізації транзакційного годинника потрібні наступні інструменти:

- мова програмування для реалізації API та основної логіки додатку;
- інструмент контейнеризації та оркестрації для зручного розгортання розподіленої бази даних;
- система контролю версій для зручності розробки;
- сервіс для зберігання коду.

Для реалізації API була обрана мова програмування Python 3.10.8, оскільки відповідає наступним вимогам:

- наявність драйверів до найбільш популярних баз даних;
- багатий вибір бібліотек для реалізації API;
- синтаксично проста для концентрації уваги саме на ідеї.

Для контейнеризації було обрано Docker через поширеність та зручність, а в якості оркестратора – Docker Compose.

Для зручної роботи з версіями була обрана технологія Git, а платформа для зберігання вихідного коду – Github.

Як було зазначено раніше, транзакційний годинник складається з трьох шарів, кожен з яких використовує необхідні фреймворки та бібліотеки. Основні залежності зазначені в табл. 3.3.

Таблиця 3.3 – Необхідні бібліотеки мови програмування Python3 для реалізації

Шар	Назва залежності	Версія	Мета
endpoints	fastapi	0.96.1	Реалізація API додатку
	uvicorn	0.22.0	Запуск API на основі fastapi
processing	Стандарта бібліотека Python	-	-
data access	pymongo	4.3.3	MongoDB драйвер для виконання CRUD операцій з базою даних

3.2.3 Тестування розробки

Тестове середовище складається з восьми реплік та для тестування використовувалась наступна процедура:

- створити N записів (документів);
- надіслати K транзакцій на обробку;
- надіслати контрольну транзакцію;

- зберегти часову мітку;
- опитувати кожну репліку до тих пір, поки вона не повертає контрольні значення;
- зберегти часову мітку;
- отримати різницю часових міток за модулем.

Результати наведено на рис. 3.14.

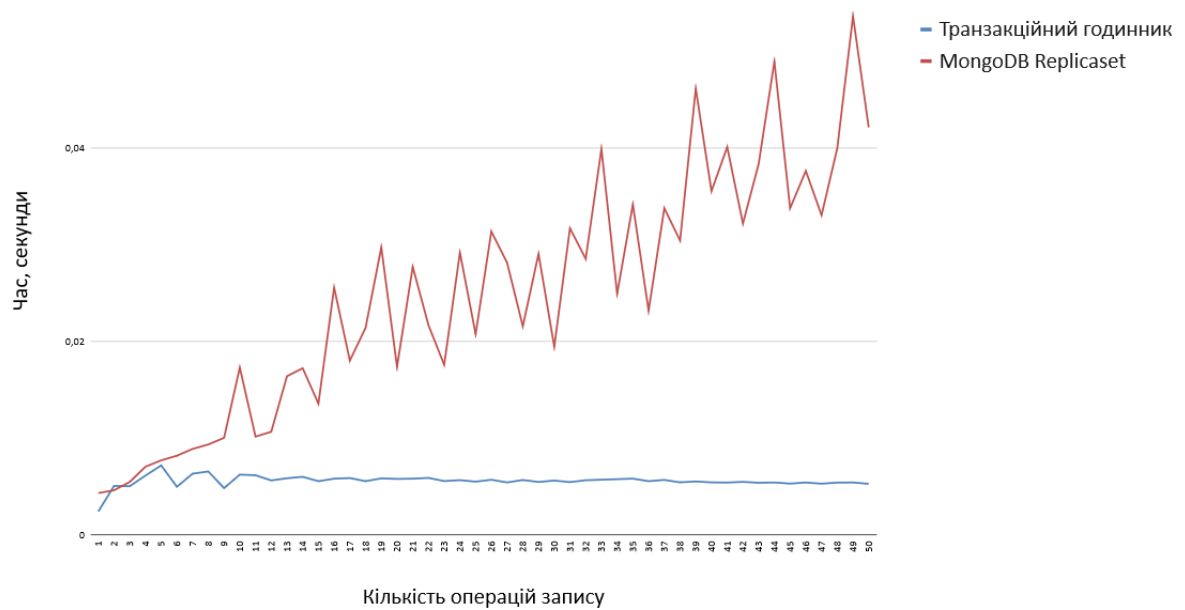


Рисунок 3.14 – Час узгодження даних від кількості оновлень

Також, було додатково проведено експерименти з використанням тієї самої процедури, але розподілена база даних складалась з п'яти та трьох вузлів (рис. 3.15 та рис. 3.16).

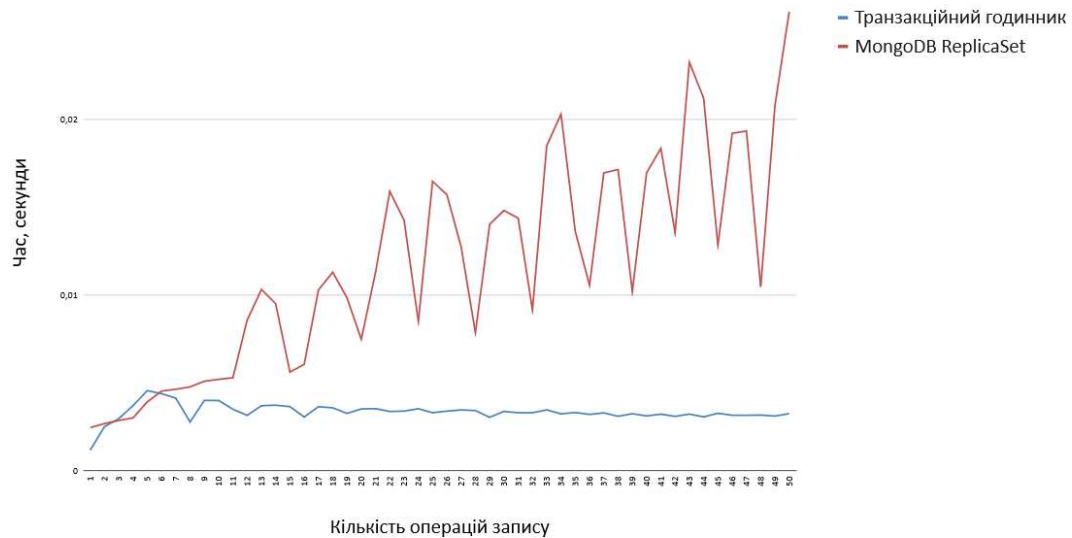


Рисунок 3.15 – Час узгодження даних при п'яти репліках

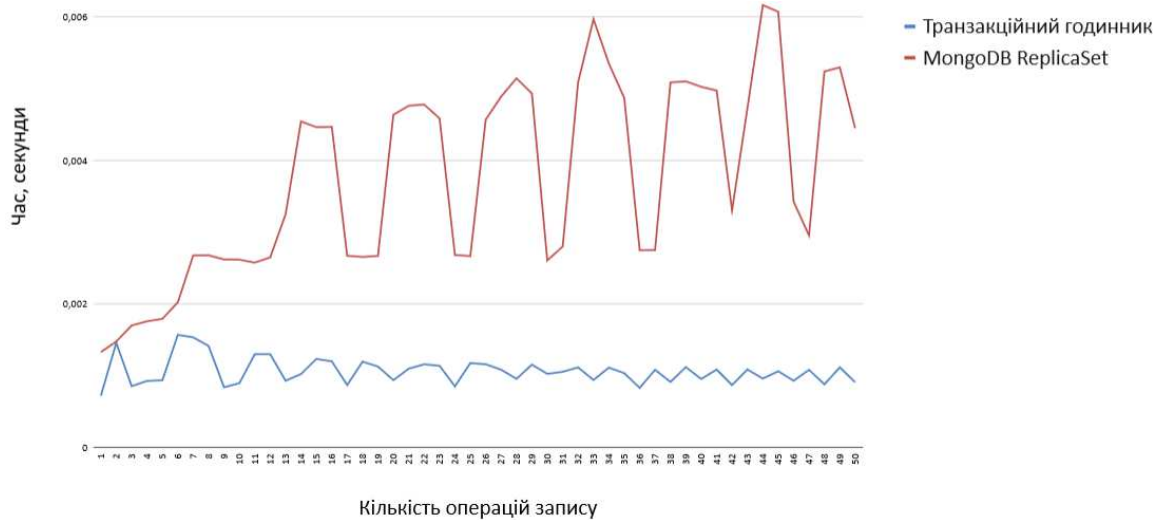


Рисунок 3.16 – Час узгодження даних при трьох репліках

На рис. 3.14, 3.15 та 3.16 видно, що час узгодження даних менше при використанні транзакційного годинника.

Всі транзакції, які надсилалися до транзакційного годинника мали значення 50 пріоритета. Червона лінія представляє собою час необхідний для узгодження даних з використанням вбудованих механізмів MongoDB ReplicaSet, а синя – час узгодження з використанням транзакційного годинника.

Окремо було протестовано пріоритезацію в межах роботи транзакційного годинника. Кроки процедури наступні:

- поставити на паузу опрацювання транзакцій годинником;
- надіслати N транзакцій до годинника;
- надіслати контрольні транзакції з вищим пріоритетом;
- створити часову мітку;
- включити опрацювання транзакцій годинником;
- опитувати кожну репліку до тих пір, поки вона не повертає контрольні значення;
- створити часову мітку;
- отримати різницю часових міток за модулем.

Результати зображено на рис. 3.17.

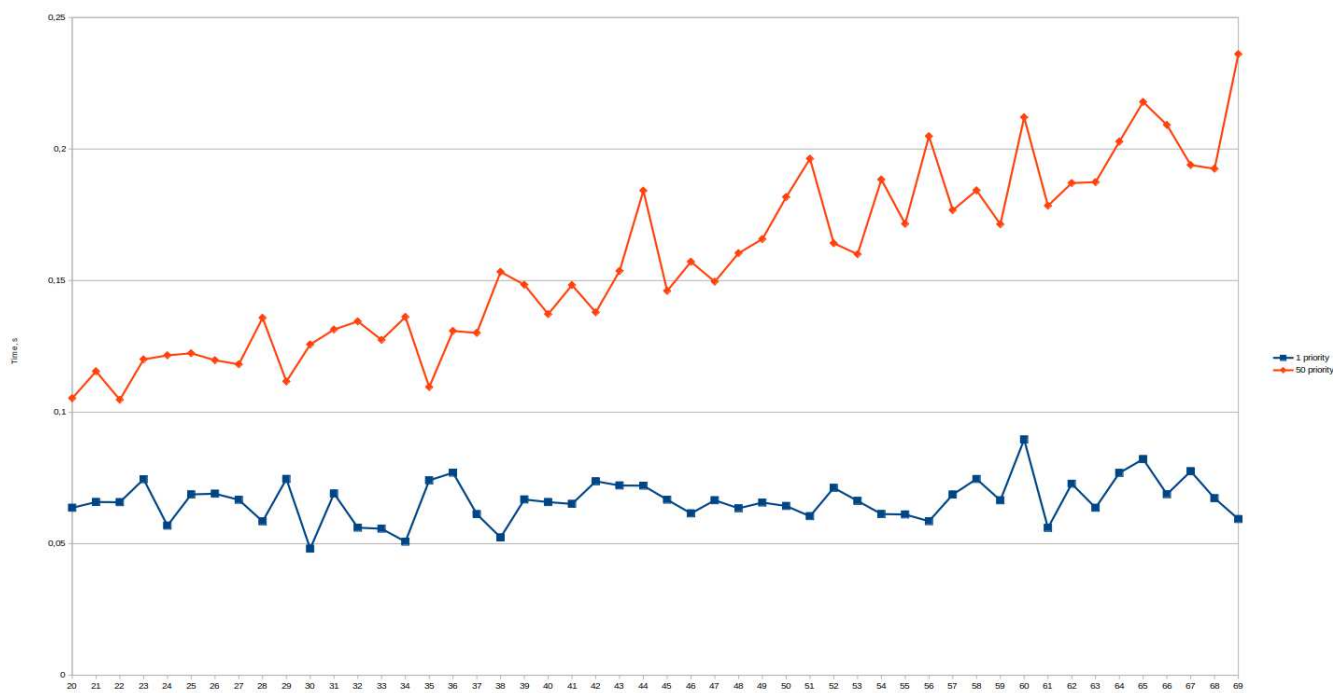


Рисунок 3.17 – Порівняння роботи транзакційного годинника для оновлень з пріоритезацією та без неї

Тобто, час, необхідний для опрацювання транзакцій з меншою пріоритетністю, більший у порівнянні з даними, які опрацьовуються з вищим значенням пріоритета.

3.2.4 Математичний апарат для розрахунку часу узгодження

Процес узгодження даних можна лінеаризувати у вигляді, як зображено на рис. 3.18.

Необхідний час для узгодженості даних залежить від процесора та його характеристик. Необхідно визначити серед основних метрик ті, які характеризують продуктивність процесору та можуть об'єктивно мати вплив на опрацювання транзакцій.

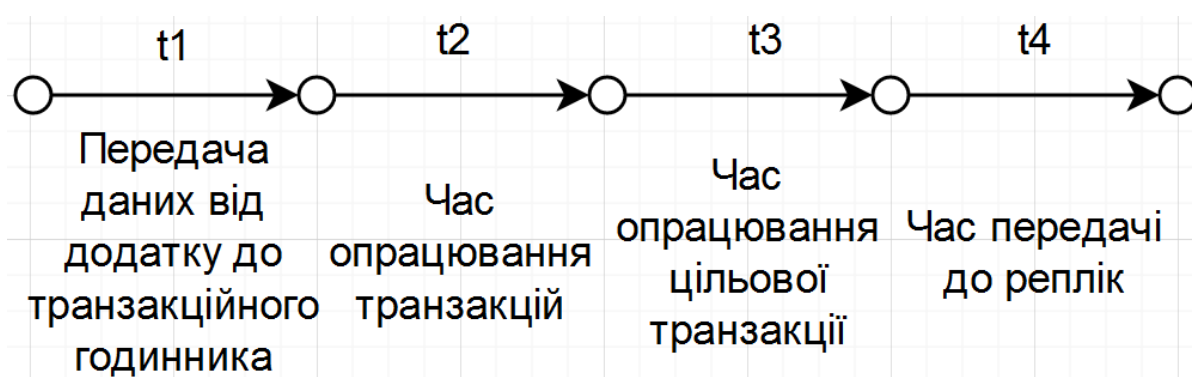


Рисунок 3.18 – Процес узгодження із використанням транзакційного годинника

До основних характеристик, які розповсюджені серед відкритих джерел є:

- Passmark – узагальнююча характеристика, що представляє здатність процесора обробляти різні дії, включаючи арифметичні розрахунки, обробку даних;
- Integer Math – ця характеристика відноситься до здатності процесора виконувати арифметичні дії над цілими числами. Проводяться різні операції додавання, віднімання, множення, ділення, а також порозрядні операції;
- Floating Point – теж саме як і з цілими числами, але з числами з плаваючою точкою;

- Find Prime Numbers – тести на основі алгоритмів пошук простих чисел, які широко використовуються в криптографії;
- Random String Sorting – тести на основі сортування рядків, які розташовані в різному порядку;
- Single Thread – оцінка продуктивності одного потоку. Оцінює здатність процесора виконувати інструкції послідовно.

Вищезазначені характеристики можна отримати емпіричним шляхом при проведенні відповідних тестів.

Для пошуку закономірності між потужністю процесора та роботою транзакційного годинника, необхідно мати експериментальні дані.

Процедуру тесту можна описати наступним чином:

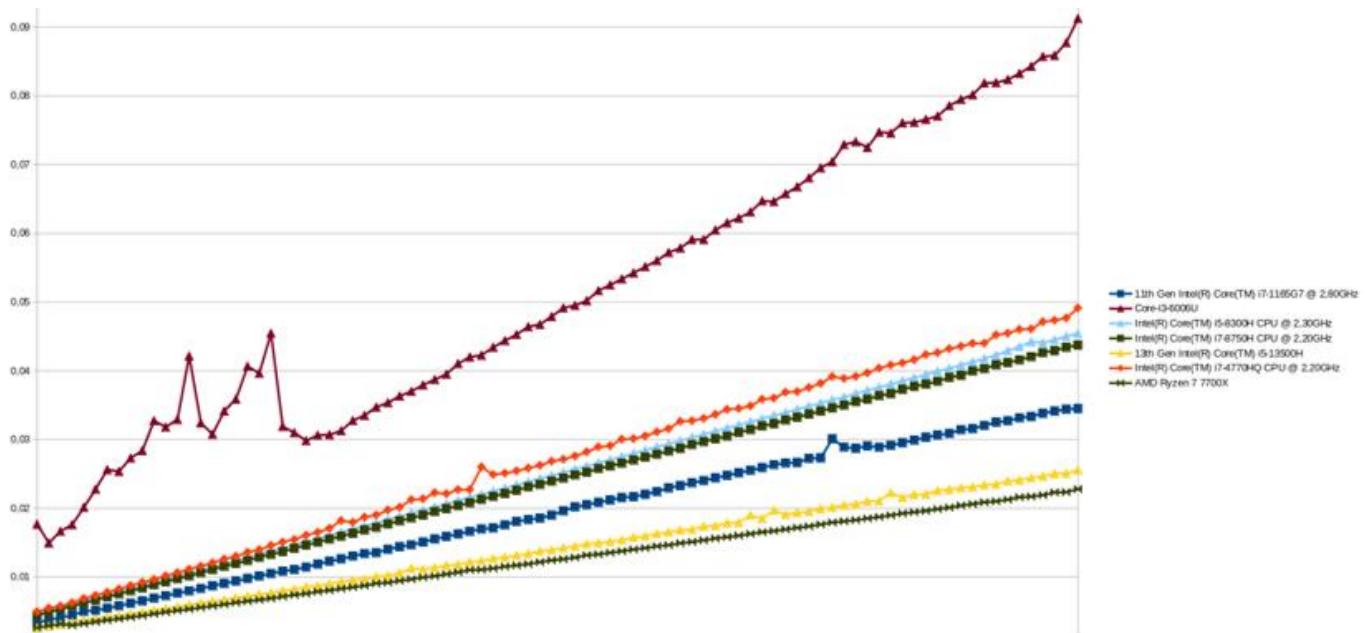
- створити N транзакцій;
- отримати часову мітку;
- опрацювати транзакції та отримати результуючу транзакцію, яка буде надсилатись всім реплікам;
- знову отримати часову мітку;
- отримати різницю між часовими мітками за модулем;
- зберегти отриману різницю, щоб провести статистичний аналіз отриманих даних;
- повторити процедуру K разів.

При проведенні експерименту використовувались наступні вхідні дані:

- $N_{\min} = 5000$;
- $N_{\text{step}} = 500$;
- $N_{\max} = 49\,500$;
- $K = 100$.

Слід зазначати, що нехтувався розмір даних, яких опрацьовувався. В даному контексті це не має суттєвого впливу і буде досліджено у майбутньому.

Отримані результати можна побачити на рис. 3.19.



Рисунки 3.19 – Результати вимірів із використанням різних процесорів

Необхідний час для обробки зростає прямолінійно. Це дає можливість для побудови рівняння прямої, яке описує теоретично можливі значення необхідного часу при використанні того чи іншого процесора.

Рівняння прямої з кутовим коефіцієнтом має наступний вигляд:

$$y = k \cdot x + b, \quad (3.1)$$

де k – кутовий коефіцієнт прямої;

b – стала, яка визначає пряму по вертикалі.

Кутовий коефіцієнт можна розрахувати за формулою:

$$k = \tan a = \frac{y_2 - y_1}{x_2 - x_1}, \quad (3.2)$$

де x_1, x_2 – координати точки по осі абсцис;

y_1, y_2 – координати точки по осі ординат.

Таким чином, залишається розрахувати сталу b . Це можна зробити з використанням побудови системи з двох рівнянь:

$$\begin{cases} y_1 = k \cdot x_1 + b \\ y_2 = k \cdot x_2 + b \end{cases} \quad (3.3)$$

Додамо два рівняння та отримаємо рівняння:

$$y_1 + y_2 = k \cdot x_1 + k \cdot x_2 + 2 \cdot b \quad (3.4)$$

Виразимо змінну b з рівняння:

$$b = \frac{(y_1 + y_2) - (k \cdot x_1 + k \cdot x_2)}{2} \quad (3.5)$$

Використовуючи дані, які були отримані експериментальним шляхом, будемо рівняння та отримуємо графік, як на рис. 3.20.

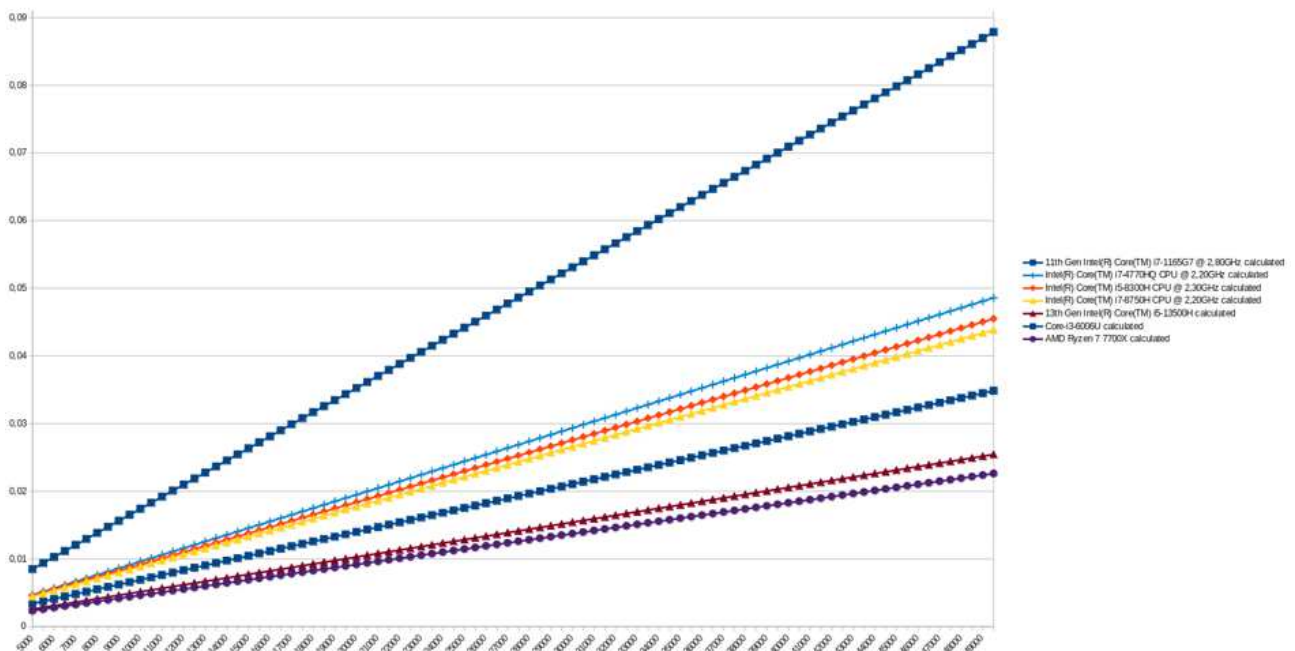


Рисунок 3.20 – Розраховані значення часу

Для визначення того факту, що прямі дійсно розташовуються по порядку збільшення потужності процесора, необхідно визначити відповідну метрику.

Найбільш коректною метрикою при виборі є Single Thread, оскільки при візуальному порівнянні, значення співпадає з пропорційним збільшенням кута нахилу. Значення для різних процесорів зазначені в таблиці 3.4.

Таблиця 3.4 – Значення Single Thread для різних процесорів

Процесор	Core-i3-6006U	Intel(R) Core(TM) i7-4770HQ CPU @ 2,20GHz	Intel(R) Core(TM) i5-8300H CPU @ 2,30GHz	Intel(R) Core(TM) i7-8750H CPU @ 2,20GHz	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2,80GHz	13th Gen Intel(R) Core(TM) i5-13500H	AMD Ryzen 7 7700X
Single Thread, Mops/sec	1,163	1,911	2,281	2,306	2,839	3,571	4,234

Емпіричним шляхом було визначено, що приблизний час обробки можна розрахувати з використанням формули:

$$t_3 = \left(\frac{k \cdot T_{single} \cdot T_c}{P} \right), \quad (3.6)$$

де k – коефіцієнт кореляції;

T_{single} – необхідний час для обробки однієї транзакції, сек;

T_c – кількість транзакцій;

P – потужність процесору, Mops/sec.

Після аналізу даних, які були отримані емпіричним шляхом, було отримано значення $k = 2,2$ та $T_{single} = 0,000000880582$ sec.

Розрахуємо ті самі значення та перевіримо, наскільки модель відхиляється від експериментально отриманих результатів. Отриманий графік можна побачити на рис. 3.21; пунктирною лінією зображено побудовані прямі лінії з використанням формули 3.6.

На рисунку видно, що існує певне відхилення, але оскільки узгодженість даних залежить від багатьох факторів, то даною похибкою можна знехтувати.

Для розрахунку t_2 потрібно використовувати формулу для розрахунку t_3 , але з врахуванням того факту, що існують пріоритети. Тобто опрацювання транзакцій цільового запису відбудеться тоді і тільки тоді, коли будуть опрацьовані більш пріоритетні записи.

$$t_2 = (\sum_{i=1}^P t_3(T_c^i)), \quad (3.7)$$

де p – пріоритет поточного цільового запису;

T_c – кількість транзакцій, які припадають на i -ий пріоритет.

Для розрахунку t_1 потрібно враховувати не тільки потужність хостової машини, але й враховувати особливості передачі даних мережею.

Час, необхідний для надсилання пакета даних, залежить від кількох факторів, таких як розмір пакета, швидкість і ефективність мережевого з'єднання та відстань між відправником і одержувачем.

Припускаючи стабільне та надійне мережеве з'єднання, час, необхідний для надсилання пакета даних, можна розрахувати за такою формулою:

$$L = \frac{D}{S}, \quad (3.8)$$

де D – фізична відстань між відправником та отримувачем, km;

S – швидкість мережевого з'єднання, bps (or Bps).

Наприклад, якщо відстань між відправником і одержувачем становить 1000 кілометрів, а швидкість мережевого з'єднання становить 10 мегабіт на секунду (Мбіт/с), то час, необхідний для відправки пакета даних, можна розрахувати наступним чином:

$$L = \frac{1000 \text{ km}}{10 \text{ Mbps} \cdot 8 \frac{\text{bits}}{\text{byte}}} = 800 \text{ milliseconds}$$

Для більш коректного розрахунку, слід врахувати, що передача даних використовує TCP з'єднання. Встановлення з'єднання вимагає додаткових обчислювальних та мережевих затрат.

Для встановлення нового ТСП-з'єднання потрібне тристороннє рукостискання. Якщо затримка ring між двома серверами становить x мілісекунд, тоді рукостискання призведе до затримки $3x/2$ мілісекунди.

Тоді формула буде виглядати наступним чином:

$$t_1 = L + \frac{(k \cdot 3 \cdot L)}{2}, \quad (3.9)$$

де k – кількість “рукостискань”.

При звичайному ТСП-з'єднанні використовується два “рукостискання”: для створення та закриття з'єднання, тому формулу можна спростити:

$$t_1 = L + 3 \cdot L = 4 \cdot L, \quad (3.10)$$

Для розрахунку часу, який потрібен для передачі даних на репліки, можна використати наступну формулу:

$$t_4 = \sum_{i=1}^m t_1(D_i, S_{TC}), \quad (3.11)$$

де m – кількість реплік,

D – фізична відстань між хостовою машиною годинника та i -ї репліки,

S – швидкість передачі даних хостової машини годинника.

Таким чином, отримавши всі складові, можна скласти остаточне рівняння:

$$t = [4 \cdot L] + \sum_{i=1}^p \left[\frac{(k \cdot T_{single} \cdot T_c^i)}{P} \right] + \sum_{i=1}^m t_1(D_i, S_{TC}) \quad (3.12)$$

3.3 Програмна реалізація методу активної антиентропії

3.3.1 Проєктування та програмна реалізація

Для максимального спрощення та зменшення часу на реалізацію було вирішено використовувати існуюче програмне забезпечення бази даних MongoDB для виконання всіх CRUD операцій..

Тобто, розробляємий сервіс буде представляти собою спеціальну надбудову, яка буде приймати всі CRUD операції та зберігати такі метадані, як часова мітка останньої

операції запису та реалізовувати активну антиентропію. Загальна схема роботи зображена на рис. 3.23.

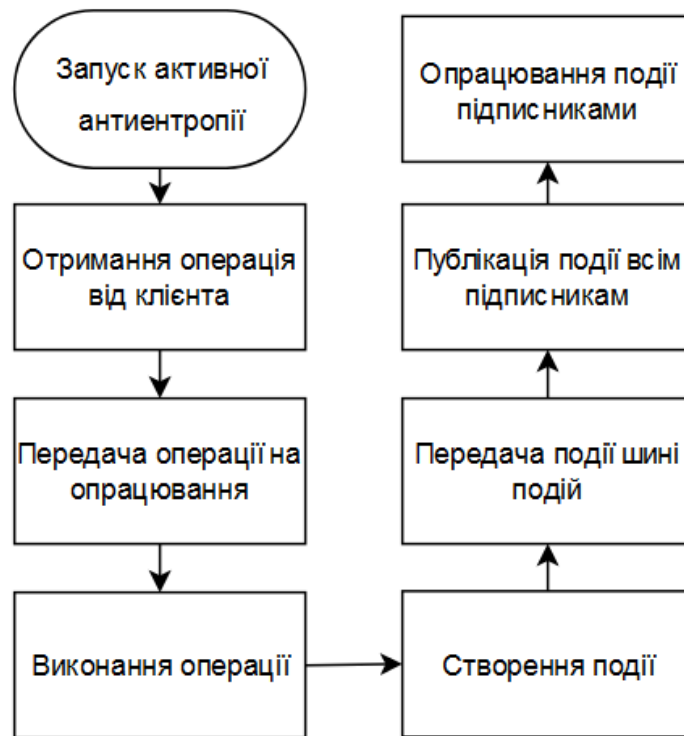


Рисунок 3.23 – Загальна схема роботи сервісу активної антиентропії

Програмна реалізація представляє собою три пов'язані між собою шари: шар сервісу для комунікації, шар з сервісами CRUD операцій та шар з плагінами.

Операції надходять через шар сервісу для комунікації, який працює за строго визначеними структурами повідомлень. Для цього шар використовує порт, на якому чекає підключень клієнтів по TCP протоколу. Клієнти передають повідомлення із визначеною структурою (рис. 3.24 – 3.26).

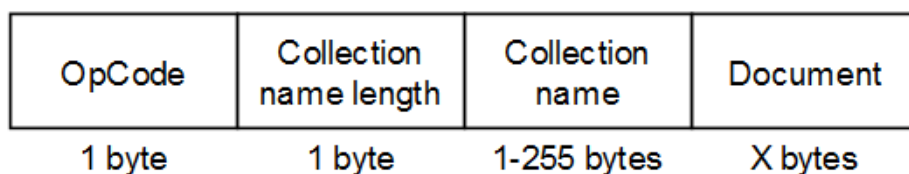


Рисунок 3.24 – Структура мережевого сегменту для створення документу

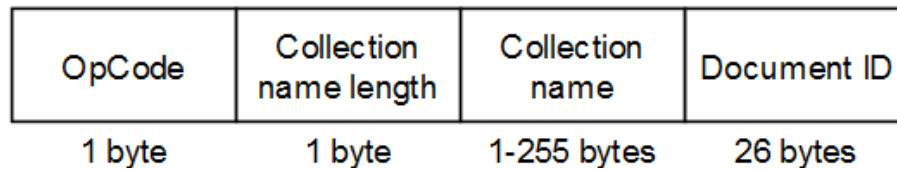


Рисунок 3.25 – Структура мережевого сегменту для читання документу

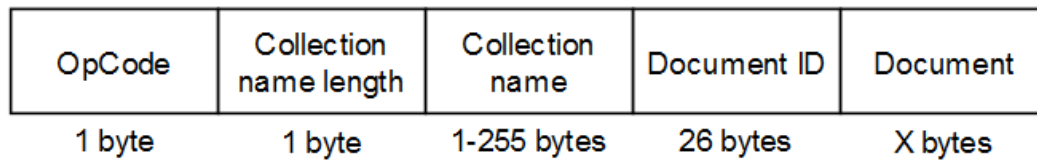


Рисунок 3.26 – Структура мережевого сегменту для оновлення документу

Опис полів структури повідомлень:

- OpCode – представляє собою номер, який використовується стороною отримувача для того, щоб розрізнити яка саме операція виконується. Якщо передається значення 1, то це операція створення документу і отримувач десеріалізує це повідомлення за відповідним шаблоном. Якщо передається значення 2, то це операція оновлення. Якщо передається 3, то це операція видалення. І останній варіант це значення 4, при якому відбувається читання документа з бази даних;
- Collection name length – представляє собою число і символізує кількість байтів після цього поля, які відносяться до ім'я колекції. Тобто, скільки байтів потрібно зчитати після, щоб отримати ім'я колекції;
- DocumentID – унікальний ідентифікатор документа;
- Document – байти, які представляють собою документ у форматі JSON. Це і є те, що записується в базу даних.

Відповідні частини отримуються із вихідного повідомлення, відбувається десеріалізація та передається до шару сервісу CRUD операцій.

Шар сервісу представляє собою інтерфейси, які дозволяють виконувати операції записів, читання та видалення. Всі операції надходять до спеціального методу, який

розподіляє їх по різних чергам. Кожна черга представляє собою структуру даних, яка слідує принципам FIFO, для забезпечення почергової обробки, але глобально операції виконуються у порядку, який забезпечує максимальну швидкість відгуку: спочатку виконуються операції читання, а потім запису.

Окрім операцій запису та читання документів, цей шар відповідальний за розрахунок знімків, які будуть використовуватись в процесі активної антиентропії. Це дозволяє одночасно з CRUD операціями підтримувати актуальний кеш, який представляє собою асоціативний контейнер, ключами якого виступають ідентифікатори документів, а значеннями – пара, яка складається з розрахованих спектрального фільтру Блума та хеш-значення RH2-48 алгоритму.

Окрім передачі документів в базу даних, цей шар оперує службовими метаданими, які також можуть використовуватись для активної антиентропії. Наприклад, час коли було виконано останній запис. Це значення має одну з ключових ролей під час комунікації сусідніх вузлів методу активної антиентропії.

Після виконання якоїсь операції створюється відповідна подія, яка потрапляє в шину подій. Її завдання полягає у тому, щоб передати подію до підписників, які підписані на певні типи подій. Наприклад, один плагін може очікувати події, які виникли внаслідок операцій записів, а інший на події, які виникли в результаті читання певних даних.

На рис. 3.27 зображена UML діаграма найголовніших складових.

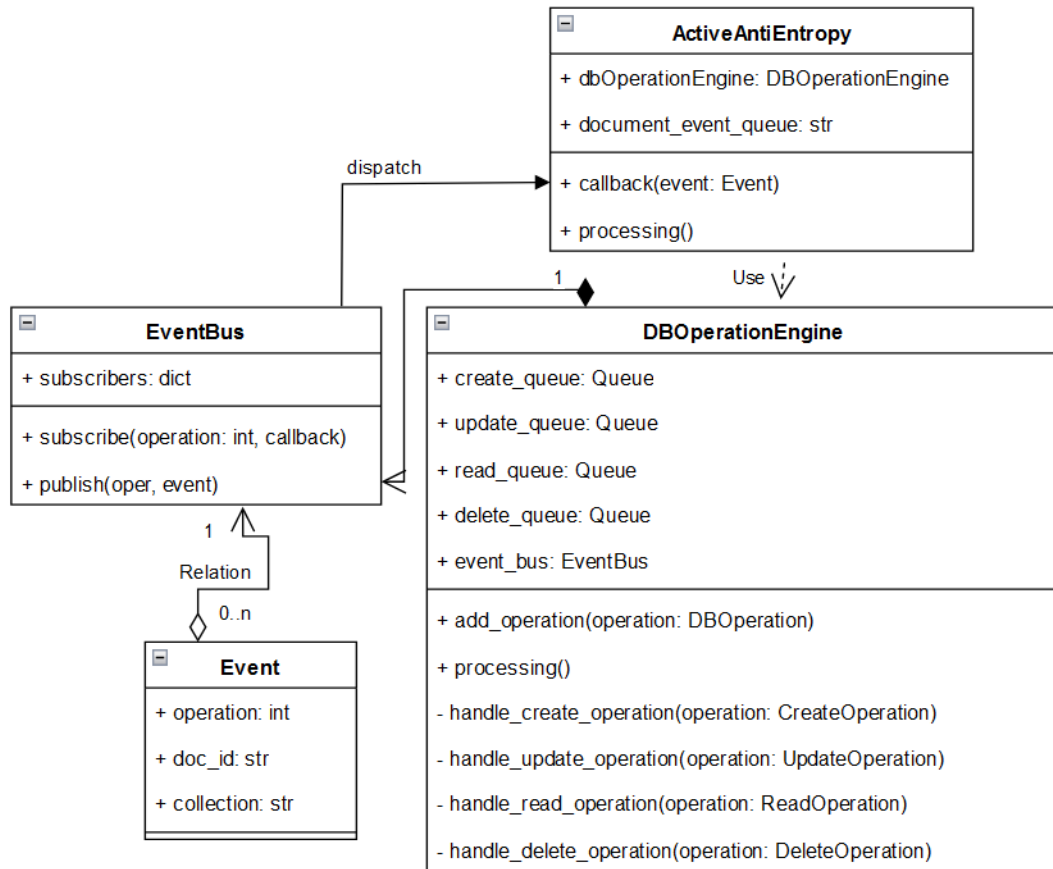


Рисунок 3.27 – UML діаграма головних складових шару CRUD операцій

Опис класу DBOperationEngine:

- поля `create_queue`, `update_queue`, `read_queue`, `delete_queue` – представляють собою черги для зберігання CRUD операцій, які працюють за принципом FIFO. Наприклад, `create_queue` зберігає операції, які пов’язані зі створенням документів;
- поле `event_bus` – представляє собою об’єкт, до якого надсилаються події, які пов’язані із CRUD операціями;
- метод `add_operation` – використовується в шарі сервісу для комунікації при отриманні операції через мережу;
- `processing` – представляє собою метод, який нескінченно проходить по чергам з CRUD операціями та опрацьовує операції. Необхідний для запуску в окремому потоці на виконання при багатопоточності;

- методи `handle_create_operation`, `handle_update_operation`, `handle_read_operation`, `handle_delete_operation` – опрацьовують відповідну операцію, яка прийшла. Відповідальні за роботу безпосередньо з екземпляром бази даних.

Опис класу `EventBus`:

- поле `subscribers` – представляє собою асоціативний контейнер, який зберігає функції зворотнього зв'язку підписників за кодами операцій;
- метод `subscribe` – зберігає функцію зворотнього зв'язку в контейнер `subscribers` за кодом операції;
- метод `publish` – отримує події та передає їх відповідним підписникам.

Опис класу `ActiveAntiEntropy`:

- поле `dbOperationEngine` – об'єкту класу `DbOperationEngine`, який використовується для операцій запису при отриманні документу при роботі механізму активної антиентропії;
- поле `document_event_queue` – представляє собою чергу подій, які надійшли від шини подій. Працює за принципом FIFO;
- метод `callback` – реалізує метод інтерфейсу `Subscriber` та опрацьовує події, які надходять від шини подій;
- метод `processing` – неперервно опрацьовує чергу подій. Якщо черга порожня, то по черзі дістає знімки документів з відповідного кешу та спілкується із сусідніми вузлами з метою узгодження даних.

Шар плагінів має плагін, який реалізує метод активної антиентропії. Для цього він використовує два порти, які працюють на транспортному рівні моделі OSI. Один із них працює з використанням протоколу TCP та потрібен для отримання документів. Другий працює з використанням протоколу UDP та використовується для того, щоб отримувати повідомлення зі знімками даних. Знімок даних має чітко визначену структуру та використовується для ідентифікації неузгодженості певного документу.

Якщо неузгодженість була виявлена, то отримується часова мітка з метаданих та надсилається у відповідь. Інший вузол, отримавши часову мітку, отримує локальну часову мітку і в результаті порівняння, якщо локальна часова мітка більш актуальна, то відбувається надсилання документу іншому вузлу. У іншому випадку, передача документу не відбувається. Коли документ надійшов до сервісу активної антиентропії на іншому вузлі, то він передає його сервісу CRUD операцій.

3.3.2 Реалізація та тестування алгоритму хешування PH2-48

Для реалізації алгоритму хешування було створено спеціальний клас, який реалізує необхідні функції для додавання байтів та хешування. Одним із ключових моментів є те, що кількість байтів, які підлягають хешуванню, повинна бути такою самою, як і розмір блоку. Отриманий набір бітів конвертується в число з використанням порядку байтів “big endian” і виконується перевірка чи є це число простим чи ні. Після цього відбувається зміна тих чи інших складових хеш-значення. Приклад роботи алгоритму хешування зображено на рис. 3.28.

```
hash1 = PH2()
hash1.append('Hello, World!'.encode('utf-8'))

hash2 = PH2()
hash2.append('World, Hello!!'.encode('utf-8'))

res1 = hash1.digest()
res2 = hash2.digest()

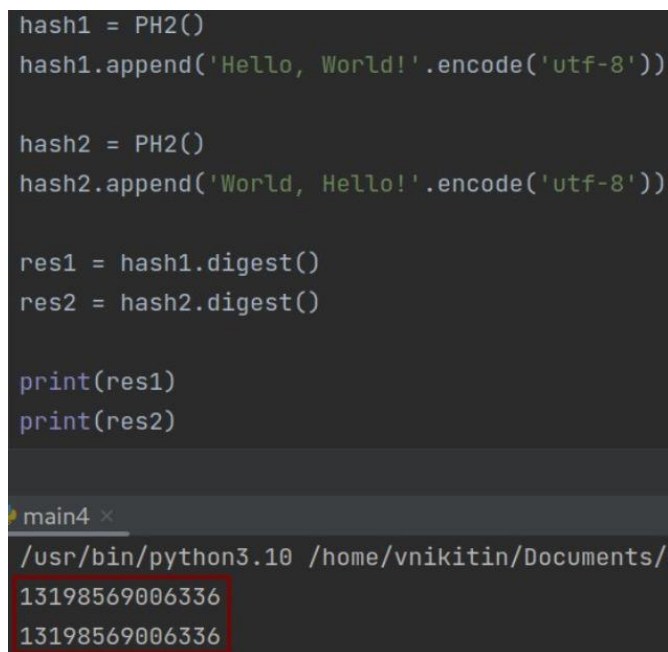
print(res1)
print(res2)
```

main4 ×

```
/usr/bin/python3.10 /home/vnikitin/Documents/K
13198569006336
14298634282240
```

Рисунок 3.28 – Приклад використання реалізації PH2

Як було зазначено при розробці методу хешування, колізії можуть траплятись при хешування даних з однакового набору байтів, але розташованих в різних порядках (рис. 3.29).



```
hash1 = PH2()
hash1.append('Hello, World!'.encode('utf-8'))

hash2 = PH2()
hash2.append('World, Hello!'.encode('utf-8'))

res1 = hash1.digest()
res2 = hash2.digest()

print(res1)
print(res2)
```

main4 x

/usr/bin/python3.10 /home/vnikitin/Documents/

13198569006336

13198569006336

Рисунок 3.29 – Приклад колізії при хешуванні PH2-48

Також було додано засіб захисту від зміни, який полягає у генерування помилки, якщо даний об'єкт було заблоковано на модифікацію. Зняти це обмеження неможливо.

Процедура для оцінки кількості колізій при різних розмірах в даному випадку представляє наступне:

- згенерувати N унікальних вхідних даних;
- отримати N хешів використовуючи згенеровані дані;
- розподілити вхідні дані по отриманим хешам;
- отримавши асоціативний контейнер можна побачити, які вхідні дані були використані для отримання відповідного хешу;
- отримати довжини вхідних даних, які відносяться до певного хеш-значення;

- збільшити значення лічильника на одиницю, якщо є як мінімум двоє вхідних даних, які відрізняються за розміром.

Процедура була виконана при генеруванні даних від 1000000 до 3900000 випадково згенерованих вхідних даних. Вхідні дані склалися зі знаків пунктуації, цифр, ASCII символів та пробілів. Результати наведено на рис. 3.30.

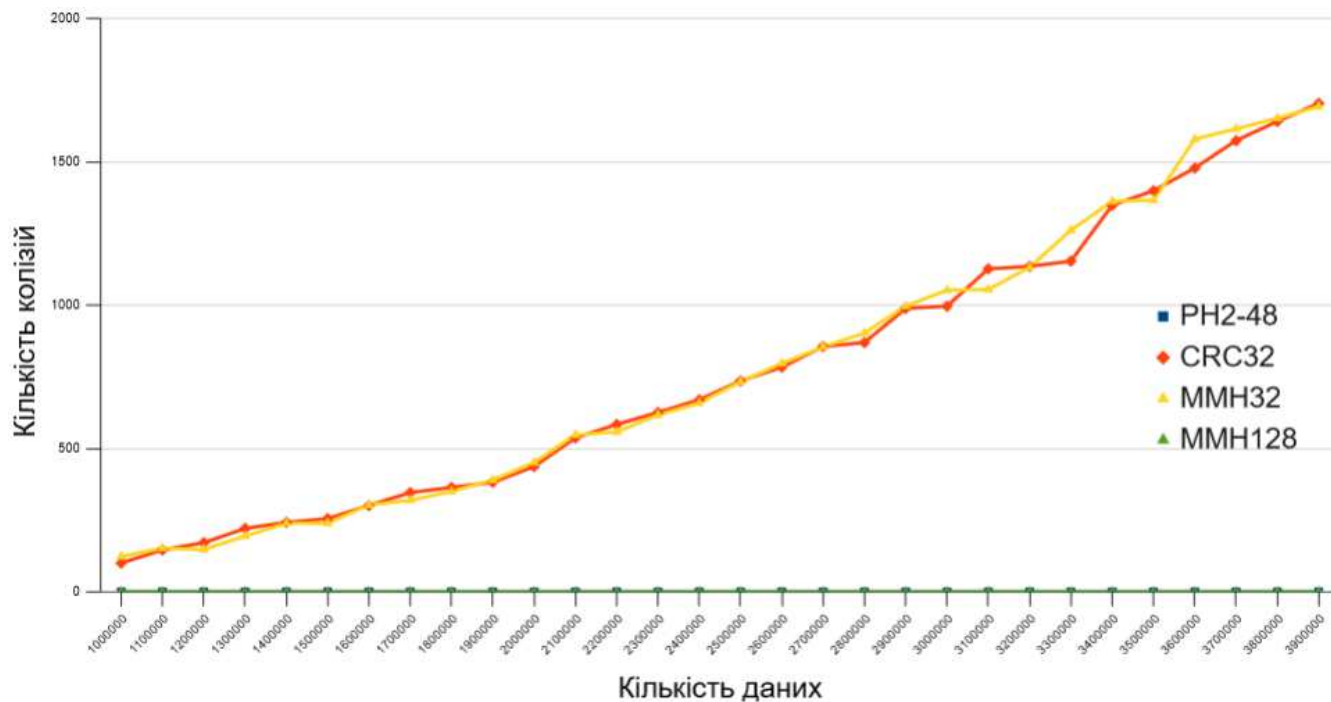


Рисунок 3.30 – Кількість колізій при хешуванні різних за розмірами даних

Колізійно стійкими в таких умовах є PH2-48 та Murmurhash3-128. Кількість колізій при їх використанні дорівнює 0, на відміну від CRC32 та Murmurhash3-32.

Для оцінки колізійної стійкості можна використати наступну процедуру:

- згенерувати N унікальних вхідних даних;
- отримати N хешів використовуючи згенеровані дані;
- розподілити вхідні дані по отриманим хешам;
- отримавши асоціативний контейнер (відношення?) можна побачити, які вхідні дані були використані для отримання відповідного хешу;

- збільшити значення лічильника на одиницю, якщо кількість вхідних даних, які асоціюються з відповідним хеш-значенням, більша за 2.

Процедура була виконана при генеруванні даних від 1000000 до 3900000 випадково згенерованих вхідних даних. Вхідні дані склалися зі знаків пунктуації, цифр, ASCII символів та пробілів. Результати наведено на рис. 3.31.

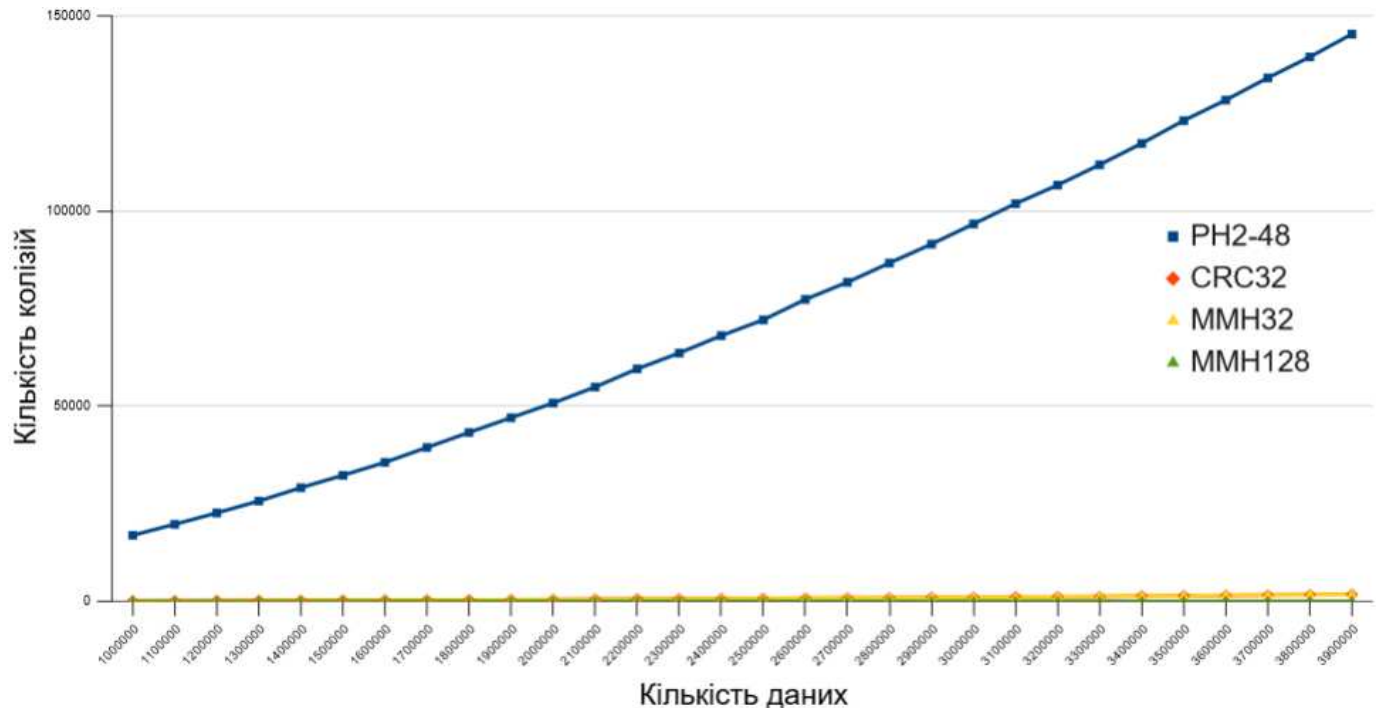


Рисунок 3.31 – Кількість колізій

На рис. 3.31 можна побачити, що алгоритм PH2-48 сильно програє в колізійній стійкості і отримана кількість колізій значно вища, ніж у інших алгоритмів.

Для оцінки швидкодії алгоритму можна використати наступну процедуру:

- згенерувати N байтів, які будуть вхідними даними для хешування;
- зробити відмітку часу;
- отримати хеш;
- зробити відмітку часу;
- отримати різницю між відмітками за модулем;

- кроки 3-5 зробити K разів;
- отримати медіану використовуючи набір із K різниць.

Процедура була виконана при генеруванні випадкових даних розміром від 1400 до 1500 байтів. Кількість ітерацій для знаходження медіани дорівнює 100. Результати наведено на рис. 3.32.

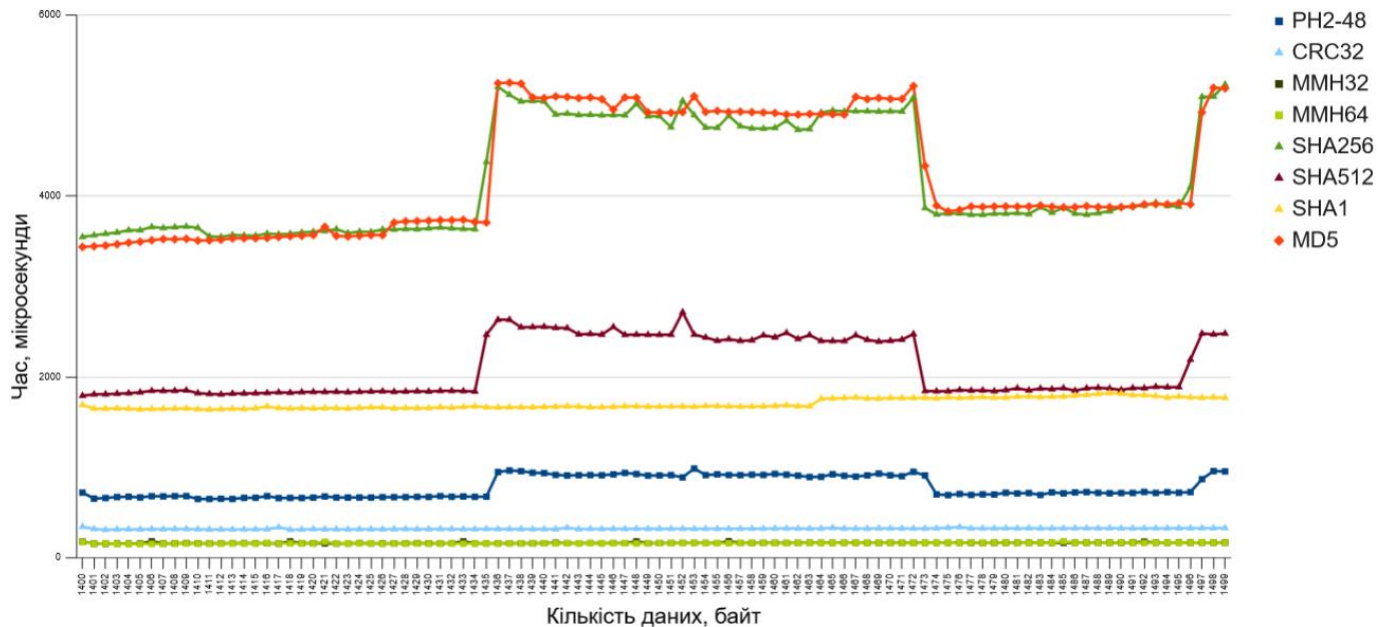


Рисунок 3.32 – Швидкість формування хеш-значень

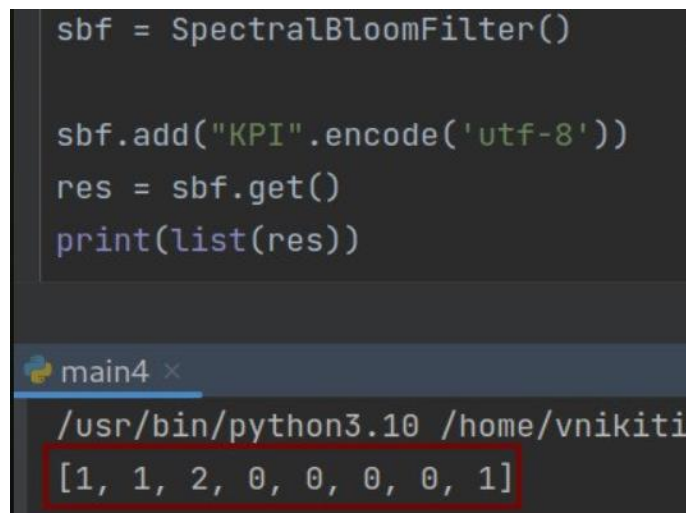
На рис. 3.32 видно, що PH2-48 поступається в швидкості формування хеш-значення в порівнянні з усіма некриптографічними алгоритмами, які використовувались в експерименті та швидше криптографічних.

3.3.3 Реалізація та тестування модифікованого спектрального фільтру Блума

Для реалізації модифікованого спектрального фільтру Блума було створено спеціальний клас, який має функції для додавання байтів до фільтру. У конструкторі створюється список, який зберігатиме лічильники. Коли додається байт, то відбувається ітерування по набору з простими числами і ділення цілочисельної репрезентації цього байту на кожне просте число. Якщо число не було поділено ні на

що, то збільшується на одиницю останній елементу списку. Кількість елементів фільтру дорівнює 8.

Для уникнення помилок, є можливість заблокувати зміни об'єкту цього класу. Приклад формування спектрального фільтру Блума зображена на рис. 3.33.



```

sbf = SpectralBloomFilter()

sbf.add("KPI".encode('utf-8'))
res = sbf.get()
print(list(res))

```

main4 ×

```

/usr/bin/python3.10 /home/vnikiti
[1, 1, 2, 0, 0, 0, 0, 1]

```

Рисунок 3.33 – Формування модифікованого спектрального фільтру Блума

Процедура експерименту полягає у тому, щоб згенерувати випадковим чином дані у певному форматі та отримати фільтри. Залежними змінними в даному випадку виступають кількість колізій та час, необхідний для генерування фільтру, а незалежною – кількість даних. Приклад даних зображено на рис. 3.34.

```

data = {
    '_id': str(ObjectId()),
    '_modified_at': datetime.datetime.utcnow().strftime(UTC_FORMAT),
    'firstname': fake.first_name(),
    'lastname': fake.last_name(),
}

```

Рисунок 3.34 – Приклад структури тестових даних

Дані представляють цілком можливі дані, які можуть зберігатися в документоорієнтованій базі даних.

Кількість колізій при формуванні спектральних фільтрів Блума з використанням модифікованого алгоритму та на основі хеш-функцій зображено на рис. 3.35.

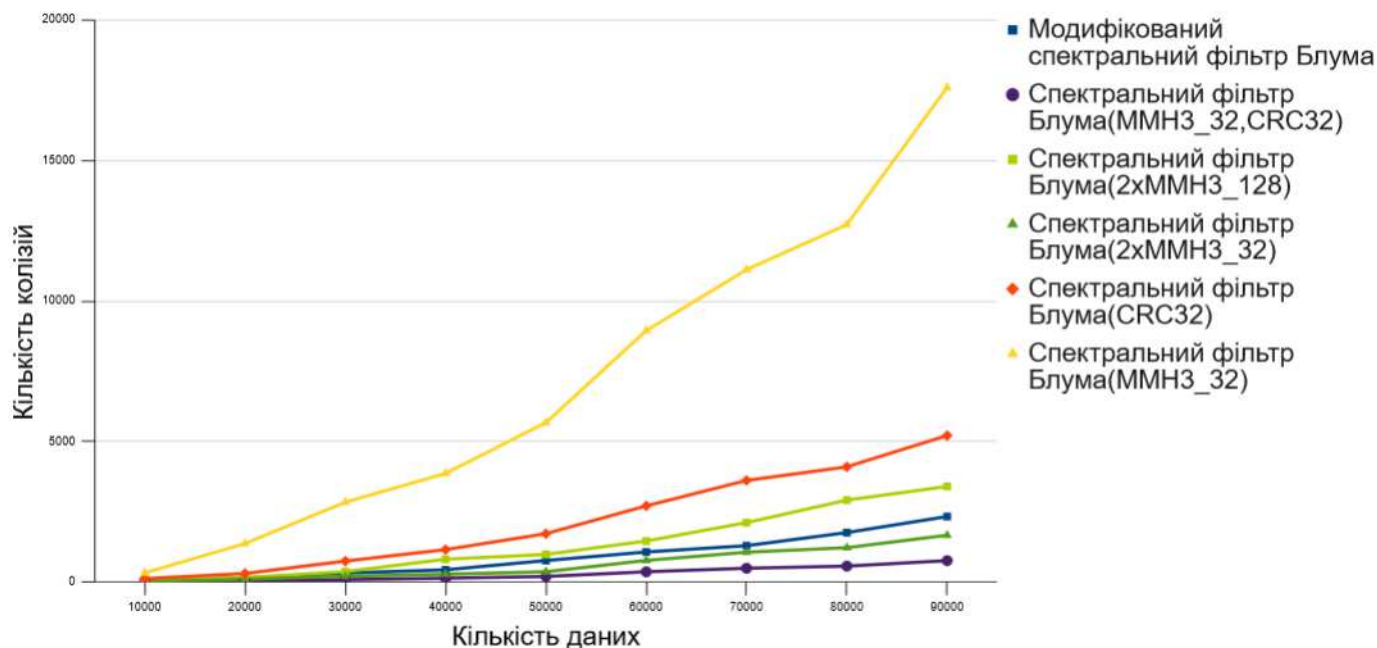


Рисунок 3.35 – Кількість колізій в залежності від кількості даних

З наведених результатів видно, що модифікований алгоритм на основі простих чисел краще працює, ніж алгоритм формування з одиночною хеш-функцією. Якщо використовуються більше ніж одна, то кількість колізій зменшується.

Слід зазначити, що використання більшої кількості хеш-функцій збільшує час, який необхідно витратити на формування фільтру. Результати замірів можна побачити на рис. 3.36.

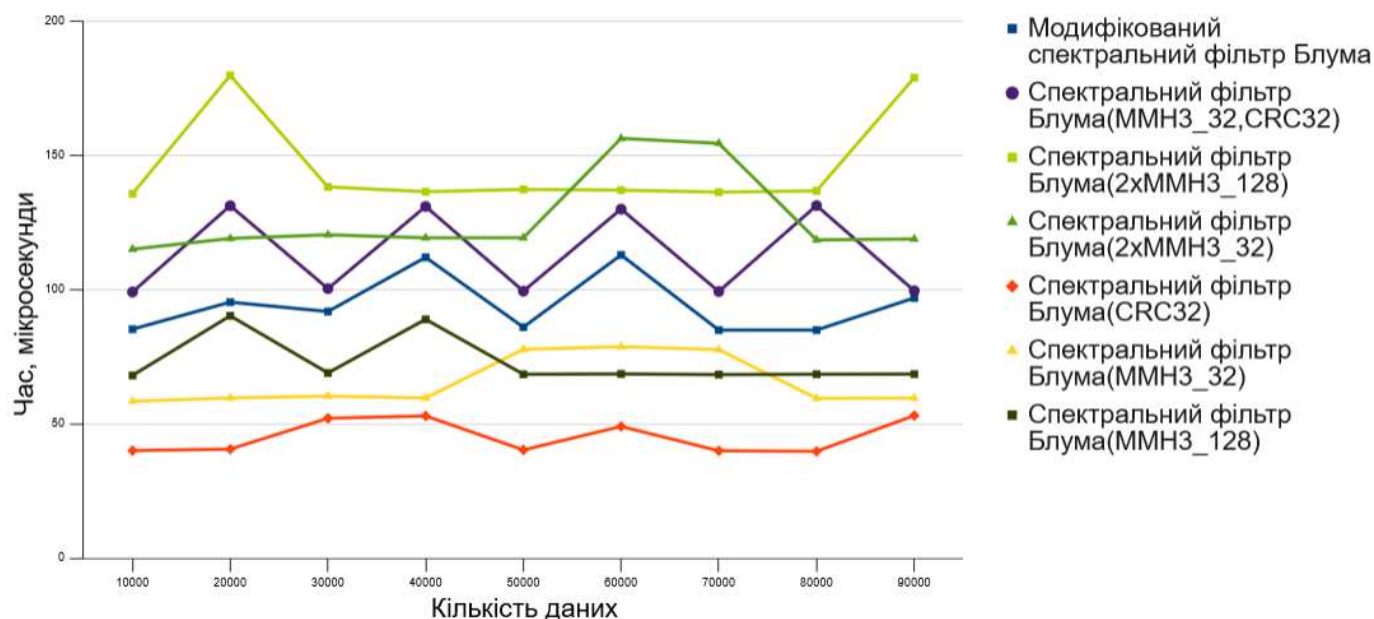


Рисунок 3.36 – Порівняння швидкості формування спектрального фільтра Блума

Таким чином, використання алгоритму формування спектрального фільтра Блума на основі простих чисел можна використовувати як аналог до наборів, які складаються з двох хеш-функцій алгоритму хешування Murmurhash3. Це дозволяє зменшити кількість колізійних ситуацій та пришвидшити формування фільтра.

3.4 Експериментальні досліді із застосуванням розробленої інформаційної системи

Узгодженість інформаційної системи можна вважати синонімом узгодженості розподіленої бази даних, оскільки будь-яка інформація, якою оперують користувачі, зберігається саме там. Від узгодженості бази даних залежить наскільки швидко користувачі, які прив'язані до іншого вузла, побачать зміни.

Для тестування та порівняння ефективності впроваджених методів було розроблено спеціальний модуль з реалізацією API, який дозволяє взаємодіяти із інформаційною системою та підсистемою зберігання, обробки та реплікації даних.

Модуль написаний із використанням мови програмування пайтон, використовує як вже існуючу бібліотеку для виконання запитів, сторонній драйвер pymongo для

роботи з MongoDB екземплярами, так і написані власноруч драйвери для створеного програмного забезпечення.

Загалом, процедура для тестування розробленої інформаційної системи складається з наступних кроків:

- зареєструвати певну кількість банкоматів;
- зареєструвати певну кількість відділень фінансової установи;
- зареєструвати певну кількість облікових записів та кредитних карток;
- зробити імітацію надсилання певної кількості транзакцій кожним користувачем з фіксуванням значення кінцевого значення балансів на картках;
- зробити часову мітку;
- робити запити до кожного вузла розподіленої бази даних та очікувати, коли всі відповіді повернули зафіксоване кінцеве значення балансу на кожній картці;
- зробити часову мітку;
- різниця за модулем між часовими мітками і буде часом узгодження.

Порівняння транзакційного годинника та вбудованого механізму узгодження даних MongoDB зображено на рис. 3.37.

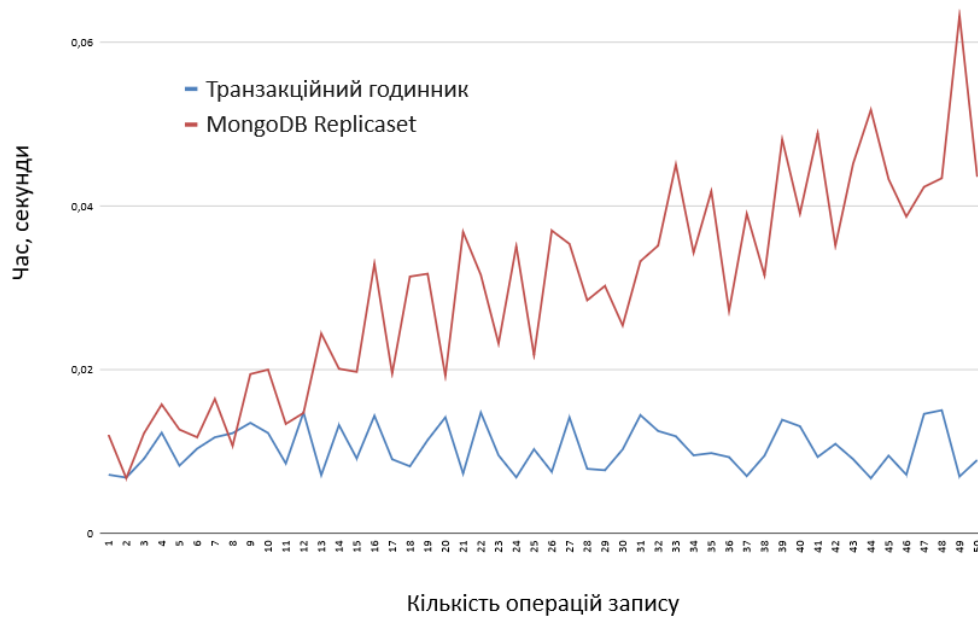


Рисунок 3.37 – Порівняння транзакційного годинника та MongoDB ReplicaSet

На рис. 3.37 червона лінія представляє собою роботу інформаційної системи із використанням вбудованих механізмів забезпечення узгодженості в MongoDB ReplicaSet. Зі збільшенням кількості оновлень за одну ітерацію час для узгодження даних збільшується і пересікає границю в 0.06 секунди.

Синя лінія представляє собою час узгодженості даних при використанні транзакційного годинника. Час затрачений для узгодженості системи менше і не збільшується в часі. Це пояснюється тим, що транзакції об'єднуються в результуючу і тільки вона відправляється реплікам. Вбудований механізм MongoDB ReplicaSet робить записи по-одному, що вимагає додаткового часу при передачі мережею та записами на жорсткий диск.

Порівняння розробленого механізму активної ентропії та MongoDB ReplicaSet зображено на рис. 3.38.

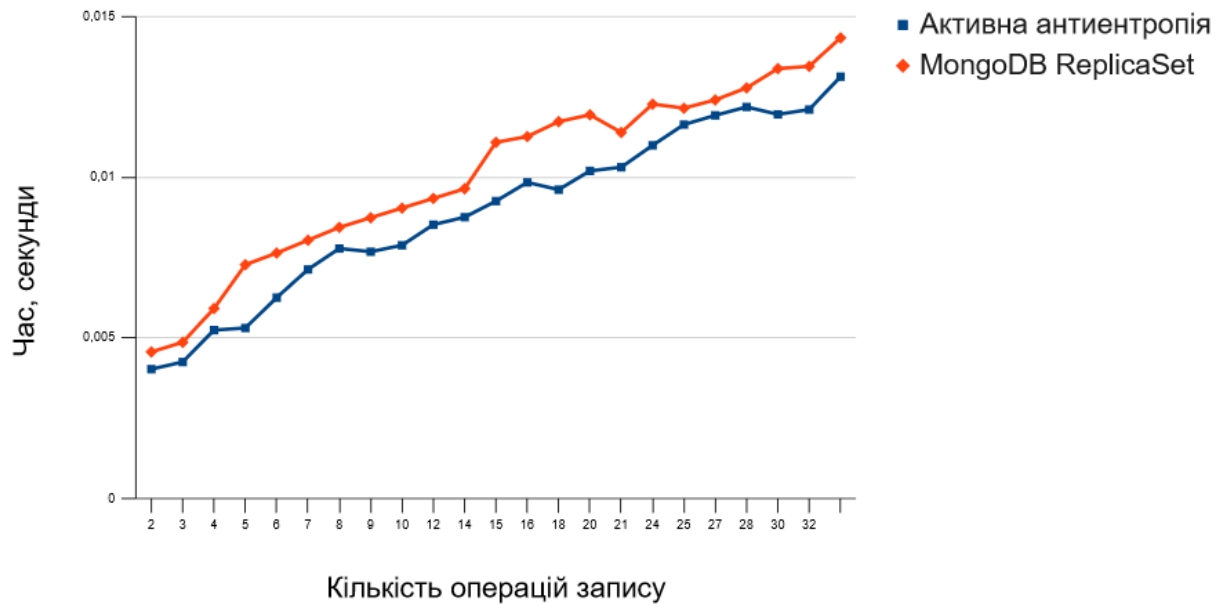


Рисунок 3.38 – Порівняння механізму активної антиентропії та MongoDB ReplicaSet

З графіку видно, що механізм активної антиентропії працює швидше, ніж механізми MongoDB ReplicaSet, але також відбувається поступове зростання необхідного часу. Це пояснюється тим, що при роботі процедури ідентифікації неузгодженості накопичуються транзакції та відбуваються додаткові затрати на комунікацію із сусідніми вузлами.

Для оцінки приблизного покращення, було вирішено використати наступну процедуру:

- розрахувати середнє арифметичне значення з набору отриманих даних при використанні вбудованого механізму MongoDB ReplicaSet;
- розрахувати середнє арифметичне значення з набору отриманих даних при використанні одного з розроблених методів;
- знайти різницю за модулем між розрахованими значеннями;
- розрахувати відсоток, який складає ця різниця від середнього арифметичного значення MongoDB ReplicaSet.

Використовуючи вищезазначену процедуру, кожен метод дозволив пришвидшити процес узгодження даних:

- використання активної антиентропії ефективно себе показало на ділянках з відносно невеликою кількістю оновлень і дозволило збільшити швидкість узгодження на 11 %;
- використання транзакційного годинника показало себе ефективно на ділянці з більшою кількістю оновлень і дозволило збільшити швидкість узгодження критично важливих даних на 77 %.

3.5 Розробка рекомендацій по впровадженню запропонованих методів

Для впровадження методів в існуючі інформаційні системи, потрібно в першу чергу мати глибоке розуміння самої системи, взаємодії її підсистем та особливостей використання, оскільки впровадження тих чи інших методів може вплинути на інші властивості системи.

Всі методи можуть бути впроваджені в інформаційні системи через програмне забезпечення нереляційних документо-орієнтованих баз даних або представляти собою спеціалізовані надбудови. Наприклад, транзакційний годинник з пріоритезацією може представляти собою окремий сервіс із власним інтерфейсом для взаємодії, але слід пам'ятати, що це може вимагати додаткових витрат.

Розглянемо в якості прикладу інформаційну систему, до складу якої входить MongoDB ReplicaSet. Можна впровадити розроблені методи наступними шляхами:

- використання відповідного програмного забезпечення MongoDB із впровадженням транзакційним годинником Цей варіант найбільш оптимальний з точки зору продуктивності та стандартизації рішення. Також, дозволяє використовувати розроблене рішення для різних проектів;
- розробка окремого сервісу із реалізацією універсального рішення задання пріоритезації. Продуктивність такого рішення буде гірша у порівняння з попереднім варіантом, але у разі потреби це є одним із можливих рішень. Також, недоліком є те, що у вже працюючій інформаційній системі може не

бути пріоритезації та обов'язкової передачі часової мітки створення транзакції. В цьому випадку, потрібно модифікувати існуюче рішення під розроблений сервіс;

- розробка окремого сервісу із реалізацією під певну конкретну інформаційну систему. При цьому варіанті необхідність в модифікуванні працюючої системи мінімальна. Єдиною потребою в цьому випадку є переналаштування на відправку запитів до сервісу. Всі пріоритети можуть бути налаштовані в самому сервісі для певної колекції або за якимись унікальними критеріями. З точки зору узгодженості даних недоліком є те, що може бути проблема впорядкування транзакцій з точки зору логіки, оскільки часові мітки будуть створюватися не в момент відправки певним додатком, а по факту надходження в сервіс. Також, недоліком є те, що його неможливо використовувати як готове рішення для інших проєктів.

Слід зазначати, що при розробці власного сервісу рекомендується мати декілька вузлів та впровадити якийсь механізм, який дозволить переключатися з одного сервісу на інший у разі потреби. Якщо цього не зробити, то можлива ситуація, коли всі операції записів не будуть потрапляти до бази даних. Для реалізації такого механізму можна звернути увагу на такі протоколи як Paxos та Raft.

При реалізації транзакційного годинника рекомендується використовувати протокол транспортного рівня TCP для реалізації інтерфейсу комунікації. Це дозволить зменшити час на декапсуляцію даних мережевим стеком.

Метод активної антиентропії також може бути впроваджено в працюючу інформаційну систему. Для цього потрібно розробити спеціальний сервіс, який буде прив'язано до певного екземпляру бази даних. Задача сервісу буде полягати у тому, щоб зчитувати дані, виконувати комунікацію із сусідніми сервісами та виконувати операції запису за необхідністю. Слід зазначити, що обов'язковими вимогами є глобальна узгодженість за часом хостів, на яких будуть працювати сервіси.

Для підвищення ефективності використання рекомендується впровадити метод в програмне забезпечення самої бази даних, оскільки в даному випадку, підсистема буде мати прямий доступ до необхідних даних та метаданих.

Також рекомендується реалізувати метод активної антиентропії з використанням шини подій, яка дозволить майже миттєво розпочати процес ідентифікації неузгодженості. Під час операцій створення або оновлення документу буде створюватись подія із необхідною інформацією щодо об'єкта. Після цього подія потрапляє до черги на обробку активною антиентропією. Приклад роботи активної антиентропії зображено на рис. 3.39.

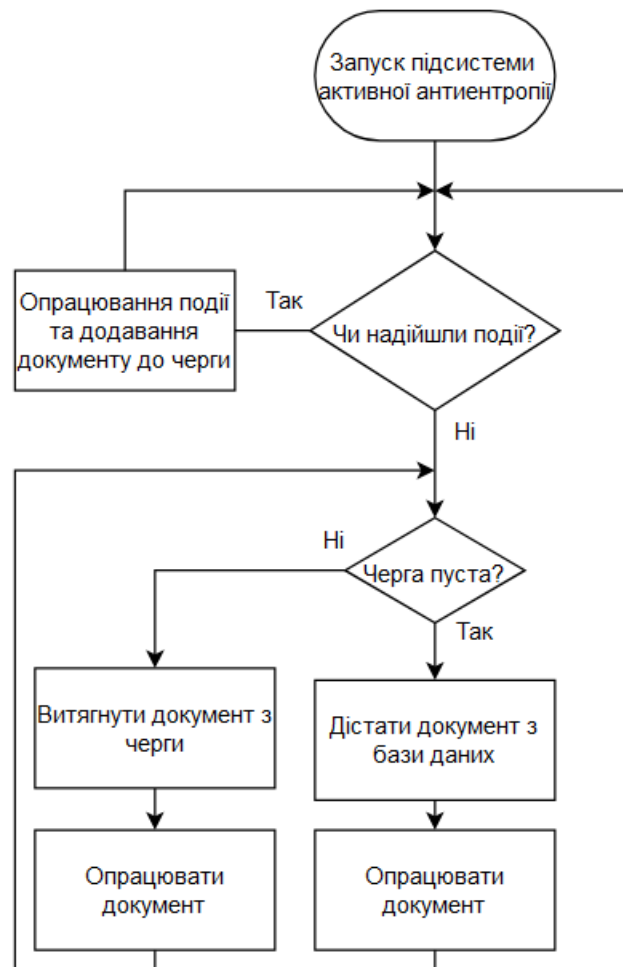


Рисунок 3.39 – Функціональна схема роботи активної антиентропії

Висновки до розділу 3

Основною метою було перевірити на практиці запропоновані методи та порівняти їх з існуючим рішенням. Це вимагає наближеного до реальності середовища, щоб виконати системну перевірку та подивитись на те, як сама система буде себе поводити. Це дозволяє побачити поведінку інформаційної системи загалом в результаті впровадження того чи іншого метода.

Прототип інформаційної системи було вирішено створити з області фінансів, оскільки тут узгодженість має ключове значення. Розподілена база даних такої системи може використовуватись для різних цілей і її активними користувачами можуть бути не тільки клієнти установи, але і їх співробітники для проведення спеціальних процедур. Наприклад, це може бути здійснення фінансового моніторингу за активністю клієнтів та поступаючим коштам на їх рахунки з метою викриття незаконних доходів.

Прототип інформаційної системи був побудований з можливістю використання будь-якого методу забезпечення узгодженості. Додаткового було реалізовано програму для проведення тесту для порівняння та оцінки того чи іншого методу.

В результаті досліджень було отримано результати, згідно яких розроблені методи дійсно пришвидшують узгодженість прототипу інформаційної системи в порівнянні з механізмом узгодження MongoDB ReplicaSet, але при певній кількості операцій запису за одиницю часу:

- використання удосконаленого методу активної антиентропії дозволило збільшити швидкість узгодження на 11 % при невеликому навантаженні на інформаційну систему;
- використання транзакційного годинника дозволило збільшити швидкість узгодження критично важливих даних на 77 %;
- модифікований спектральний фільтр Блума дозволяє пришвидшити розроблений механізм активної антиентропії на 16 % та покращити колізійну стійкість на 33 %;

- впровадження нового метода хешування дозволяє підвищити колізійну стійкість до 100 % при хешуванні даних бази даних різного розміру.

ВИСНОВКИ

В дисертаційній роботі досліджено та розроблено методи забезпечення узгодженості даних в інформаційних системах, які використовують розподілені нереляційні документо-орієнтовані бази даних. Концентрація була на використанні MongoDB як еталонного рішення для проведення дослідів та оцінки отриманих результатів.

Було отримано наступні наукові та практичні результати:

- удосконалено метод узгодження даних з використанням транзакційного годинника, який дозволяє підвищити швидкодію синхронізації критично важливих даних в розподілених інформаційних системах з великою кількістю операцій запису через введення пріоритетності транзакцій та використання транзакційної логіки;
- удосконалено процес активної антиентропії, який відрізняється від існуючого використанням алгоритму хешування PH2 замість дерева Меркла та використанням модифікованого спектрального фільтра Блума, що дозволяє підвищити швидкодію процесу узгодження даних в розподілених інформаційних системах;
- вперше розроблено метод хешування на основі простих чисел PH2, який відрізняється від відомих CRC32 та MurmurHash3 та дозволяє збільшити колізійну стійкість процесу узгодження даних в розподілених інформаційних системах;
- набула подальшого розвитку інформаційна технологія, яка ґрунтується на комплексному застосуванні запропонованих методів та моделей, що дозволяє підвищити швидкодію та колізійну стійкість;
- показано, що застосування транзакційного годинника пришвидшує узгодження даних на 77 % при підвищеному навантаженні на систему операціями запису. Вдосконалення метода активної антиентропії

пришвидшує узгодження даних на 11 % при невеликих навантаженнях на систему. Модифікований спектральний фільтр Блума дозволяє пришвидшити розроблений механізм активної антиентропії на 16 % та покращити колізійну стійкість на 33 %. Впровадження нового метода хешування дозволяє підвищити колізійну стійкість під час процесу узгодження до 100 % при хешуванні даних різного розміру

- реалізовано спеціальний сервіс транзакційного годинника із власним програмним інтерфейсом для використання розробленого методу на основі MongoDB;
- реалізовано спеціальний сервіс активної антиентропії із власним програмним інтерфейсом для використання розробленого методу активної антиентропії на основі MongoDB;
- реалізовано прототип фінансової інформаційної системи для використання для дослідження узгодженості даних при застосуванні різних методів забезпечення узгодженості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is 5G in IoT in 2022?. Available at: <https://www.iotallknow.com/4g-5g-iot-technology/> (Accessed 01 October 2023);
2. Antarctica Gets A Cellular Network For The 'Internet Of Things'. Available at: <https://www.forbes.com/sites/aarontilley/2016/01/19/antarctica-gets-a-cellular-network-for-the-internet-of-things/?sh=7ee92e4277a6> (Accessed 01 October 2023);
3. The 2022 Manual to Payment System Architecture. Available at: <https://devoxsoftware.com/blog/the-2022-manual-to-payment-system-architecture/> (Accessed 01 October 2023);
4. What evolves never dies: Why SIEM will eat your data lake in the long run. Available at: <https://www.linkedin.com/pulse/what-evolves-never-dies-why-siem-eat-your-data-lake-long-rochford> (Accessed 04 October 2023);
5. Horizontal vs Vertical scaling: An in-depth Guide. Available at: <https://www.nops.io/blog/horizontal-vs-vertical-scaling/> (Accessed 04 October 2023);
6. What is Data Inconsistency in DBMS? Available at: <https://www.geeksforgeeks.org/what-is-data-inconsistency-in-dbms/> (Accessed 04 October 2023);
7. Information system. Available at: <https://www.britannica.com/topic/information-system> (Accessed 04 October 2023);
8. Сосновенко, В. В. (2022) Мобільний застосунок для діджиталізації процедури прийому вторинних відходів : дипломна робота : 122 Комп'ютерні науки. Київ. 2022. 76 с;
9. Нікітін В. А., Крилов Є. В., Корнага Я. І., Анікін В. К. (2021) Комбінований метод індексування у нереляційних базах даних, 1 (38), 3-9. <https://doi.org/10.20535/1560-8956.38.2021.232948> ;
10. What is NoSQL? Available at: <https://www.mongodb.com/nosql-explained> (Accessed 05 October 2023);

11. A Dive Deep into Types of NoSQL Databases. Available at: <https://blazeclan.com/blog/dive-deep-types-nosql-databases/> (Accessed 05 October 2023);
12. What are ACID Properties in Database Management Systems? Available at: <https://www.mongodb.com/basics/acid-transactions> (Accessed 05 October 2023);
13. What is BASE in Database and System Design? Available at: <https://medium.com/ayokoding/what-is-base-in-database-and-system-design-472c9747ccfc> (Accessed 05 October 2023);
14. CAP theorem. Available at: <https://subscription.packtpub.com/book/data/9781800560659/2/ch02lv11sec05/cap-theorem> (Accessed 06 October 2023);
15. What is CAP theorem? Available at: <https://its-about.engineering/what-is-cap-theorem> (Accessed 06 October 2023)
16. Transactions in distributed systems. Available at: <https://medium.com/nerd-for-tech/transactions-in-distributed-systems-b5ceea869d7d> (Accessed 07 October 2023);
17. PACELC Theorem & Distributed Databases. Available at: <https://ritesh-kapoor.medium.com/pacelc-theorem-and-distributed-databases-301d971deda3> (Accessed 07 October 2023);
18. Compensating Action: Ensuring consistency in distributed systems. Available at: <https://www.linkedin.com/pulse/compensating-action-ensuring-consistency-distributed-systems-worku> (Accessed 07 October 2023);
19. Consistency Guarantees in Distributed Systems Explained Simply. Available at: <https://kousiknath.medium.com/consistency-guarantees-in-distributed-systems-explained-simply-720caa034116> (Accessed 07 October 2023);
20. Steen M., Tanenbaum A. (2017) Distributed Systems: 3rd Edition. CreateSpace;
21. What is Strict Consistency vs Eventual Consistency? Available at: <https://www.cohesity.com/blogs/strict-vs-eventual-consistency/> (Accessed 08 October 2023);

22. Sequential Consistency In Distributive Systems. Available at: <https://www.geeksforgeeks.org/sequential-consistency-in-distributive-systems/> (Accessed 08 October 2023);
23. What is causal consistency in distributed systems? Available at: <https://www.educative.io/answers/what-is-causal-consistency-in-distributed-systems> (Accessed 08 October 2023);
24. Eventual Consistency Definition. Available at: <https://www.scylladb.com/glossary/eventual-consistency/> (Accessed 08 October 2023);
25. MongoDB – a Distributed Database. Available at: <https://ravisrc.medium.com/mongodb-say-no-to-sql-8ef7b94a6ae9> (Accessed 08 October 2023);
26. Understanding Paxos. Available at: <https://people.cs.rutgers.edu/~pxk/417/notes/paxos.html> (Accessed 09 October 2023);
27. The Raft Consensus Algorithm. Available at: <https://raft.github.io/> (Accessed 09 October 2023);
28. Gossip Protocol Explained. Available at: <https://highscalability.com/gossip-protocol-explained/> (Accessed 09 October 2023);
29. Logical Clock in Distributed System. Available at: <https://www.geeksforgeeks.org/logical-clock-in-distributed-system/> (Accessed 09 October 2023);
30. Lamport's logical clock. Available at: <https://www.geeksforgeeks.org/lamports-logical-clock/> (Accessed 09 October 2023);
31. Vector Clocks: The Simple Way to Keep Distributed Systems in Sync. Available at: <https://www.linkedin.com/pulse/vector-clocks-simple-way-keep-distributed-systems-sync-maheshwari> (Accessed 09 October 2023);
32. Understanding the concept of read-repair in the context of databases. Available at: <https://copyprogramming.com/howto/what-does-the-term-read-repair-mean-when-applied-to-databases> (Accessed 10 October 2023);

33. What is Two-Phase Commit Protocol? Available at: <https://www.dremio.com/wiki/two-phase-commit-protocol/> (Accessed 10 October 2023);
34. Mukhin V., Zavgorodnii V., Nikitin V., Kornaga Y., Fartushnyi I., Stepanov A. (2021) Method of Determining the Required Number of Database Nodes in a Distributed Data Processing System. IEEE, 2021 IEEE 3rd International Conference on Advanced Trends in Information Theory (ATIT). <https://doi.org/10.1109/ATIT54053.2021.9678569> ;
35. Anti-Entropy Protocols. Available at: <https://www.linkedin.com/pulse/anti-entropy-protocols-yeshwanth-n-2c> (Accessed 22 October 2023);
36. Нікітін В. А., Крилов Є. В. (2021) Порівняння методів хешування у підтримці консистентності розподілених баз даних. Адаптивні системи автоматичного управління, 1 (40), 48-53. <https://doi.org/10.20535/1560-8956.40.2022.261646> ;
37. How Do Merkle Trees Work? Available at: <https://www.baeldung.com/cs/merkle-trees> (Accessed 22 October 2023);
38. Нікітін В. А., Крилов Є. В., Корнага Я. І., Анікін В. К. (2021) Модифікація алгоритму хешування з метою підвищення швидкодії операцій у нереляційних базах даних, 2 (39), 39-43. <https://doi.org/10.20535/1560-8956.39.2021.247395> ;
39. Bloom Filters – Introduction and Implementation. Available at: <https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/> (Accessed 22 October 2023);
40. Ramabaja L. (2019) The Distributed Bloom Filter. arXiv. 1-17. <https://doi.org/10.48550/arXiv.1910.07782>
41. Kumar A., Xu J., Wang J. (2006) Space-Code Bloom Filter for Efficient Traffic Flow Measurement. IEEE Journal on Selected Areas in Communications. 24 (12). 1-12. <https://doi.org/10.1109/JSAC.2006.884032> ;
42. Cohen S., Matias Y. (2003) Spectral Bloom Filters. Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. 1-12. <http://dx.doi.org/10.1145/872757.872787> ;

43. Replica Set Fundamental Concepts. Available at: <https://www.mongodb.com/docs/v2.2/core/replication/> (Accessed 28 October 2023);
44. Transactions Busted! Available at: <https://faun.pub/cassandra-light-weight-transactions-busted-229e0633c8b> (Accessed 01 November 2023);
45. DynamoDB Streams Use Cases and Design Patterns. Available at: <https://aws.amazon.com/blogs/database/dynamodb-streams-use-cases-and-design-patterns/> (Accessed 01 November 2023);
46. Крилов Є. В., Нікітін В. А. (2023) Використання транзакційного годинника для пришвидшення процесу узгодження даних в розподілених системах. Науковий вісник Ужгородського університету. Серія Математика і інформатика, 42(1), 188-192. [https://doi.org/10.24144/2616-7700.2023.42\(1\).188-192](https://doi.org/10.24144/2616-7700.2023.42(1).188-192) ;
47. Нікітін В. А., Крилов Є. В. (2023) Active Anti-Entropy mechanism based on Spectral Bloom filter and PH-2 hash algorithm for reconciliation of replicas of NoSQL distributed document oriented databases. Information Technology and Society, 3 (9), 63-67. <https://doi.org/10.32689/maup.it.2023.3.8> ;
48. Нікітін В. А., Крилов Є. В. (2023) Primary-based Spectral Bloom filter for the ensuring consistency in distributed document-based NoSQL databases using Active Anti-Entropy mechanism. Computer Systems and Information Technologies, (3), 75–80. <https://doi.org/10.31891/csit-2023-3-9> ;
49. Нікітін В. А., Крилов Є. В. (2022) Алгоритм хешування з підвищеною колізійною стійкістю для підтримки консистентності в розподілених базах даних. Адаптивні системи автоматичного управління, 2 (41), 45-57. <https://doi.org/10.20535/1560-8956.41.2022.271338> .

ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ В ТОВ “ІНТЕХФОРВАРД”

АКТ

про впровадження результатів дисертаційної роботи
Нікітіна Валерія Андрійовича

Комісія в складі: голови комісії менеджера С.І. Крилової, членів комісії: інженера І.І. Казмирчука; склала цей акт про те, що результати, отримані в дисертаційному дослідженні аспіранта Національного технічного університету «Київський політехнічний інститут імені Ігоря Сікорського» Нікітіна Валерія Андрійовича, реалізовані при впровадженні в розподіленій інформаційній системі фінансового моніторингу. Ця система була розроблена аспірантом для проведення експериментального дослідження наукових результатів, які були досягнуті в процесі навчання в аспірантурі. Тема дисертації «Методи підвищення ефективності узгодженості даних в інформаційних системах».

Прийняті до впровадження такі результати дослідження: використання транзакційного годинника та модифікованого алгоритму формування спектрального фільтра Блума в процесі активної ентропії для підвищення швидкодії процесу узгодження даних в інформаційних системах,

Імітаційна модель фінансового моніторингу з використанням транзакційного годинника показала підвищення швидкодії узгодження даних віртуальної розподіленої системи.

Даний акт не є підставою для фінансових взаєморозрахунків.

Голова комісії:
менеджер



С.І. Крилова

Члени комісії:

інженер

І.І. Казмирчук

20.12.2023

ДОДАТОК Б. АКТ ВПРОВАДЖЕННЯ В НАВЧАЛЬНИЙ ПРОЦЕС

Декан факультету інформатики
та обчислювальної техніки
КПІ ім. Ігоря Сікорського

 Я.І. Корнага
« 26 » листопада 2024р.

АКТ

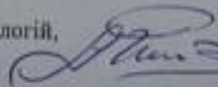
про впровадження в навчальний процес кафедри Інформаційних систем та технологій факультету Інформатики та обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» результатів дисертаційної роботи Нікітіна Валерія Андрійовича «Методи підвищення ефективності узгодженості даних в інформаційних системах», поданої на здобуття наукового ступеня доктора філософії.

Ми, що нижче підписалися: завідувач кафедри Інформаційних систем та технологій, д.т.н. проф. Ролік О.І., доцент кафедри інформаційних систем та технологій, к.т.н., доц. Крилов Є.В., вчений секретар кафедри, к.ф.-м.н., доц. Гавриленко О.В. розглянувши матеріали навчально-методичного забезпечення курсів: «Штучний інтелект в задачах обробки зображень», «Мережеве управління та протоколи», «Технології Інтернету речей», «Технології та протоколи мультисервісних мереж» які підготовлені та викладаються асистентом Нікітином В. А., встановили, що в навчальному процесі на лабораторних заняттях, а також в процесі підготовки бакалаврських атестаційних робіт використовуються такі наукові результати, отримані дисертантом особисто:

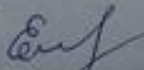
- транзакційний годинник для узгодження вузлів розподілених нереляційних документо-орієнтованих баз даних розділених мережею;
- механізм активної антиентропії на основі протоколу пліток з використанням спектрального фільтру Блума та алгоритму хешування для узгодження вузлів розподіленої не реляційної документо-орієнтованої бази даних;
- інформаційна технологія для автоматизації процесів розробки програмного коду.

Впровадження результатів дисертаційної роботи Нікітін В.А. в навчальний процес дозволило підвищити якість підготовки студентів освітнього рівня «Бакалавр» 126 «Інформаційні системи та технології» на кафедрі Інформаційних систем та технологій КПІ ім. Ігоря Сікорського.

Завідувач кафедри
інформаційних систем та технологій,
д.т.н., проф.

 О.І. Ролік

Вчений секретар кафедри
інформаційних систем та технологій,
к.ф.-м.н., доц.

 О.В. Гавриленко

ДОДАТОК В. РЕАЛІЗАЦІЯ ФОРМУВАННЯ СПЕКТРАЛЬНОГО ФІЛЬТРУ БЛУМА З ВИКОРИСТАННЯМ ПРОСТИХ ЧИСЕЛ

```
from algorithms import Frozen
```

```
class SpectralBloomFilter(Frozen):
    PRIMES = [2, 3, 5, 7, 11, 13, 17]

    def __init__(self, *args, **kwargs):
        super().__init__()
        self._size = len(SpectralBloomFilter.PRIMES) + 1
        self._entries = [0] * self._size

    @Frozen.decorator
    def add(self, _bytes: bytes):
        for _b in _bytes:
            self._add_single(_b)

    @Frozen.decorator
    def _add_single(self, _byte: int):
        is_jocker = True
        for i, prime in enumerate(SpectralBloomFilter.PRIMES):
            if _byte % prime == 0:
                self._entries[i] = (self._entries[i] + 1) % 255
                is_jocker = False

        if is_jocker:
            self._entries[-1] = (self._entries[-1] + 1) % 255

    def get(self) -> bytes:
        return bytes(self._entries)
```

ДОДАТОК Г. РЕАЛІЗАЦІЯ АЛГОРИТМУ ХЕШУВАННЯ PH2-48

```
class PH2(Frozen):
```

```
    def __init__(self):
        super().__init__()
        self._bytes = bytearray()
```

```
    @Frozen.decorator
```

```
    def append(self, _bytes: bytes):
        self._bytes.extend(_bytes)
```

```
    def hashing(self) -> bytes:
        required_bytes_per_block = BLOCK_SIZE // 8
```

```
        need_to_add_bytes = required_bytes_per_block - (len(self._bytes) % required_bytes_per_block)
        need_to_add_bytes = need_to_add_bytes % required_bytes_per_block # to avoid extra adding
        self._bytes.extend([0] * need_to_add_bytes)
```

```
        bytes_len = len(self._bytes)
```

```
    def search(alist, item):
        'Locate the leftmost value exactly equal to item'
        i = bisect_left(alist, item)
        if i != len(alist) and alist[i] == item:
            return True
```

```
        return False
```

```
        sum_regular = 0
        overflow_regular = 0
        sum_primes = 0
        overflow_primes = 0
        primes_count = 0
        regular_count = 0
```

```
        for i in range(0, bytes_len, required_bytes_per_block):
            start = i
            end = i + required_bytes_per_block
            part = self._bytes[start:end:1]
            item = int.from_bytes(part, byteorder='big', signed=False)
```

```

    if search(PRIME_NUMBERS, item):
        diff = MAX_VALUE - sum_primes
        if item >= diff:
            sum_primes = item - diff
            overflow_primes = (overflow_primes + 1) % MAX_VALUE
        else:
            sum_primes += item
        primes_count = (primes_count + 1) % MAX_VALUE
    else:
        diff = MAX_VALUE - sum_regular
        if item >= diff:
            sum_regular = item - diff
            overflow_regular = (overflow_regular + 1) % MAX_VALUE
        else:
            sum_regular += item
        regular_count = (regular_count + 1) % MAX_VALUE

    return bytes(
        [
            regular_count % MAX_VALUE,
            primes_count % MAX_VALUE,
            sum_regular % MAX_VALUE,
            overflow_regular % MAX_VALUE,
            sum_primes % MAX_VALUE,
            overflow_primes % MAX_VALUE,
        ]
    )

def digest(self) -> int:
    hashed = self.hashing()

    ph2_hash = 0
    def push(part):
        nonlocal ph2_hash
        part = part % MAX_VALUE
        ph2_hash <<= BLOCK_SIZE
        ph2_hash |= part
    for b in hashed:
        push(b)

    return ph2_hash

```

ДОДАТОК Е. DOCKER-COMPOSE ФАЙЛ ДЛЯ РОЗГОРТАННЯ РОЗПОДІЛЕНОЇ БАЗИ ДАНИХ

```
services:
  mongo1:
    hostname: mongo1
    image: mongo
    networks:
      default:
        ipv4_address: 172.20.0.11
    expose:
      - 27017
    restart: always
    #command: mongod --replSet my-mongo-set
  mongo2:
    depends_on:
      - mongo1
    hostname: mongo2
    image: mongo
    networks:
      default:
        ipv4_address: 172.20.0.12
    expose:
      - 27017
    restart: always
    #command: mongod --replSet my-mongo-set
  mongo3:
    depends_on:
      - mongo2
    hostname: mongo3
    image: mongo
    networks:
      default:
        ipv4_address: 172.20.0.13
    expose:
      - 27017
    restart: always
    #command: mongod --replSet my-mongo-set
  mongo4:
    depends_on:
      - mongo3
    hostname: mongo4
    image: mongo
    networks:
      default:
        ipv4_address: 172.20.0.14
    expose:
      - 27017
```

```

restart: always
#command: mongod --replSet my-mongo-set
mongo5:
  depends_on:
    - mongo4
  hostname: mongo5
  image: mongo
  networks:
    default:
      ipv4_address: 172.20.0.15
  expose:
    - 27017
  restart: always
  command: mongod --replSet my-mongo-set
mongo6:
  depends_on:
    - mongo5
  hostname: mongo6
  image: mongo
  networks:
    default:
      ipv4_address: 172.20.0.16
  expose:
    - 27017
  restart: always
  #command: mongod --replSet my-mongo-set
mongo7:
  depends_on:
    - mongo6
  hostname: mongo7
  image: mongo
  networks:
    default:
      ipv4_address: 172.20.0.17
  expose:
    - 27017
  restart: always
  #command: mongod --replSet my-mongo-set
mongo8:
  depends_on:
    - mongo7
  hostname: mongo8
  image: mongo
  networks:
    default:
      ipv4_address: 172.20.0.18
  expose:
    - 27017

```

```
restart: always
#command: mongod --replSet my-mongo-set
networks:
  default:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/24
```

ДОДАТОК Д. КОМАНДИ ДЛЯ НАЛАШТУВАННЯ MONGODB REPLICASET

```
config = {
  "_id" : "my-mongo-set",
  "writeConcernMajorityJournalDefault": false,
  "members" : [
    { "_id" : 0, "host" : "172.20.0.11:27017", "priority": 1 },
    { "_id" : 1, "host" : "172.20.0.12:27017", "priority": 0.5 },
    { "_id" : 2, "host" : "172.20.0.13:27017", "priority": 0.5 },
    { "_id" : 3, "host" : "172.20.0.14:27017", "priority": 0, "votes": 0 },
    { "_id" : 4, "host" : "172.20.0.15:27017", "priority": 0, "votes": 0 },
    { "_id" : 5, "host" : "172.20.0.16:27017", "priority": 0, "votes": 0 },
    { "_id" : 6, "host" : "172.20.0.17:27017", "priority": 0, "votes": 0 },
    { "_id" : 7, "host" : "172.20.0.18:27017", "priority": 0, "votes": 0 }
  ],
  "settings": {
    "chainingAllowed": false
  }
};
rs.initiate(config);
db.adminCommand({ setDefaultRWConcern: 8, defaultWriteConcern: {w: 8}})
```