

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

БЕЛОУС РОМАН ВОЛОДИМИРОВИЧ

УДК 004.65

**МЕТОДИ І ПРОГРАМНІ ЗАСОБИ ПІДВИЩЕННЯ
ЕФЕКТИВНОСТІ ВИКОНАННЯ ЗАПИТІВ У
ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ**

121 Інженерія програмного забезпечення

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 _____ Р. В. Белоус

Науковий керівник Крилов Євген Володимирович,

к.т.н., доцент

АНОТАЦІЯ

Белоус Р. В. Методи і програмні засоби підвищення ефективності виконання запитів у високонавантажених системах. – Кваліфікаційна наукова праця на правах рукопису. Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 – Інженерія програмного забезпечення. – Національний Технічний Університет України «Київський Політехнічний Інститут імені Ігоря Сікорського», Київ, 2025.

Актуальність дослідження обумовлена стрімким зростанням обсягів даних та кількості запитів, що обробляються сучасними розподіленими інформаційними системами. Високонавантажені розподілені бази даних забезпечують обробку інформації в режимі реального часу, підтримуючи роботу широкого спектра застосувань – від соціальних мереж до корпоративних хмарних платформ. Однак, збільшення кількості та складності запитів створює значні труднощі у підтримці високої продуктивності та надійності. Зокрема, істотний вплив на ефективність мають мережеві затрати при передаванні даних між вузлами, коректне підтримання узгодженості даних та оптимальне розподілення навантаження.

У дисертаційній роботі запропоновано науково-обґрунтовані методи та програмні засоби, що спрямовані на підвищення ефективності виконання запитів у високонавантажених розподілених системах. Для досягнення цієї мети було враховано потребу в оптимізації мережевого трафіку, вдосконаленні механізмів узгодженості даних та ефективному ребалансуванні ресурсів. Особливу увагу приділено оптимізації поведінки розподілених систем, що використовують алгоритми консенсусу Raft, оскільки саме в цих підходах закладено основу для надійного оновлення та синхронізації даних між множиною вузлів. Оптимізація Raft та пов'язаних з ним процесів передачі та узгодження даних дозволяє суттєво покращити час відгуку системи та знизити мережеве навантаження, що є критичним для стабільного функціонування високонавантажених застосувань.

Вперше розроблено метод мінімізації обсягу мережевого трафіку у Raft Consensus Algorithm розподілених базах даних, який базується на поєднанні принципів, притаманних як Raft, так і Leaderless Replication та ґрунтується на

попередньому обміні метаданими між вузлами та в подальшому збереженні отриманих результатів. Суть методу полягає в тому, що перед початком передавання основних даних, вузли спочатку обмінюються метаданими, які містять інформацію про стан кардинальності та вектори даних. Це дозволяє зменшити обсяг даних, що передаються через мережу, оскільки вузли можуть узгодити лише ті зміни, які дійсно потребують синхронізації. Після цього, на основі отриманих метаданих, відбувається локальне збереження результатів, що мінімізує кількість переданих даних, знижуючи тим самим навантаження на мережу та підвищуючи ефективність роботи алгоритму Raft у розподілених базах даних.

Удосконалено метод оптимізації запитів у розподілених базах даних шляхом удосконалення ребалансування даних за допомогою генетичних алгоритмів з елітарністю та адаптивним схрещенням. Цей підхід дозволяє ефективніше розподіляти дані між вузлами системи, що зменшує час виконання запитів. Використання елітарності забезпечує збереження найкращих рішень на кожному етапі алгоритму, а адаптивне схрещення підвищує різноманітність рішень та прискорює конвергенцію до оптимального. У результаті, модифікований метод ребалансування сприяє підвищенню ефективності виконання запитів у розподілених базах даних, особливо в умовах високих навантажень.

Удосконалено метод узгодженості даних у розподілених базах даних на основі методу Левенштейна, який відрізняється від існуючих підходів і мінімізує обсяг мережевого трафіку під час процесу узгодження даних, особливо при частих і малих змінах. Цей метод використовує вдосконалений метод Левенштейна, що дозволяє передавати тільки зміни замість повних копій даних. Завдяки цьому, значно зменшується кількість переданих даних по мережі, що особливо важливо в умовах частих оновлень і модифікацій невеликих обсягів текстів даних, забезпечуючи ефективну синхронізацію реплік і підтримуючи високу продуктивність системи.

Для дослідження отриманих наукових результатів було розроблено спеціалізоване програмне забезпечення, яке являє собою електронний онлайн-журнал для студентів, викладачів, батьків та адміністрації навчальних закладів.

Цей програмний продукт дозволяє вчителям виставляти оцінки, створювати та призначати домашні завдання, а також вести звітність щодо успішності студентів. Студенти мають доступ до персонального кабінету, де відображається їхня академічна успішність, а батьки можуть отримувати доступ до інформації, що стосується їхньої дитини, включаючи оцінки та завдання. Адміністрація навчального закладу, у свою чергу, може генерувати різноманітні звіти та статистику щодо успішності студентів та інших показників.

Для забезпечення ефективності, надійності та масштабованості застосунок було побудовано на основі Raft-архітектури, яка гарантує узгодженість даних у розподіленій системі. Застосунок реалізовано з використанням сучасного стека технологій, включаючи Docker, Laravel та Vue.js. Застосування цих технологій дозволило створити гнучку, стійку до помилок і легко масштабовану систему, яка ефективно підтримує всі необхідні функції та забезпечує можливість дослідження і аналізу наукових результатів у контексті роботи розподілених систем.

Дисертаційна робота складається зі вступу, 5 розділів, загальних висновків, списку використаних джерел із 47 найменувань та 2 додатків. Загальний обсяг дисертації становить 152 сторінок, з яких 126 сторінки основного тексту, містить 45 рисунків та 6 таблиць.

Ключові слова: бази даних, розподілені системи, розподілені бази даних, ребалансування, узгодженість, нереляційні бази даних, NoSQL, інформаційні системи, система управління, архітектура програмного забезпечення, інформаційні технології, програмне забезпечення, Raft, мережевий трафік, цілісність, доступність, інженерія програмного забезпечення.

ABSTRACT

Belous R. V. Methods and Software Tools for Improving the Efficiency of Query Execution in High-Load Systems. – Qualification scientific work as a manuscript. Dissertation for obtaining the degree of Doctor of Philosophy in specialty 121 – Software Engineering. – National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2025.

The relevance of this research is driven by the rapid growth in data volumes and the increasing number of queries processed by modern distributed information systems. High-load distributed databases provide real-time information processing, supporting a wide range of applications—from social networks to enterprise cloud platforms. However, the increasing number and complexity of queries pose significant challenges in maintaining high performance and reliability. Notably, network costs during data transmission between nodes, proper data consistency, and optimal load distribution significantly impact the overall efficiency of these systems.

This dissertation proposes scientifically grounded methods and software tools aimed at improving query performance in high-load distributed systems. To achieve this goal, the research focuses on optimizing network traffic, enhancing data consistency mechanisms, and implementing efficient resource rebalancing. Special attention is paid to optimizing the behavior of distributed systems that utilize the Raft consensus algorithm, as these approaches form the foundation for reliable data updates and synchronization across multiple nodes. The optimization of Raft and its related processes for data transmission and consistency allows for a significant reduction in response time and network load, which is critical for the stable operation of high-load applications.

For the first time, a method for minimizing network traffic volume in the Raft Consensus Algorithm for distributed databases has been developed. This method combines principles inherent to both Raft and leaderless replication, based on the preliminary exchange of metadata between nodes and subsequent result caching. The essence of the method lies in the initial exchange of metadata containing information on cardinality and data vectors before transmitting primary data. This approach reduces the

volume of data transmitted over the network, as nodes can synchronize only the changes that require updates. Following this metadata exchange, local caching of the results occurs, minimizing the amount of transmitted data and consequently reducing network load while enhancing the efficiency of the Raft algorithm in distributed databases.

The method for query optimization in distributed databases has been improved by enhancing data rebalancing using genetic algorithms with elitism and adaptive crossover. This approach enables more efficient data distribution across system nodes, reducing query execution time. The inclusion of elitism ensures that the best solutions are preserved at each stage of the algorithm, while adaptive crossover increases solution diversity and accelerates convergence toward the optimal solution. As a result, the modified rebalancing method improves query performance in distributed databases, especially under high-load conditions.

Additionally, a data consistency method for distributed databases has been enhanced using the Levenshtein-based approach. Unlike existing methods, this approach minimizes network traffic during the data consistency process, particularly in scenarios involving frequent and minor changes. The method employs an advanced version of the Levenshtein algorithm to accurately identify minimal differences between data versions, allowing for the transmission of only the changes rather than full data copies. Consequently, the volume of transmitted data is significantly reduced, which is particularly important in environments with frequent updates and modifications of small data volumes. This ensures efficient replica synchronization while maintaining high system performance.

A specialized software system was developed to investigate the scientific results obtained. The software represents an electronic online journal for students, teachers, parents, and school administration. This product enables teachers to assign grades, create and manage homework, and generate performance reports for students. Students can access a personal dashboard displaying their academic performance, while parents can obtain information related to their child's grades and assignments. The school administration can generate various reports and statistics on student performance and other metrics.

To ensure efficiency, reliability, and scalability, the application was built based on Raft architecture, which guarantees data consistency in the distributed system. The application is implemented using a modern technology stack, including Docker for containerization and deployment management, Laravel for backend development following MVC principles, and Vue.js for building a dynamic and responsive user interface. These technologies allowed for the creation of a flexible, fault-tolerant, and easily scalable system that effectively supports all necessary functions and enables the research and analysis of scientific results within the context of distributed systems.

The dissertation consists of an introduction, 5 chapters, general conclusions, a list of sources used with 47 names and 2 appendices. The total volume of the dissertation is 152 pages, of which 126 pages are the main text, contains 45 figures and 6 tables.

Keywords: databases, distributed systems, distributed databases, rebalancing, consistency, non-relational databases, NoSQL, information systems, management system, software architecture, information technology, software, Raft, network traffic, integrity, availability, software engineering.

Список публікацій здобувача

Наукові праці, в яких опубліковано основні наукові результати дисертації:

1. **Белоус Р.В.**, Крилов Є.В. Оптимізація використання RAFT в розподілених системах з базами даних // Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки Том 35 (74). № 3, 2024. DOI: <https://doi.org/10.32782/2663-5941/2024.3.1/07>.
2. **Белоус Р.В.**, Крилов Є.В., Оптимізація часу процесу узгодженості даних в NOSQL. Вісник Хмельницького національного університету, №3 (321), с. 37-42, 2023. DOI: <https://www.doi.org/10.31891/2307-5732-2023-321-3-37-42>.
3. **Belous, R.**, Krylov, I., Mukhin, V., Zavgorodnii, V., Kornaga, Y., Zavgorodnya, A., Rybalochka, A., Kornaga, V.,. Devising a method to identify an incoming object based on the combination of unified information spaces // Eastern-European Journal of Enterprise Technologies. №3 (2), (111), 2021. P. 35 – 44. DOI: <https://doi.org/10.15587/1729-4061.2021.229568>.
4. **Белоус Р.В.**, Крилов Є.В., Анікін В.К.. Методи оптимізації запитів розподілених БД. Адаптивні системи автоматичного управління. Том 2 № 39 (2021). DOI: <https://doi.org/10.20535/1560-8956.39.2021.247364>.
5. **Белоус Р.В.**, Крилов Є.В. Мінімізація мережевого трафіку в Raft-like consensus algorithm. Науково-технічний збірник «Комунальне господарство міст». Серія: технічні науки та архітектура, <https://doi.org/10.33042/2522-1809-2024-4-185-2-6>.
6. **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-ofscience.17.02.2023.065> .
7. **Белоус Р.В.**, Анікін В.К., Крилов Є.В.. Ребалансування даних у розподілених базах даних за допомогою генетичних алгоритмів з адаптивним схрещенням та елітарністю. // жовтня 18, 2024; Cambridge, UK VII International Scientific and Practical Conference «Education and science of today: intersectoral issues and development of sciences». DOI: <https://doi.org/10.36074/logos-18.10.2024.045> .
8. **Белоус Р.В.**, Анікін В.К. Ребалансування розподілених баз даних за допомогою генетичних алгоритмів // VII Міжнародна науково-практична

конференція «Теоретичні та емпіричні наукові дослідження: концепції та тенденції», 16 серпня 2024 р., Оксфорд, Велика Британія. DOI: 10.36074/logos-16.08.2024.

ЗМІСТ

ВСТУП.....	14
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	20
РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ ПІДВИЩЕННЯ ШВИДКОДІЇ ВИКОНАННЯ ЗАПИТІВ У ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ.....	21
1.1 Класифікація високонавантажених систем.....	21
1.2 Технології виконання запитів у високонавантажених систем.....	23
1.3 Аналіз розподілених баз даних у високонавантажених системах	25
1.3.1 Централізовані та розподілені бази даних	25
1.3.2 Проектування розподілених баз даних	30
1.4 Основні технології розподілених баз даних, що впливають на швидкодію запитів.....	32
1.4.1 Фрагментація даних у розподілених системах	32
1.4.2 Ребалансування даних у розподілених системах	33
1.4.3 Реплікація даних у розподілених системах	38
1.4.4 Узгодженість даних у розподілених системах	42
1.5 Постановка наукового завдання	47
Висновки до першого розділу	48
РОЗДІЛ 2 МЕТОДИ ПРИШВИДШЕННЯ УЗГОДЖЕНОСТІ ДАНИХ В ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ.....	50
2.1 Огляд протоколів консенсусу в розподілених системах.....	50
2.1.1 Paxos.....	52
2.1.2 Raft	53
2.1.3 Аналіз ефективності в умовах реальних сценаріїв.....	54
2.2 Методи оптимізації мережевого трафіку у Raft	55
2.3 Перетворення Лапласа-Стільтєса	56

2.4 Метод оптимізації мережевого трафіку Semi-join.....	57
2.4.1 Визначення та принцип роботи	57
2.4.1 Процес з'єднання та оцінка витрат при використанні Semi-join	58
2.4.3 Використання методу.....	59
2.4.4 Алгоритм SDD-1 для оптимізації запитів	60
2.5 Метод мінімізації мережевого трафіку в Raft.....	61
2.5.1 Логіка роботи методу	61
2.5.2 Математична модель	62
2.5.3 Використання методу.....	63
2.5.4 Локальне збереження результатів	66
2.6 Метод узгодженості даних на основі алгоритму Левенштейна	67
2.6.1 Опис методу	67
2.6.2 Математична модель	68
2.6.3 Визначення вагових коефіцієнтів.....	69
2.6.4 Використання методу.....	71
Висновки до другого розділу	73
РОЗДІЛ 3 МЕТОДИ ПОКРАЩЕННЯ ПРОЦЕСУ РЕБАЛАНСУВАННЯ В	
ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ.....	75
3.1 Процеси ребалансування в розподілених базах даних.....	75
3.2 Ребалансування даних на основі генетичних алгоритмів	78
3.3 Ребалансування з елітарністю та адаптивним схрещенням	83
3.3.1 Механізм елітарності в генетичних алгоритмах.....	84
3.3.2 Адаптивне схрещення	85
3.4 Процеси генетичного алгоритму з урахуванням елітарності та адаптивності	
.....	86
3.5 Формалізація задачі ребалансування	88

	12
3.6 Використання запропонованого методу ребалансування	91
Висновки до третього розділу	96
РОЗДІЛ 4 ПРОГРАМНІ ЗАСОБИ ЗАПРОПОНОВАНИХ МЕТОДІВ.....	98
4.1 Опис програмних засобів та структурних складових системи	98
4.1.1 Використані програмні засоби	98
4.1.2 Використані програмні засоби	100
4.1.3 Проектування інтегрального середовища	101
4.2 Програмна реалізація методів	106
4.2.1 Модифікований метод Raft Consensus Algorithm	106
4.2.2 Модифікований метод узгодженості на основі алгоритму Левенштейна	108
4.2.3 Метод ребалансування з використанням генетичних алгоритмів	111
4.3 Генерація тестових даних	112
Висновки до четвертого розділу	113
РОЗДІЛ 5 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНИХ МЕТОДІВ	115
5.1 Розробка інтегрального середовища системи онлайн-журналу	115
5.1.1 Бізнес-логіка інтегрального середовища системи онлайн-журналу	116
5.1.2 Реалізація серверного застосунку	118
5.1.3 Графічний інтерфейс реалізованого середовища	119
5.2 Експериментальні дослідження методів	121
5.2.1 Метод мінімізації мережевого трафіку в Raft.....	121
5.2.2 Модифікований метод узгодженості даних на основі методу Левенштейна	125
5.2.3 Ребалансування на основі генетичних алгоритмів елітарністю та адаптивним схрещенням	128

Висновки до п'ятого розділу	131
ВИСНОВКИ	133
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	135
ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ В НАВЧАЛЬНИЙ ПРОЦЕС	140
ДОДАТОК Б. ФАЙЛ DOCER-COMPOSE.YAML ДЛЯ РОЗГОРТАННЯ	141

ВСТУП

Актуальність теми. У сучасному світі спостерігається експоненціальне зростання обсягів даних та кількості користувачів, які взаємодіють з інформаційними системами. Високонавантажені системи стають нормою в різних галузях, включаючи фінанси, освіту, медицину, промисловість та інші. Швидке та ефективне виконання запитів у таких системах є критично важливим для забезпечення якісного обслуговування користувачів і підтримки конкурентоспроможності організацій.

Технології обробки та передачі даних постійно розвиваються, пропонуючи нові можливості для підвищення продуктивності систем. Розвиток хмарних технологій, IoT, Big Data та мереж п'ятого покоління 5G створює умови для інтеграції величезної кількості пристроїв і сервісів, що генерують та обробляють терабайти інформації. Проте це також призводить до збільшення навантаження на інформаційні системи та вимагає нових підходів до оптимізації їх роботи.

Високонавантажені системи стикаються з проблемами масштабування, ефективного розподілу ресурсів та забезпечення узгодженості даних у розподіленому середовищі. Традиційні методи обробки запитів часто не справляються з сучасними вимогами щодо швидкодії та надійності. Тому розробка нових методів та програмних засобів для підвищення ефективності виконання запитів є актуальною науковою та практичною задачею. Таким чином оптимізація виконання запитів є важливим елементом для забезпечення стабільної та ефективної роботи високонавантажених систем в умовах стрімкого зростання обсягів даних і кількості користувачів. Це дозволить підвищити продуктивність систем, знизити витрати на інфраструктуру та покращити якість сервісів, що надаються.

Зв'язок роботи з науковими програмами, планами, темами. Дослідження, що лягло в основу цієї дисертації, проводилося в Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського». Воно виконувалося відповідно до затвердженого Переліку пріоритетних тематичних напрямів наукових досліджень і науково-технічних розробок, які є актуальними на період до 2025 року. Дане дослідження було

частиною роботи над Ініціативною темою «Оптимізація функціонування веб-орієнтованих систем з великими обсягами даних», яка має державний реєстраційний номер 0117U004913. Пріоритетний напрям науково-технічної діяльності зосереджений на розвитку сучасних інформаційних та комунікаційних технологій, що мають вирішальне значення для подальшого прогресу в цій галузі. Галузь застосування результатів роботи охоплює широкий спектр досліджень і розробок у сферах природничих і технічних наук, що сприяє підвищенню ефективності та якості розробок у зазначених областях.

Мета і завдання дослідження. Метою дисертаційної роботи є підвищення ефективності виконання запитів у високонавантажених розподілених системах шляхом розробки нових методів та удосконалення існуючих методів оптимізації мережевого трафіку, часу узгодженості даних та ребалансування ресурсів, а також можливості їх впровадження в існуючі інформаційні системи.

Для досягнення поставленої мети були поставлені та вирішені такі задачі:

- огляд сучасних методів та технологій оптимізації виконання запитів у розподілених базах даних, включаючи алгоритми консенсусу Raft, Paxos та методи мінімізації мережевого трафіку ;
- аналіз існуючих підходів до узгодженості даних у розподілених системах, дослідження моделей узгодженості та методів синхронізації, зокрема, алгоритмів на основі Левенштейна ;
- розробка нового методу мінімізації обсягу мережевого трафіку у Raft Consensus Algorithm для розподілених баз даних, який базується на попередньому обміні метаданими між вузлами та подальшій матеріалізації отриманих результатів;
- модифікація методу узгодженості даних на основі алгоритму Левенштейна з ваговими коефіцієнтами для зменшення обсягу мережевого трафіку при частих і невеликих змінах даних;
- удосконалення методу ребалансування даних у розподілених базах даних шляхом застосування генетичних алгоритмів з елітарністю та адаптивним схрещенням;

- проектування та реалізація розподіленої системи на основі електронного онлайн-журналу для навчальних закладів з використанням розроблених методів, що дозволить провести експериментальні дослідження та оцінити ефективність запропонованих підходів у реальних умовах.

Об'єкт дослідження – процес виконання запитів у високонавантажених розподілених базах даних.

Предмет дослідження – методи підвищення ефективності виконання запитів у високонавантажених розподілених системах шляхом оптимізації мережевого трафіку, часу узгодженості даних та ребалансування даних.

Методи дослідження – теорія алгоритмів, методи оптимізації, теорія ймовірностей та математичної статистики, методи об'єктно-орієнтованого та процедурного програмування, емпіричні методи, комп'ютерне моделювання та експериментальне дослідження.

Наукова новизна отриманих результатів:

- Вперше розроблено новий метод мінімізації мережевого трафіку при виконанні запитів у розподілених базах даних на основі алгоритму консенсусу Raft. Завдяки попередньому обміну метаданими та подальшій матеріалізації результатів метод дозволяє суттєво знизити обсяг інформації, що передається між вузлами, тим самим прискорюючи виконання запитів у високонавантажених системах. На відміну від існуючих рішень, запропонований метод не передає повні набори даних, а оперує лише метаданими, тим самим зменшує час виконання запитів до 34.22%, що забезпечує більш ефективне використання мережевих ресурсів та підвищену продуктивність.
- Удосконалено метод узгодженості даних, застосувавши алгоритм Левенштейна з ваговими коефіцієнтами, що забезпечує більш точне визначення мінімальних змін між версіями даних. Такий підхід мінімізує передачу надлишкової інформації, передаючи лише необхідні зміни замість повних копій даних, знижуючи мережевий трафік та прискорюючи процес узгодження. Відмінно від стандартних методів, де часто передається велика кількість зайвої інформації, запропонована модифікація дозволяє точково

оновлювати дані, що особливо ефективно при частих та невеликих оновленнях тим самим зменшуючи середній час виконання запитів на 6.85%.

- Удосконалено метод ребалансування даних у розподілених базах даних, інтегрувавши генетичні алгоритми з елітарністю та адаптивним схрещенням. Завдяки динамічному підбору параметрів та збереженню найкращих рішень метод дозволяє ефективніше розподіляти дані між вузлами, знижуючи час обробки запитів до 8.2% та підвищуючи загальну продуктивність системи. На відміну від традиційних підходів, що зазвичай застосовують статичні або прості евристичні схеми, нова методика динамічно адаптується до умов системи, забезпечуючи кращу оптимізацію розподілу даних та стабільніші результати під високими навантаженнями.

Практичне значення отриманих результатів:

- реалізовано метод мінімізації мережевого трафіку в алгоритмі консенсусу Raft шляхом попереднього обміну метаданими між вузлами та подальшої матеріалізації отриманих результатів, метод впроваджено в систему на основі фреймворку Laravel із використанням розподіленої бази даних MySQL та кешування Redis, що дозволяє знизити навантаження на мережу та підвищити ефективність виконання запитів у високонавантажених розподілених системах;
- реалізовано модифікований метод узгодженості даних на основі алгоритму Левенштейна з ваговими коефіцієнтами, що дозволяє передавати лише необхідні дельти між версіями даних при частих та невеликих змінах, зменшуючи обсяг даних та покращуючи швидкодію синхронізації в розподілених системах, реалізація виконана за допомогою Observers та Jobs у фреймворку Laravel;
- реалізовано модифікований метод ребалансування даних з використанням генетичних алгоритмів з елітарністю та адаптивним схрещенням, що оптимізує розподіл даних між вузлами системи, зменшуючи час виконання запитів та підвищує продуктивність системи в умовах високих навантажень, реалізація методу здійснена у вигляді програмного модуля, інтегрованого в існуючу систему.

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:

- [6] – у статті розглянуто різноманітні типи баз даних та проаналізовано переваги, недоліки та вимоги до розподілених БД. Проведено аналіз підходів до оцінки ефективності розподілених БД за критеріями продуктивності, масштабованості, надійності та інших показників.
- [7] – у статті розглянуто оптимізацію використання алгоритму Raft в розподілених системах з базами даних. Висвітлено основні методи оптимізації, включаючи налаштування параметрів алгоритму, асинхронний ввід-вивід, кешування стану, розпаралелювання операцій, реплікацію даних, застосування технік компресії, оптимізацію мережевої взаємодії.
- [8] – у статті представлено новий метод мінімізації мережевого трафіку в алгоритмі консенсусу, схожому на Raft, для розподілених баз даних. Основою методу є попередній обмін ключовими векторами та кардинальностями між вузлами, що дозволяє зменшити обсяг передаваних даних шляхом уникнення дублювання та передачі лише необхідної інформації.
- [9] – у статті запропоновано метод для ідентифікації вхідних об'єктів через інтеграцію інформаційних просторів в єдиний проміжний інформаційний простір. Розглянуто процес формування єдиного простору на основі збору метаданих про об'єкти, що дозволяє зменшити обсяг переданих даних. Метод є основою методу оптимізації мережевого трафіку в Raft.
- [10] – у статті досліджено проблеми узгодженості даних у нереляційних базах даних NoSQL та наведено приклади оптимізації часу узгодженості для великих розподілених систем. Висвітлено концепцію NoSQL як перспективного підходу для зберігання великих обсягів даних та забезпечення швидкого доступу до них.

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

- VII International Scientific and Practical Conference «Education and science of today: intersectoral issues and development of sciences» (18.10.2024 року, Cambridge, UK);
- VII International Scientific and Practical Conference «Theoretical and empirical scientific research: concept and trends» (16.08.2024 року, Oxford, UK);
- V Міжнародна науково-практична конференція «Scientific researches and methods of their carrying out: world experience and domestic realities» (17.02.2023, Вінниця).
- V Міжнародна науково-практична конференція «An integrated approach to science modernization: methods, models and multidisciplinary» (23.12.2022, Вінниця).
- The 5th International scientific and practical conference “European scientific discussions”. (28-29.03.2021, Рим Італія).

Публікації. За результатами дисертаційних досліджень опубліковано 5 наукових публікацій, серед яких: 1 публікація, яка проіндексована в Scopus базі даних; – 4 статей у фахових наукових виданнях категорії «Б»;

Структура та обсяг роботи. Дисертація складається зі вступу, 5 розділів, висновків, списку використаних джерел з 41 найменувань та 2 додатків. Загальний обсяг роботи складає 152 сторінок, з них 126 сторінки основного тексту, 45 рисунків та 6 таблиць.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ГА – Генетичні алгоритми

ПЛС – Перетворення Лапласа-Стільтьєса

API – Application Programming Interface – прикладний програмний інтерфейс

BBO – Biogeography-Based Optimization - оптимізація на основі біогеографії

CDN – Content Delivery Network – географічно розподілена мережева інфраструктура

CRUD – Create, Read, Update, Delete – основні операції з базою даних

DBMS – Database Management System – система управління базами даних

DDBS – Distributed Database System – система розподілених баз даних

HTTP – Hypertext Transfer Protocol – протокол передачі даних

IoT – Internet of Things – інтернет речей

JSON – JavaScript Object Notation

MDAP – Multiprocessor Document Allocation Problem

MVC – Model-View-Controller

NoSQL – Non-SQL – нереляційний підхід збереження даних

RPC – Remote Procedure Call – віддалений виклик процедури

UML – Unified Modeling Language – уніфікована мова моделювання

UTC – Coordinated Universal Time – стандарт регулювання часу

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ ПІДВИЩЕННЯ ШВИДКОДІЇ ВИКОНАННЯ ЗАПИТІВ У ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ

1.1 Класифікація високонавантажених систем

Удосконалення технічних можливостей засобів обчислювальної техніки, розвиток комунікаційних засобів та технологій управління інформаційними ресурсами в останні роки призвели до появи більших інформаційних систем. Йдеться масштабах систем як щодо обсягу підтримуваних інформаційних ресурсів, а й кількості їх користувачів. З'явилися системи дуже великих баз даних, що підтримують багато гігабайтів і навіть петабайтів даних, системи текстового пошуку з великими колекціями документів, а також IoT-системи, які генерують і обробляють величезні обсяги даних з різноманітних сенсорів і пристроїв. Обсяг інформаційних ресурсів Web нині обчислюється багатьма мільйонами сторінок. Корпоративні системи баз даних налічують тисячі користувачів, відповідно кількість баз даних зростає також.

Сьогодні є чотири популярних типи баз даних. Кожен із них використовує різні типи моделі даних, що призводить до значних відмінностей між кожним типом NoSQL[2]:

- Базы даних документів;
- Графічні бази даних;
- Базы даних ключ-значення;
- Сховище з широким стовпчиком;

Базы даних документів. Ці бази даних також називаються сховищами документів. Ці бази даних зберігають наполовину структуровані дані та описи цих даних у форматі документа. Вони дозволяють розробникам створювати й оновлювати програми без посилання на головну схему. Використання баз даних документів зросло разом із використанням JavaScript та JSON, формату обміну даними, який набув широкого поширення серед розробників веб-додатків. Базы даних документів використовуються для керування вмістом та обробки даних

мобільних додатків, таких як платформи для ведення блогів, веб-аналітика та програми електронної комерції. Couchbase Server, CouchDB, MarkLogic і MongoDB є прикладами баз даних документів[1].

Графічні бази даних . Сховища даних графіків організовують дані як вузли, подібні до рядків у реляційній базі даних, і ребра, які представляють зв'язки між вузлами. Оскільки система графів зберігає зв'язки між вузлами, вона може підтримувати більш розширене представлення зв'язків між даними. Крім того, на відміну від реляційних моделей, які спираються на строгі схеми, модель даних графіка може розвиватися з часом і використанням. Бази даних графіків застосовуються в системах, які мають відображати відносини, наприклад, платформи соціальних медіа, системи бронювання або управління відносинами з клієнтами. Приклади графічних баз даних включають AllegroGraph, IBM Graph і Neo4j.

Бази даних 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 високо масштабованих і продуктивних додатків. Бази даних "ключ-значення" ідеально підходять для керування сесіями та кешування у веб-додатках, наприклад тих, які необхідні для керування деталями кошика для онлайн-покупців або для керування деталями сесії для багатокористувацьких ігор. Реалізації відрізняються тим, як вони орієнтовані на роботу з оперативною пам'яттю, твердо тільними або дисковими накопичувачами. Приклади популярних баз даних ключ-значення включають Aerospike, DynamoDB, Redis та Riak.

Сховище з широким стовпчиком. Ці бази даних використовують знайомі таблиці, стовпці та рядки, як таблиці реляційної бази даних, але імена стовпців і форматування можуть відрізнятися від рядка до рядка в одній таблиці. Кожен стовпець також зберігається окремо на диску. На відміну від традиційного сховища, орієнтованого на рядки, сховище з широкими стовпцями оптимально під час запиту даних за стовпцями. Типові програми, де магазини з широкими колонками можуть бути відмінними, включають механізми рекомендацій, каталоги, виявлення шахрайства та реєстрацію подій. Accumulo, Amazon

SimpleDB, Cassandra, HBase і Hypertable є прикладами магазинів із широкими стовпцями.

1.2 Технології виконання запитів у високонавантажених систем

Високонавантажені системи повинні забезпечувати швидке та надійне виконання запитів, незважаючи на кількість користувачів чи обсяг даних. У цьому розділі розглянемо сучасні технології та методи, які застосовуються для підвищення ефективності виконання запитів у таких системах. Окрім розподілених баз даних, що дозволяють зберігати та обробляти дані на декількох вузлах, що підвищує масштабованість та стійкість системи існують технології, що допомагають пришвидшити час виконання запитів:

- кешування;
- паралельна та розподілена обробка даних;
- CDN;
- мікросервісна архітектура.

Кешування є однією з найефективніших технологій для підвищення продуктивності високонавантажених систем. Воно дозволяє зменшити час доступу до даних та знизити навантаження на основні системні ресурси шляхом зберігання часто використовуваних або ресурсномістких даних у швидко доступній пам'яті. Це особливо актуально в умовах, коли кількість запитів до системи постійно зростає. Існують наступні типи кешування:

- кешування на стороні клієнта, Зберігання даних у браузері або на пристрої користувача для зменшення кількості запитів до сервера. Використовуються механізми кешування HTTP, такі як заголовки Expires, Cache-Control [3] .
- Кешування на стороні сервера де сервер зберігає результати обчислень або запитів для швидкого доступу при повторних зверненнях. Це може бути реалізовано як у пам'яті сервера, так і з використанням окремих сервісів кешування.

- Кешування на рівні бази даних, деякі СУБД мають вбудовані механізми кешування, які дозволяють зберігати результати запитів або індекси в пам'яті для швидшого доступу [1].

Паралельна та розподілена обробка даних є ключовими технологіями для забезпечення ефективного виконання запитів у високонавантажених системах. Вони дозволяють масштабувати обчислювальні потужності відповідно до зростаючих потреб, забезпечуючи швидкість, надійність та гнучкість систем. Правильне впровадження цих технологій вимагає розуміння їхніх принципів, викликів та найкращих практик.

Паралельна обробка даних передбачає одночасне виконання декількох процесів або потоків на одному або декількох процесорах в межах однієї системи. Це дозволяє ефективно використовувати багатоядерні процесори та багатопроцесорні системи [4].

Розподілена обробка даних включає виконання обчислень на декількох вузлах мережі, які можуть бути географічно розподіленими. Вузли взаємодіють між собою через мережеві протоколи, обмінюючись даними та результатами обчислень [5].

Використання CDN та географічно розподілених серверів є ключовим фактором у забезпеченні швидкого та надійного доступу до контенту у високонавантажених системах. Вони дозволяють зменшити затримки, розподілити навантаження та покращити загальний користувацький досвід. Правильне впровадження цих технологій вимагає розуміння їхніх принципів, врахування можливих викликів та дотримання найкращих практик.

Мікросервісна архітектура є потужним підходом для розробки високонавантажених систем, що дозволяє підвищити масштабованість, гнучкість та стійкість. Вона надає можливість швидко адаптуватися до змін, впроваджувати нові функції та ефективно розподіляти навантаження. Однак впровадження мікросервісів вимагає ретельного планування, розуміння можливих викликів та використання відповідних інструментів та практик.

.3 Аналіз розподілених баз даних у високонавантажених системах

1.3.1 Централізовані та розподілені бази даних

Централізований підхід, як показано на рис. 1.1, добре підходив для задоволення структурованих інформаційних потреб корпорацій, але виявився недостатнім, коли події, що швидко змінювалися, вимагали швидшого часу реагування та настільки ж швидкого доступу до інформації. У динамічному середовищі повільний процес від запиту інформації до її затвердження та передачі спеціалісту й кінцевому користувачеві просто не відповідає потребам осіб, що приймають рішення. Необхідним був швидкий, неструктурований доступ до баз даних, з використанням довільних запитів для генерації інформації на місці.

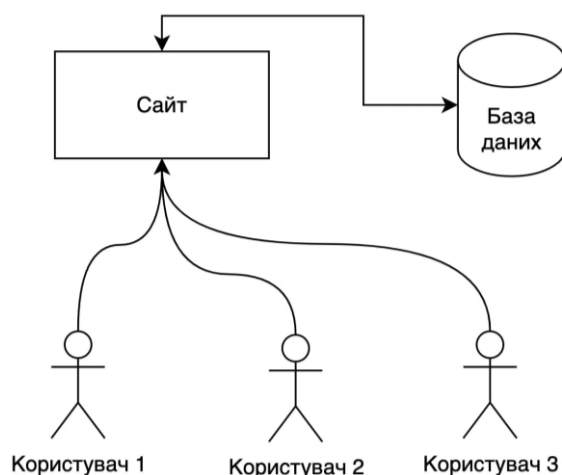


Рис. 1.1 Централізований підхід БД

Для того, щоб будь-яка організація функціонувала належним чином, необхідна добре підтримувана система баз даних, яка дозволяє автоматизувати бізнес-процеси та полегшує зберігання даних, з'явилися розподілені бази даних, які показали високий рівень продуктивності та ефективності в обробці та управлінні даними в межах великих організацій. Впровадження розподілених баз даних стало необхідним у зв'язку зі збільшенням глобалізації, і організації почали розширюватися по всьому світу.

Розподілена база даних (Distributed Database System) – це просто колекція кількох взаємопов'язаних баз даних, які розподілені по комп'ютерній мережі (рис. 1.2). Це реляційна схема, яка сприяє децентралізації баз даних для управління та маніпулювання даними в базі даних за допомогою однієї або спільної мови.

Розподілена база даних є єдиною логічною базою даних, яка фізично охоплює кілька комп'ютерів у різних місцях і з'єднана через канали зв'язку.

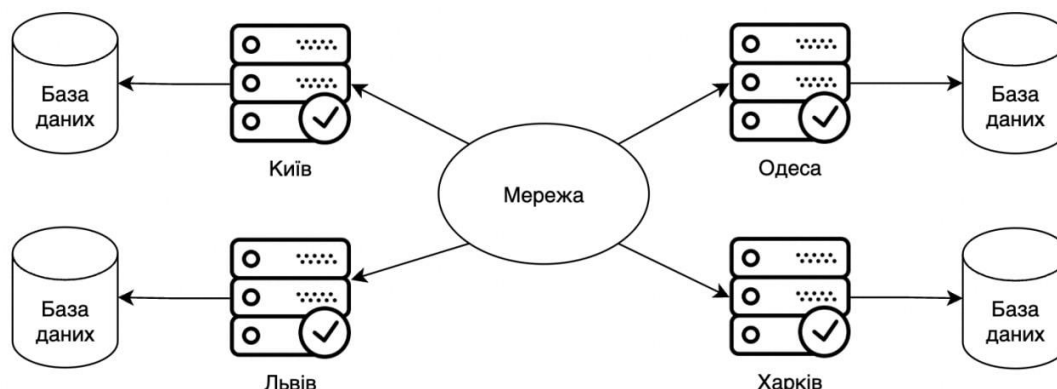


Рис. 1.2 Приклад роботи розподіленої бази даних

Це просто віртуальна база даних, компоненти якої фізично зберігаються в декількох окремих реальних базах даних в окремих місцях, що дозволяє користувачам отримувати доступ до даних будь-де в мережі. Згідно з дослідженнями, багато організацій мотивовані впровадженням ефективних систем розподілених баз даних з метою забезпечення масштабованості.

До основних компонентів системи розподіленої бази даних можна віднести програмне забезпечення, яке відповідає за управління сегментом розподіленої бази даних на кожному вузлі системи. Відповідає за обробку запитів до бази даних, забезпечення цілісності даних, управління транзакціями та виконання запитів.

- Менеджер розподілених транзакцій
- Система управління даними
- Інтерфейс запитів користувача
- Комунікаційна підсистема
- Storage Distributed Data Catalog

Порівнюючи розподілені бази даних з централізованими системами, можна побачити, що розподілені бази даних мають численні переваги:

- надійність та доступність - централізована система виходить з ладу, база даних стає недоступною для всіх користувачів. Проте розподілена система продовжує функціонувати на певному зниженому рівні навіть при відмові одного з компонентів. Наприклад, якщо дані та програмне забезпечення

DBMS розподілені на кілька вузлів, і якщо один вузол виходить з ладу, інші вузли продовжують працювати нормально, і доступ до даних на пошкоджені вузли залишається можливим. Це покращує надійність та доступність системи баз даних.

- масштабованість - розширення розподіленого середовища є набагато легшим у порівнянні з іншими системами, зокрема додаванням нових даних, збільшенням розміру бази даних та додаванням обчислювальної потужності.
- оптимізація витрат - дані в розподіленій системі можуть бути розташовані ближче до місця їх використання, що може знизити витрати на комунікацію порівняно з центральною системою.
- підтримка - розподіл даних заохочує місцеві групи до більшого контролю над своїми даними, що сприяє покращенню цілісності даних та адмініструванню. Користувачі можуть отримувати доступ до не місцевих даних, коли це необхідно.
- час відгуку - у розподіленій базі даних запити на дані від користувачів певного вузла можуть задовольнятися даними, які зберігаються на цьому вузлі. Це прискорює обробку запитів, оскільки зменшуються затримки комунікацій та центрального комп'ютера.

Для розподілених систем, на відміну від централізованої системи даних, дані розподіляються між різними базами даних системи організації. Ці системи баз даних з'єднані одна з одною за допомогою зв'язку посилання. Такі посилання допомагають кінцевим користувачам легко отримати доступ до даних. Є два типи розподілених баз даних: гомогенні та гетерогенні DDBS.

Гомогенні розподілені бази даних є такими, де всі вузли системи працюють на одній операційній системі, використовують однакові прикладні програми та однакове апаратне забезпечення, що забезпечує узгодженість та сумісність між різними компонентами системи. У такій конфігурації всі вузли мають однакові параметри налаштування, що полегшує процес управління та обслуговування системи, а також спрощує виявлення та усунення можливих проблем.

Кожен вузол у гомогенній розподіленій базі даних використовує одну й ту ж систему управління базами даних, що дозволяє забезпечити єдині стандарти для

збереження, обробки та доступу до даних у всій системі. Це дозволяє уникнути проблем, пов'язаних з різними форматами даних або несумісними версіями програмного забезпечення, що може бути значущим фактором при масштабуванні системи або інтеграції нових вузлів. Гомогенні розподілені бази даних можуть бути поділені на два основних типи, кожен з яких має свої особливості та переваги:

1. Автономні гомогенні розподілені бази даних (рис. 1.3) - в такій конфігурації кожна DBMS на окремому вузлі працює незалежно від інших.

2. Неавтономні гомогенні розподілені бази даних - у цьому випадку існує центральна або головна DBMS, яка координує всі процеси, пов'язані з доступом до даних і їх оновленням на всіх вузлах системи.

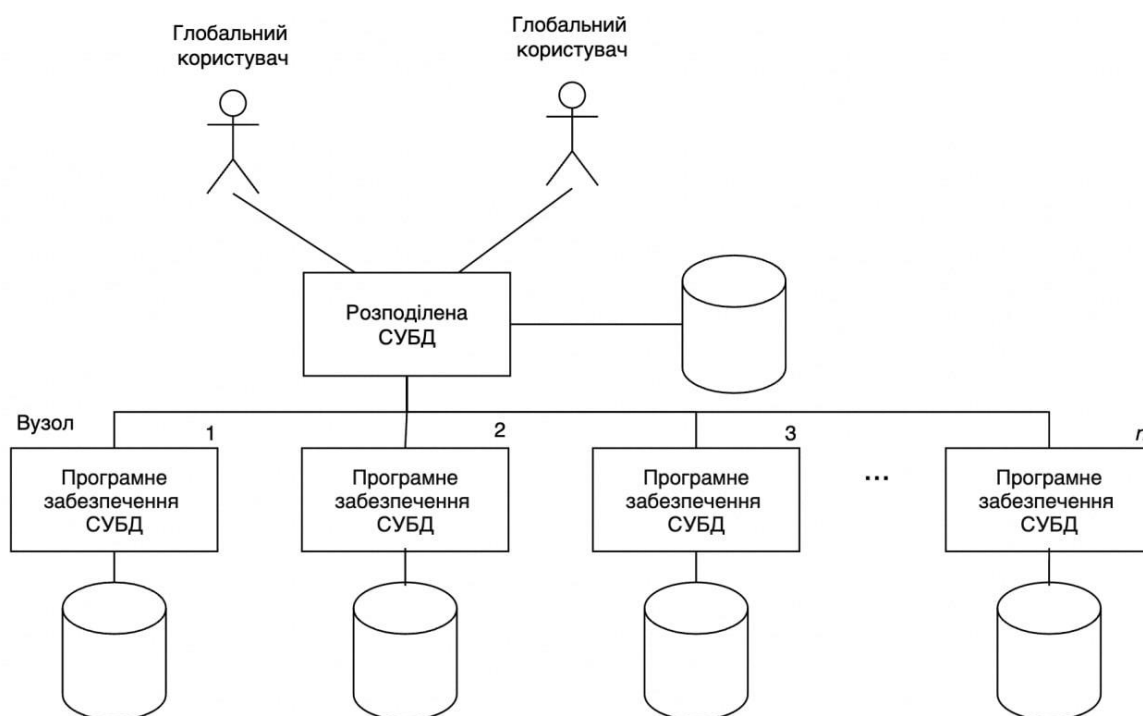


Рис. 1.3 Середовище гомогенної розподіленої бази даних

Гетерогенні розподілені бази даних є такими, де вузли системи працюють на різних операційних системах, використовують різні прикладні програми та різне апаратне забезпечення. Це означає, що в такій системі різні компоненти можуть значно відрізнятися один від одного за своїми технічними характеристиками, що створює додаткові виклики для забезпечення сумісності та взаємодії між ними.

У середовищі гетерогенної розподіленої бази даних (рис. 1.4), різні вузли можуть використовувати різні системи управління базами даних (DBMS), як реляційні, так і нереляційні.

Це означає, що дані можуть зберігатися в різних форматах і структурах, і необхідно забезпечити відповідні механізми для їх інтеграції та узгодження. Наприклад, один вузол може використовувати реляційну DBMS, таку як MySQL або Oracle, для зберігання даних у вигляді таблиць, тоді як інший вузол може використовувати нереляційну DBMS, таку як MongoDB або Cassandra, для зберігання даних у вигляді документів або графів.

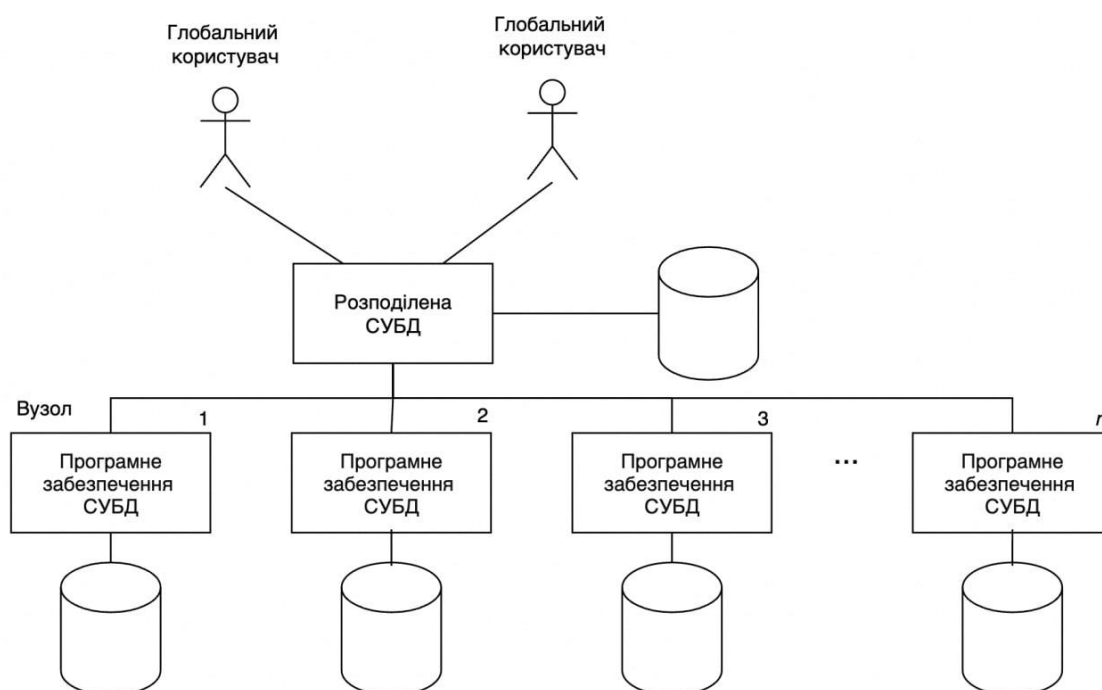


Рис. 1.4 Середовище гетерогенної розподіленої бази даних

У сучасних інформаційних системах розподілені бази даних відіграють ключову роль, забезпечуючи зберігання та обробку величезних обсягів даних у різних географічно розподілених вузлах. Їхня основна мета полягає в наданні високої доступності та продуктивності системи, зберігаючи при цьому цілісність даних та забезпечуючи стійкість до збоїв мережі. Проте, керування такими системами стикається з численними викликами, пов'язаними з необхідністю забезпечення узгодженості, доступності та стійкості до розділень у випадку мережних збоїв.

1.3.2 Проєктування розподілених баз даних

Проєктування розподіленої комп'ютерної системи включає ухвалення рішень щодо розміщення даних і програм на різних вузлах комп'ютерної мережі, а також, можливо, проєктування самої мережі[1]. У випадку розподілених систем управління базами даних, розподіл застосунків включає два аспекти: розподіл програмного забезпечення розподіленої DBMS та розподіл програм, які на ньому виконуються. Було запропоновано[11], що організацію розподілених систем можна досліджувати за трьома ортогональними вимірами:

- *Рівень спільного використання*: відсутність спільного використання, спільне використання даних, або спільне використання як даних, так і програм.
- *Поведінка шаблонів доступу*: може бути статичною або динамічною, залежно від того, як змінюються шаблони доступу до даних у часі.
- *Рівень знань про поведінку шаблонів доступу*: може бути повним або частковим, залежно від того, наскільки добре відомі шаблони доступу до даних.

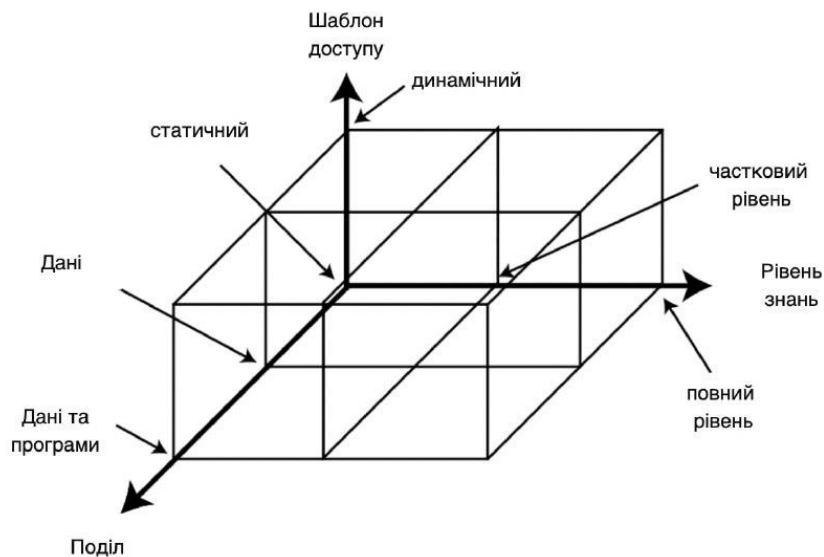


Рис. 1.5 Структура розподілу проєктування

Щодо рівня спільного використання, є три можливості. По-перше, є відсутність спільного використання - кожен застосунок і його дані виконуються на одному вузлі, і немає жодної комунікації з будь-якою іншою програмою або доступу до будь-якого файлу даних на інших вузлах. Це характеризує дуже ранні дні мережевих технологій і, ймовірно, сьогодні не дуже поширене. Далі ми

знаходимо рівень спільного використання даних - всі програми дублюються на всіх вузлах, але файли даних — ні. Відповідно, запити користувачів обробляються на вузлі, де вони виникають, а необхідні файли даних переміщуються мережею. Нарешті, у спільному використанні даних і програм як дані, так і програми можуть бути спільно використовувані, що означає, що програма на певному вузлі може запитувати послугу від іншої програми на іншому вузлі, яка, у свою чергу, можливо, має отримати доступ до файлу даних, що знаходиться на третьому вузлі.

Другий вимір поведінки шаблонів доступу можна ідентифікувати за двома альтернативами. Шаблони доступу користувачів можуть бути статичними, тобто вони не змінюються з часом, або динамічними. Очевидно, що набагато легше планувати і керувати статичними середовищами, ніж у випадку динамічних розподілених систем. На жаль, важко знайти багато реальних розподілених застосувань, які можна було б класифікувати як статичні. Отже, важливим питанням є не те, чи є система статичною або динамічною, а наскільки вона динамічна.

Третій вимір класифікації стосується рівня знань про поведінку шаблонів доступу. Одна можливість полягає в тому, що проєктувальники не мають жодної інформації про те, як користувачі отримуватимуть доступ до бази даних. Це теоретично можливо, але дуже важко, якщо не неможливо, спроектувати розподілену DBMS, яка може ефективно впоратися з такою ситуацією. Більш практичні альтернативи полягають у тому, що проєктувальники мають повну інформацію, коли шаблони доступу можна досить точно передбачити і вони не відхиляються суттєво від цих передбачень, або часткову інформацію, коли є відхилення від передбачень.

Проблема проєктування розподіленої бази даних повинна розглядатися в рамках цього загального підходу. У всіх випадках у розподіленому середовищі виникають нові проблеми, які не мають відношення до централізованого середовища.

1.4 Основні технології розподілених баз даних, що впливають на швидкодію запитів

1.4.1 Фрагментація даних у розподілених системах

Основне завдання проєктування розподілених баз даних – це ребалансування відносин у випадку реляційної бази даних або класів у випадку об'єктно-орієнтованих баз даних, розподіл і реплікація фрагментів на різних сайтах розподіленої системи, а також локальна оптимізація на кожному сайті [12]. Фрагментація – це метод проєктування, який дозволяє розділити одну відношення або клас бази даних на дві або більше частини таким чином, щоб комбінація цих частин забезпечувала оригінальну базу даних без втрати інформації (рис 1.6).

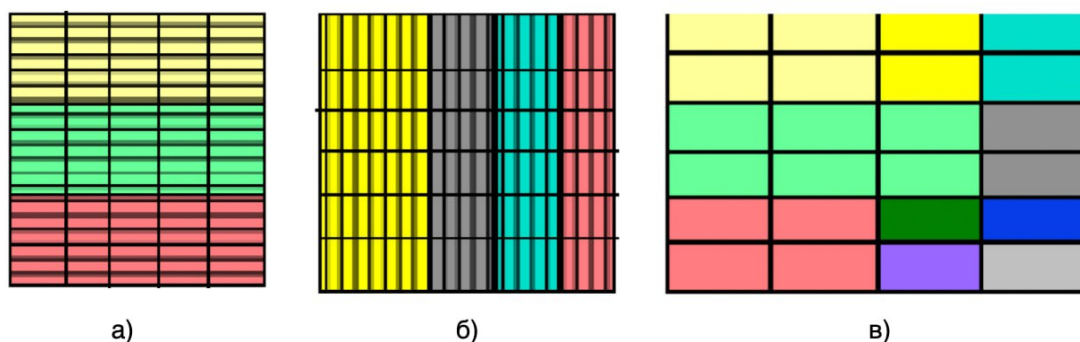


Рис. 1.6 Типи фрагментацій

Це зменшує обсяг нерелевантних даних, які доступні базам даних, тим самим зменшуючи кількість дискових доступів. Фрагментація за типами може бути

- горизонтальною,
- вертикальною
- змішаною/гібридною.

Горизонтальна фрагментація - при даному варіанті відбувається розподіл рядків таблиці чи відношення між різними вузлами (рис 1.6а). Це означає, що кожен вузол отримує певну підмножину рядків таблиці. Фрагментація у такому випадку може включати перенесення частини рядків з одного вузла на інший, щоб досягти рівномірного розподілу навантаження. Наприклад: Якщо є система яка працює у різних регіонах, то дані можна розділити по вузлах відповідно до регіонів, вузол який обслуговує Київ, він може зберігати дані студентів та груп, що належать до шкіл Києва.

Вертикальна фрагментація - даний варіант передбачає розподіл стовпців таблиці між вузлами (рис 1.6б). У такому випадку кожен вузол зберігає підмножину атрибутів таблиці. Фрагментація полягає у переміщенні стовпців між вузлами. Наприклад: Якщо у вас є таблиця з інформацією про студентів, яка містить такі атрибути, як ім'я, дата народження, поточна оцінка, та історія оцінок, вертикальне ребалансування може передбачати перенесення атрибута “історія оцінок” на інший вузол. Це може бути корисним, якщо запити, які часто виконуються, в основному використовують лише поточні оцінки студентів. Таким чином, основний вузол буде відповідати за обробку запитів, які стосуються імені, дати народження і поточної оцінки, а запити, що вимагають доступу до історії оцінок, будуть перенаправлені на інший вузол.

Змішана (гібридна) фрагментація - є поєднанням горизонтальної та вертикальної фрагментації, що дозволяє використовувати переваги обох підходів для досягнення максимального рівня оптимізації розподілу даних у розподілених базах даних (рис 1.6в).

Це дозволяє досягти більш гнучкого та ефективного розподілу даних між вузлами системи.

1.4.2 Ребалансування даних у розподілених системах

У процесі функціонування бази даних відбуваються постійні зміни, що вимагають адаптації ресурсів системи. Зокрема, зі збільшенням пропускну здатності запитів виникає потреба у додаткових процесорних потужностях для обробки зростаючого навантаження. Збільшення розміру набору даних обумовлює необхідність у додаткових дискових ресурсах та оперативній пам'яті для забезпечення ефективного зберігання. Крім того, у разі відмови одного з вузлів, інші вузли системи повинні взяти на себе його функції, що також вимагає зміни конфігурації. Ці зміни потребують перенесення даних та обчислювальних процесів з одного вузла на інший, що реалізується через процес ребалансування. Ребалансування — це процес перенаправлення навантаження (зберігання даних, запити на читання та запис) між вузлами кластера для забезпечення рівномірного розподілу ресурсів [13].

Процес ребалансування, незалежно від підходів та схем розділення даних, має відповідати кільком ключовим вимогам:

- Після завершення ребалансування навантаження повинно бути рівномірно розподілене між вузлами в кластері, що забезпечує справедливий доступ до ресурсів.
- Під час процесу ребалансування база даних повинна залишатися доступною для операцій читання та запису, що забезпечує безперервність обслуговування користувачів.
- Обсяг даних, який переміщується між вузлами під час ребалансування, повинен бути мінімальним, щоб процес був швидким та не створював додаткового навантаження на мережу та підсистему введення/виведення.

Таким чином, процес ребалансування є критично важливим для підтримання продуктивності та надійності розподіленої бази даних у динамічному середовищі. З плином часу, бази даних зазнають різноманітних змін, що вимагають адаптації системи для забезпечення стабільної роботи та відповідності зростаючим вимогам. Деякі з цих змін включають:

- Збільшення пропускної здатності запитів - з підвищенням кількості запитів виникає потреба в додатковій обчислювальній потужності. Це зазвичай досягається шляхом додавання процесорів, що дозволяє більш ефективно обробляти навантаження.
- Зростання розміру набору даних - збільшення обсягів даних вимагає розширення систем зберігання, що передбачає додавання нових жорстких дисків або твердотільних накопичувачів, а також збільшення обсягу оперативної пам'яті для підтримки швидкого доступу до даних
- Вихід з ладу обладнання - у випадку відмови однієї з машин, інші вузли в кластері повинні автоматично взяти на себе функції несправної машини, щоб забезпечити безперервність роботи.

Для забезпечення цих змін у кластері необхідно виконати ребалансування. Ребалансування, незалежно від використовуваної схеми розділення даних, має відповідати кільком ключовим вимогам:

- Справедливий розподіл навантаження - після завершення ребалансування, навантаження у вигляді зберігання даних та обробки запитів повинно рівномірно розподілятися між усіма вузлами.
- Безперервність обслуговування - під час процесу ребалансування, база даних повинна продовжувати приймати запити на читання та запис без суттєвих збоїв чи затримок.
- Ефективність процесу - для того, щоб процес ребалансування був максимально швидким та не створював надмірного навантаження на мережу обсяг даних, що переміщується між вузлами, повинен бути мінімально необхідним.

Забезпечення цих умов дозволяє підтримувати високу продуктивність бази даних навіть в умовах зміни навантаження та конфігурації системи. Існують наступні стратегії ребалансування [14]

- статичне ребалансування;
- динамічне ребалансування;
- ребалансування на основі хешування;
- ребалансування на основі діапазонів;
- гібридне ребалансування;

Статичне ребалансування в контексті розподілених баз даних передбачає фіксований, заздалегідь визначений розподіл даних між вузлами системи, який не змінюється протягом роботи системи, якщо не відбувається ручне втручання адміністратора або значні зміни в топології або структурі даних. Переваги статичного ребалансування:

- простота розгортання та управління;
- прогнозованість продуктивності в умовах стабільного навантаження;
- відсутність необхідності в складному програмному забезпеченні для моніторингу та автоматичного перерозподілу даних.

Недоліки статичного ребалансування:

- неможливість автоматичного реагування на зміни в навантаженні або структурі даних;

- потенційна неефективність в умовах нерівномірного розподілу запитів або даних;
- можливість перевантаження окремих вузлів і зниження загальної продуктивності системи.

Динамічне ребалансування є підходом, який передбачає автоматичний перерозподіл даних між вузлами розподіленої бази даних у реальному часі, залежно від змін в навантаженні на систему, обсязі даних, частоті запитів, або інших параметрах. Цей підхід дозволяє системі адаптуватися до змін, забезпечуючи оптимальне використання ресурсів і підвищуючи загальну продуктивність. Переваги динамічного ребалансування:

- адаптивність, можливість автоматично адаптуватися до змін у навантаженні та структурі даних;
- оптимізація продуктивності, а саме зменшення затримок і підвищення продуктивності завдяки більш рівномірному розподілу навантаження;
- гнучкість та легке додавання або видалення вузлів без потреби в ручному перенесенні даних;
- масштабованість: Забезпечення стабільної роботи системи при зростанні обсягу даних або кількості запитів.

Недоліки динамічного ребалансування:

- потребує складного програмного забезпечення для моніторингу та автоматизації процесу ребалансування;
- динамічне ребалансування може вимагати додаткових ресурсів для постійного моніторингу та переносу даних;
- у випадку великої кількості одночасних змін можуть виникати колізії, що потребують додаткового управління.

Ребалансування на основі хешування є однією з популярних стратегій для розподілу даних у розподілених базах даних. Цей підхід дозволяє рівномірно розподіляти дані між вузлами системи, використовуючи хеш-функцію для визначення, на якому вузлі повинні зберігатися певні дані. Завдяки цьому забезпечується балансування навантаження та ефективне використання ресурсів у розподілених системах. Переваги ребалансування на основі хешування:

- простота реалізації - хешування є простим і ефективним методом розподілу даних, який легко реалізувати та підтримувати. Це робить його популярним вибором для багатьох розподілених систем;
- ефективне використання ресурсів - завдяки рівномірному розподілу даних між вузлами, забезпечується баланс навантаження на всі вузли системи, що дозволяє уникнути ситуацій, коли деякі вузли перевантажені, а інші залишаються недовантаженими;
- масштабованість - система, що використовує хешування для розподілу даних, легко масштабувати, додавання нових вузлів не вимагає значного перенесення даних, що знижує витрати на ребалансування і дозволяє швидко розширювати систему;
- зменшення ризику колізій - добре підібрана хеш-функція знижує ймовірність колізій, що забезпечує уникнення конфліктів при зберіганні даних і підвищує надійність системи.

Недоліки ребалансування на основі хешування:

- Можливість колізій - незважаючи на низьку ймовірність, колізії можуть виникати, коли дві різні ключі отримують одне і те ж хеш-значення. Це може призвести до потреби у додаткових перевірках і механізмах для уникнення конфліктів.
- Обмеженість у гнучкості - хешування на основі фіксованої хеш-функції може бути менш гнучким, коли йдеться про специфічні вимоги до розподілу даних. Наприклад, може бути важко реалізувати сценарії, де потрібен більш складний підхід до розподілу даних або адаптації до динамічних змін у використанні системи.
- Проблеми з гарячими точками (hot spots) - якщо певні ключі стають значно популярнішими за інші, це може призвести до перевантаження вузлів, які зберігають ці дані, навіть за умови використання хешування.

Ребалансування на основі діапазонів (Range Partitioning) є підходом до розподілу даних у розподілених базах даних, при якому дані розбиваються на окремі діапазони значень, кожен з яких зберігається на певному вузлі системи. Цей

підхід дозволяє ефективно керувати великими обсягами даних, оптимізуючи продуктивність системи і забезпечуючи збалансоване навантаження на вузли.

Гібридне ребалансування: Поєднує елементи кількох підходів, наприклад, хешування і діапазонів, для досягнення більшої гнучкості та оптимізації

1.4.3 Реплікація даних у розподілених системах

У розподілених системах, де дані та програми спільно використовуються на різних вузлах, реплікація є необхідним механізмом для забезпечення доступності та надійності даних [14]. Реплікація даних є критично важливим механізмом у розподілених базах даних, що полягає у створенні та підтриманні кількох копій одного й того ж набору даних на різних вузлах системи [15].

Основна мета реплікації полягає в забезпеченні:

- надійності;
- підвищенні продуктивності;
- масштабованості системи.

Однією з основних переваг реплікації є підвищення доступності системи. В умовах розподілених баз даних, які використовують реплікацію, зникає загроза наявності єдиної точки відмови. Завдяки розміщенню кількох копій даних на різних вузлах, навіть у разі виходу з ладу одного або декількох вузлів, інші вузли можуть продовжувати обслуговувати запити, забезпечуючи безперервний доступ до даних. Це особливо важливо для систем, що працюють у реальному часі або мають критичне значення для бізнесу, де простої можуть призвести до значних фінансових втрат або інших негативних наслідків.

Ще одним важливим аспектом реплікації є підвищення продуктивності системи. У розподілених базах даних одним з основних факторів, що впливають на час відповіді, є затримки, спричинені передачею даних між вузлами. Реплікація дозволяє розташовувати копії даних ближче до місць, де вони найчастіше використовуються, що значно зменшує затримки та підвищує швидкість доступу до даних. Це особливо актуально для географічно розподілених систем, де фізична відстань між вузлами може бути значною.

Масштабованість є ще одним важливим аспектом, який забезпечується завдяки реплікації [16]. У міру зростання системи як у географічному плані, так і за кількістю вузлів, реплікація дозволяє підтримувати високий рівень обслуговування запитів навіть при збільшенні навантаження на систему. Це досягається шляхом того, що реплікація дозволяє ефективно розподіляти запити між різними вузлами, де зберігаються копії даних, тим самим зменшуючи навантаження на окремі вузли і забезпечуючи рівномірний розподіл ресурсів.

Важливо зазначити, що вимоги до застосунків також можуть диктувати необхідність використання реплікації. Деякі додатки вимагають наявності кількох копій даних у різних місцях для виконання своїх операційних завдань.

Існують наступні типи реплікацій:

- реплікація Master-Slave;
- реплікація Master-Master;
- реплікація знімків;
- реплікація транзакцій;
- реплікація злиттям.

Реплікація типу “головний-підлеглий” – це процес, який забезпечує копіювання та синхронізацію даних з основної бази даних на одну або більше вторинних баз даних (рис. 1.7). У цьому випадку головна база даних виконує всі операції запису, такі як вставка, оновлення або видалення даних. Зміни, внесені до головної бази даних, автоматично передаються до підлеглих баз даних, які підтримують актуальну копію цих даних. Такий підхід дозволяє розвантажити основну базу та підвищити доступність даних для читання.

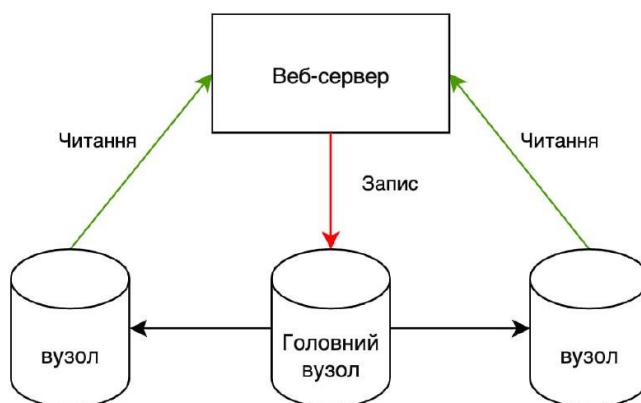


Рис. 1.7 Реплікація типу “головний-підлеглий”

Реплікація “головний-головний”, також відома як двонаправлена реплікація (рис. 1.8), передбачає налаштування, де дві або більше баз даних працюють як основні. У цьому випадку кожна з цих баз даних може приймати операції запису, що означає, що будь-які зміни, внесені в одну базу, автоматично синхронізуються з іншими. Цей двонаправлений зв’язок дозволяє підвищити доступність даних, але вимагає механізмів вирішення конфліктів у разі, якщо одночасно в різних базах даних виникають суперечливі зміни.

Коли в одній з головних баз даних здійснюється операція запису, ця зміна синхронізується з усіма іншими головними базами, щоб забезпечити узгодженість і актуальність даних у всій системі. Однак двонаправлена природа цієї реплікації може призводити до виникнення конфліктів, коли зміни вносяться в різних базах одночасно. Для підтримки цілісності та консистентності даних необхідні спеціальні механізми вирішення конфліктів, які можуть включати пріоритизацію змін, автоматичне виправлення або залучення адміністратора для ручного вирішення конфліктів.

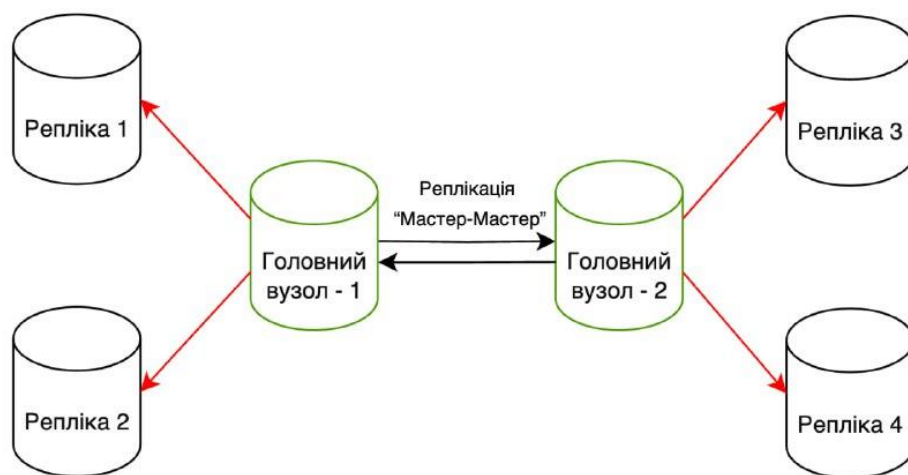


Рис. 1.8 Реплікація типу “головний-головний”

Знімкова реплікація, відома як snapshot replication, є технологією в базах даних (рис. 1.9), яка забезпечує створення точного дублікату всієї бази даних на конкретний момент часу. Цей знімок потім переноситься на один або кілька цільових серверів для подальшого використання. Такий підхід часто застосовується для забезпечення резервного копіювання, створення звітів або реалізації розподілених баз даних. Завдяки знімковій реплікації можна отримати

актуальну копію даних, що відображає стан системи в конкретний момент часу, що є важливим для підтримки цілісності та відновлюваності інформації.

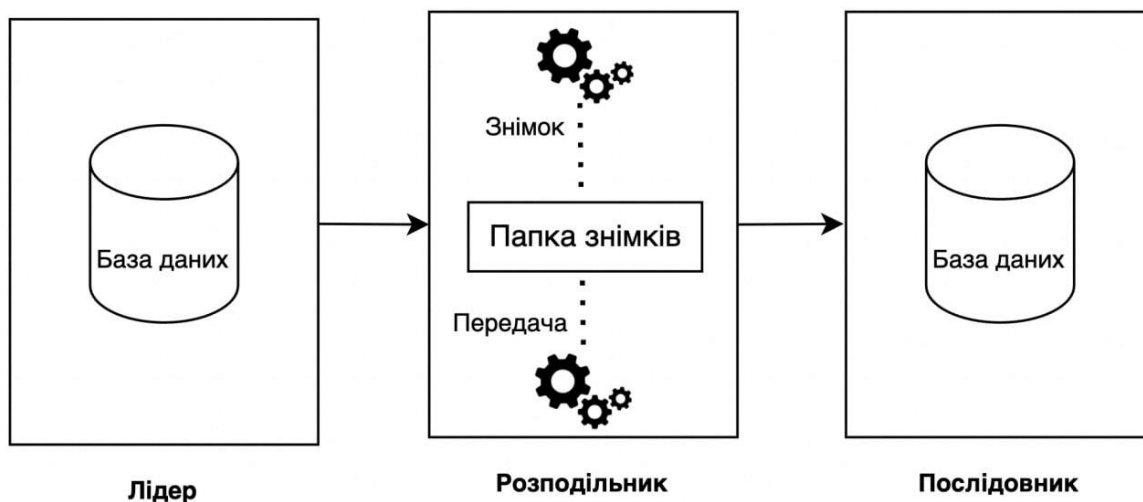


Рис. 1.9 Знімова реплікація

Реплікація транзакцій є однією з найважливіших методологій у сфері розподілених систем керування базами даних, оскільки вона забезпечує синхронізацію кількох копій даних у режимі реального часу (рис. 1.10). Зокрема, таке рішення дозволяє основній базі даних, відомій як “лідер”, відстежувати операції вставки, оновлення та видалення, що виконуються в певних таблицях або наборах таблиць. Усі відповідні зміни фіксуються й негайно передаються базам даних-передплатникам, унаслідок чого всі копії зберігають однаковий стан і залишаються синхронізованими. Завдяки цій безперервній передачі даних досягається висока консистентність інформації в масштабі цілої мережі, незалежно від географічного розташування окремих вузлів.

Транзакційна реплікація критично важлива для систем, де цілісність і узгодженість даних мають визначальне значення, наприклад у банківському середовищі або телекомунікаційних додатках. Найменші розходження в транзакціях можуть призвести до серйозних наслідків, тому механізм оперативної синхронізації допомагає уникнути потенційних помилок і мінімізувати затримки. Крім того, постійне копіювання змін у розподіленій системі підвищує відмовостійкість: у разі збою одного з вузлів інші можуть продовжувати роботу без переривання бізнес-процесів.

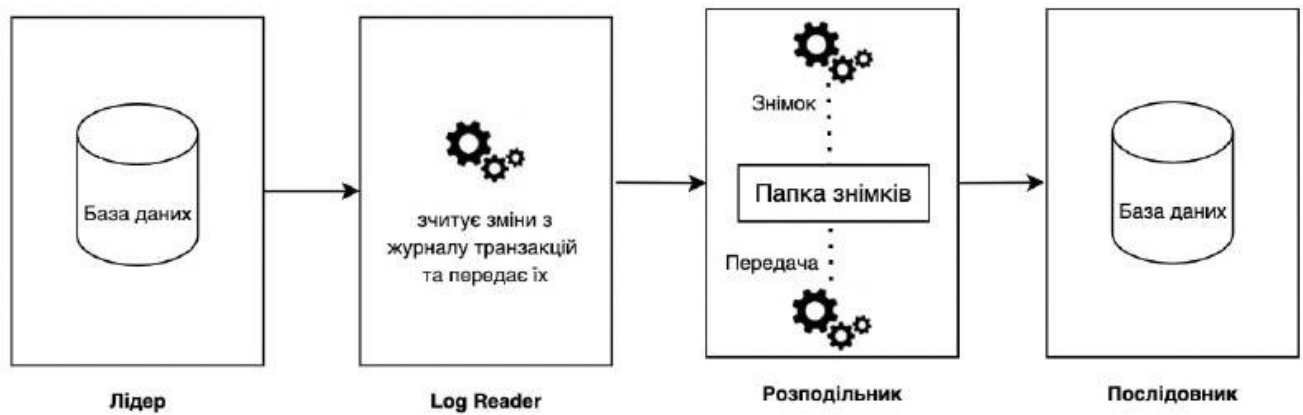


Рис. 1.10 Реплікація транзакцій

Реплікація злиттям — це метод синхронізації бази даних, який дозволяє як центральному серверу, так і підключеним до нього пристроям вносити зміни в дані, вирішуючи конфлікти, коли це необхідно (рис. 1.11).

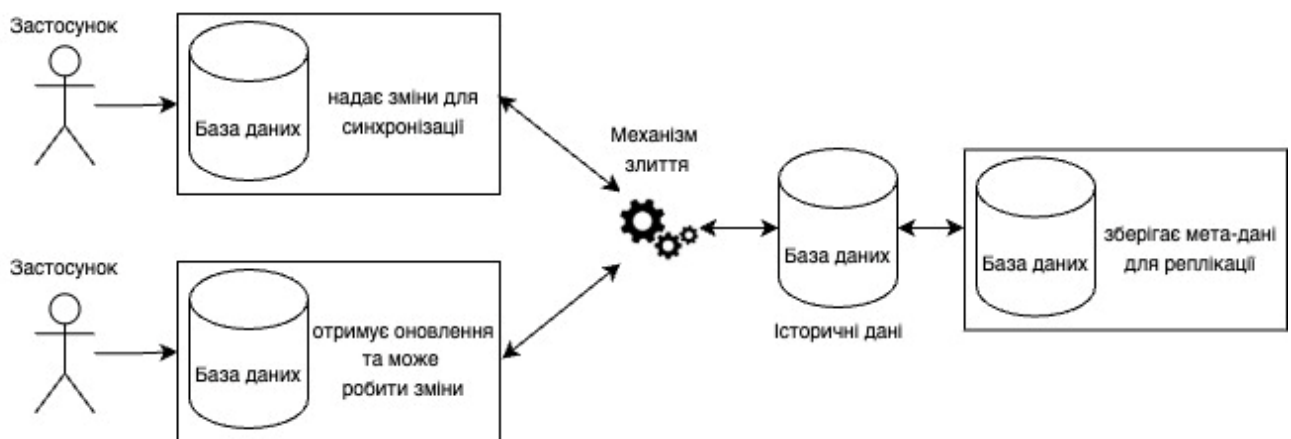


Рис. 1.11 Реплікація злиттям

Сильна та слабка взаємна узгодженість. Сильна узгодженість вимагає, щоб усі копії даних мали однакові значення після завершення оновлюючої транзакції. Слабка узгодженість допускає тимчасові розбіжності між копіями, але гарантує, що вони зрештою узгодяться (наприклад, концепція «eventual consistency»).

1.4.4 Узгодженість даних у розподілених системах

Після розгляду методів реплікації у розподілених базах даних, важливо зосередити увагу на ключовій концепції, яка безпосередньо впливає на ефективність та надійність цих методів, — узгодженості даних. Реплікація

забезпечує фізичну доступність даних на різних вузлах системи, однак саме узгодженість гарантує, що всі користувачі системи працюють з актуальною та однорідною інформацією. У контексті розподілених систем, узгодженість є критичною властивістю, яка визначає, наскільки синхронізованими є дані між різними вузлами після виконання операцій зчитування та запису.

Узгодженість відіграє центральну роль у забезпеченні коректності роботи розподілених систем, оскільки різні моделі узгодженості по-різному впливають на поведінку системи у випадку одночасного виконання множинних операцій [17]. Вибір відповідної моделі узгодженості залежить від архітектурних та функціональних вимог до системи, і саме він визначає баланс між продуктивністю, доступністю та надійністю даних. Узгодженість є критично важливою в ситуаціях, де навіть найменша розбіжність у даних може призвести до серйозних наслідків. Одним з найбільш показових прикладів є банківські системи та різні системи контролю. Розрізняють такі види узгодженості:

- суворая узгодженість;
- послідовна узгодженість;
- кінцева узгодженість;
- причинно-наслідкова узгодженість.

У моделі суворої узгодженості передбачається, що всі операції виконуються у певному глобальному порядку, незалежно від розташування вузлів у системі (рис.1.12). Це означає, що кожна операція читання завжди повертає найсвіжіше значення, яке було записано до того самого елемента даних, незалежно від того, на якому вузлі вона виконується. Такий підхід забезпечує найвищий рівень гарантії узгодженості, оскільки користувачі завжди працюють з актуальними та узгодженими даними. Однак це може негативно вплинути на продуктивність системи через значні накладні витрати на синхронізацію між вузлами, особливо у великих та географічно розподілених системах.

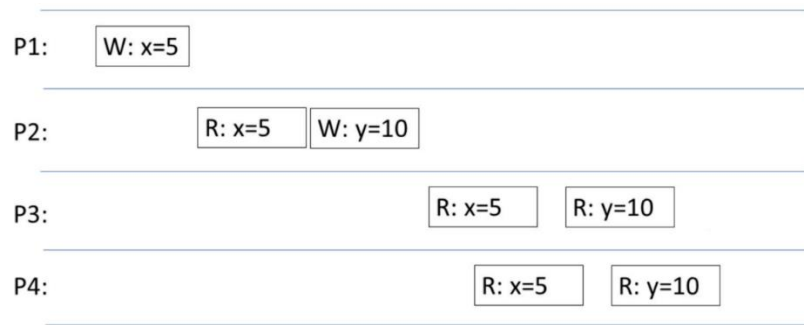


Рис. 1.12 Приклад суворої узгодженості

Послідовна узгодженість є однією з важливих моделей узгодженості, що використовується у розподілених системах для забезпечення коректного порядку виконання операцій з даними. Ця модель надає гарантії щодо порядку операцій, дозволяючи системам підтримувати узгодженість, не вимагаючи жорсткого глобального синхронного виконання всіх операцій, що є ключовим для забезпечення високої продуктивності та масштабованості. Послідовна узгодженість гарантує (рис. 1.13), що всі операції у системі будуть виконані в такому порядку, в якому вони були подані процесами, але цей порядок не обов'язково є глобально узгодженим з реальним часом. Це означає, що для кожного процесу всі операції виглядають так, ніби вони виконуються в деякому єдиному порядку, який відповідає порядку їх подання, незалежно від того, на якому вузлі або процесорі вони були виконані. Однак, на відміну від сильної узгодженості, ця модель не гарантує, що всі процеси побачать однаковий стан системи одночасно.

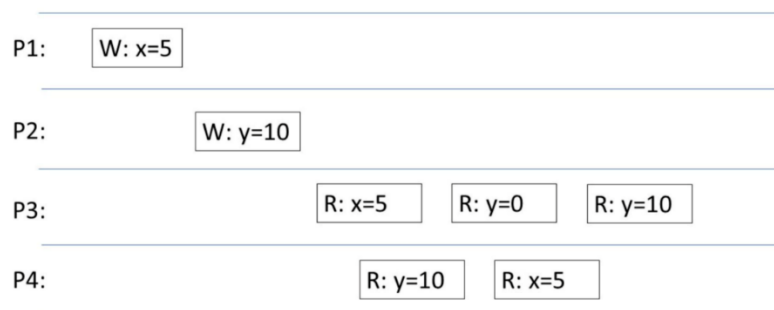


Рис. 1.13 Приклад послідовної узгодженості

- *P1*: Виконує операцію запису $W: x=5$. Це означає, що процес *P1* записує значення 5 у змінну x .

- *P2*: Виконує операцію запису *W*: $y=10$, яка встановлює значення 10 для змінної y .
- *P3*: Читає *R*: $x=5$ та *R*: $y=0$, а потім *R*: $y=10$. Це означає, що процес *P3* бачить записані значення змінних у певному порядку: спочатку $x=5$, потім $y=0$, і тільки після цього $y=10$. Це вказує на те, що процес *P3* спочатку звернувся до вузла, де значення y ще не було оновлене, але потім побачив актуальне значення $y=10$.
- *P4*: Спочатку читає *R*: $y=10$, а потім *R*: $x=5$. Це означає, що процес *P4* бачить послідовно обидва актуальні значення змінних y та x , але їх порядок звернення може відрізнятися від того, що побачив процес *P3*.

Таким чином, послідовна узгодженість забезпечує, що всі процеси бачать операції у певному порядку, який узгоджується з послідовністю подачі операцій, але при цьому різні процеси можуть бачити різні версії даних, залежно від моменту звернення до них. Ця модель дозволяє підвищити продуктивність системи, знижуючи вимоги до негайної синхронізації між вузлами, але зберігаючи узгодженість виконання операцій.

Кінцева узгодженість - ця модель узгодженості забезпечує, що всі репліки даних врешті-решт синхронізуються і зійдуться до одного і того ж значення [18], за умови, що до даних не вносяться нові оновлення (рис. 1.14). Вона дозволяє реплікам тимчасово мати розбіжності, а синхронізація даних відбувається пізніше, що сприяє кращій масштабованості системи. Проте така модель забезпечує слабші гарантії узгодженості, оскільки клієнти можуть тимчасово отримувати різні версії даних залежно від того, до якого вузла вони звертаються.

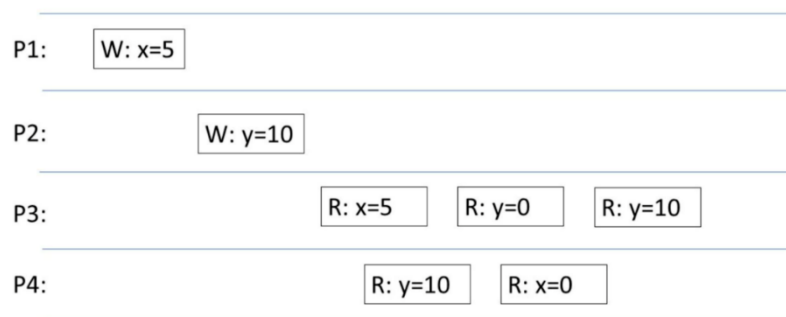


Рис. 1.14 Приклад кінцевої узгодженості

- *P3*: Читає *R*: $x=5$, *R*: $y=0$, а потім *R*: $y=10$. Це означає, що процес *P3* бачить різні версії значень змінних. Спочатку він бачить актуальне значення $x=5$, але значення y він спочатку бачить як 0 (до того, як зміна $y=10$ була синхронізована), а потім як 10.
- *P4*: Читає *R*: $y=10$, а потім *R*: $x=0$. Це вказує на те, що процес *P4* бачить змінну y з актуальним значенням 10, але ще бачить старе значення змінної x , яке дорівнює 0.

Це демонструє основний принцип кінцевої узгодженості: репліки даних можуть тимчасово мати розбіжності, і процеси можуть бачити різні версії даних залежно від того, до якого вузла вони звертаються. Однак, згодом, всі репліки синхронізуються, і система повертається до узгодженого стану, за умови, що не буде нових оновлень даних. Такий підхід дозволяє підвищити масштабованість і продуктивність системи, але забезпечує слабші гарантії узгодженості.

Причинно-наслідкова узгодженість. Ця модель узгодженості гарантує, що операції, які мають причинний зв'язок (наприклад, коли одна операція безпосередньо впливає на іншу), будуть видимі всім процесам у тому ж порядку, в якому вони були виконані (рис. 1.15). Іншими словами, якщо одна операція відбулася раніше іншої в одній сесії, всі вузли системи будуть бачити ці операції в тому ж порядку. Проте повний порядок усіх операцій у системі може бути відсутнім, що дозволяє більш ефективно використовувати ресурси системи за рахунок зниження накладних витрат на синхронізацію.

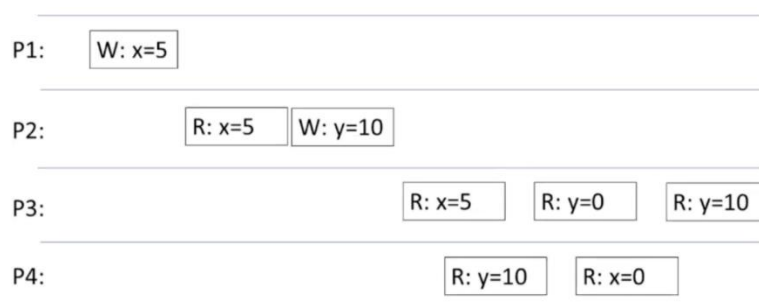


Рис. 1.15 Приклад причинно-наслідкової узгодженості

- *P3*: Читає *R*: $x=5$, *R*: $y=0$, а потім *R*: $y=10$. Це вказує на те, що процес *P3* спочатку бачить актуальне значення змінної x , але бачить застаріле значення

змінної y (до того, як зміна $y=10$ була синхронізована), а потім бачить оновлене значення $y=10$.

- $P4$: Спочатку читає R : $y=10$, а потім R : $x=0$. Це означає, що процес $P4$ бачить зміни в змінних y і x , але порядок звернення може відрізнитися від того, що бачать інші процеси.

Цей приклад демонструє ключову властивість причинно-наслідкової узгодженості: якщо одна операція має залежність від іншої, всі процеси повинні бачити ці операції у тому ж порядку. Однак, послідовність інших операцій, які не пов'язані між собою причинно-наслідковими зв'язками, може відрізнитися для різних процесів. Це дозволяє системі бути більш гнучкою та ефективною з точки зору використання ресурсів, оскільки нема потреби підтримувати строгий глобальний порядок для всіх операцій.

1.5 Постановка наукового завдання

Існуючі підходи до оптимізації виконання запитів часто мають обмеження у випадках великої частоти змін даних, що вимагає розробки нових методів. Основними проблемами залишаються надмірне споживання мережевого трафіку, затримки при синхронізації даних між вузлами та ефективний розподіл фрагментів бази даних для зниження загального часу обробки запитів.

Під час аналізу існуючих рішень виділено кілька ключових проблем, які необхідно вирішити для підвищення ефективності виконання запитів у високонавантажених системах.

Оптимізація мережевого трафіку у розподілених системах типу Leaderless Replication, які використовують алгоритми консенсусу, передача даних між вузлами створює значне навантаження на мережу, що впливає на швидкодію виконання запитів традиційні методи передбачають передачу значних обсягів даних, що не завжди є оптимальним.

Забезпечення узгодженості даних, яка є ключовим фактором у розподілених системах, де кожен вузол може мати локальну копію бази даних. Часті зміни даних створюють додаткові затримки через необхідність синхронізації між вузлами.

Традиційні підходи часто базуються на повній передачі даних, що значно збільшує час виконання операцій.

Нерівномірний розподіл даних між вузлами може спричинити перевантаження окремих вузлів, що призводить до збільшення часу виконання запитів. Сучасні методи ребалансування не завжди враховують динамічний характер змін у системах із високою частотою оновлення даних.

На основі аналізу існуючих проблем було сформульовано наступне наукове завдання: метою дослідження є розробка нових методів підвищення ефективності виконання запитів у високонавантажених розподілених системах шляхом оптимізації мережевого трафіку, забезпечення узгодженості даних та удосконалення методів ребалансування. Для досягнення цієї мети необхідно вирішити такі підзадачі:

1. Розробка методу мінімізації обсягу мережевого трафіку у Raft. Запропонувати підхід, що дозволяє скоротити обсяг передаваних даних шляхом попереднього обміну метаданими між вузлами та матеріалізації результатів на стороні вузла-лідера.
2. Удосконалення методу узгодженості даних на основі модифікованого методу Левенштейна з ваговими коефіцієнтами, що дозволяє скоротити обсяг передаваних даних шляхом передачі лише змін між версіями, що особливо ефективно при частих і невеликих оновленнях.
3. Удосконалення методу ребалансування даних у розподілених системах на основі генетичних алгоритмів. Впровадити адаптивне схрещення та елітарність для більш ефективного розподілу даних між вузлами та скорочення часу обробки запитів.

Висновки до першого розділу

Було проведено детальний аналіз існуючих технологій узгодження даних у розподілених інформаційних системах. Зокрема, розглянуто типи розподілених баз даних, їх структури, та специфіку використання в умовах глобалізації та збільшення обсягів даних. Було підкреслено, що розподілені бази даних мають

численні переваги, включаючи підвищену надійність, масштабованість, оптимізацію витрат та час відгуку. Однак, важливим аспектом є забезпечення узгодженості даних, що вимагає впровадження ефективних механізмів управління транзакціями та реплікації даних.

Значна увага приділена проектуванню та ребалансуванню даних, а також реплікації, як ключовому механізму для підвищення надійності та продуктивності розподілених баз даних. Зроблено акцент на важливості вибору відповідної стратегії узгодження даних для забезпечення стабільної та ефективної роботи системи.

На основі проведеного аналізу існуючих технологій узгодження, реплікації та ребалансування даних у розподілених системах сформульовано основні задачі дослідження, які необхідно вирішити для досягнення мети дисертаційної роботи. Реалізація цих задач дозволить підвищити продуктивність високонавантажених систем за рахунок скорочення часу обробки запитів та покращення узгодженості даних.

РОЗДІЛ 2 МЕТОДИ ПРИШВИДШЕННЯ УЗГОДЖЕНОСТІ ДАНИХ В ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ

2.1 Огляд протоколів консенсусу в розподілених системах

Реплікація станів машин широко використовується для об'єднання набору ненадійних вузлів в одну надійну службу, яка може забезпечити гарантії сильної узгодженості, включаючи лінеаризовність. Як результат, спеціалісти можуть розглядати службу, реалізовану за допомогою реплікованих станів машин, як єдину систему, що полегшує розуміння очікуваної поведінки.

Реплікація станів машин вимагає, щоб кожна машина отримувала одні й ті ж операції в одному й тому ж порядку, що можна досягти за допомогою розподіленого консенсусу.

Протоколи консенсусу дозволяють серверам досягати згоди щодо послідовності операцій для виконання, попри відмову деяких серверів та довільні затримки в мережі. Класичний Paxos [19, 20] є одним з найстаріших і найвиваженіших протоколів консенсусу. Однак, в останні роки Raft [21] поступово випередив Paxos як обраний протокол консенсусу, особливо в промисловості. Багато алгоритмів консенсусу, включаючи Paxos та Raft, використовують підхід на основі лідера для досягнення розподіленого консенсусу. На високому рівні ці алгоритми працюють наступним чином:

- один із [21] серверів призначається лідером, усі операції для машини стану надсилаються лідеру;
- лідер додає операцію до свого журналу та просить інші сервери зробити те саме;
- після того, як лідер отримує підтвердження від більшості серверів про те, що операцію було додано до журналу, він застосовує операцію до своєї машини стану, цей процес повторюється доти, доки лідер не виходить з ладу;
- коли лідер виходить з ладу, інший сервер стає лідером, процес вибору нового лідера включає принаймні більшість серверів, що забезпечує не переписування будь-яких раніше застосованих операцій.

Як показано на рис 2.1, у будь-який момент сервер може перебувати в одному з трьох станів:

- Follower - пасивний стан, у якому сервер відповідає лише на RPC (Remote Procedure Call - виклики віддалених процедур).
- Candidate - активний стан, у якому сервер намагається стати лідером, використовуючи запити на голосування.
- Leader - Активний стан, у якому сервер відповідає за додавання операцій до реплікованого журналу за допомогою RPC AppendEntries.



Рис. 2.1 Переходи між станами сервер

Спочатку сервери знаходяться в стані послідовника. Кожен сервер залишається послідовником, поки він вважає, що лідер працює. Коли послідовник вважає, що лідер вийшов з ладу, він стає кандидатом і намагається бути обраним лідером за допомогою наступних дій:

- Сервер-кандидат надсилає запит на голосування іншим серверам у кластері;
- У запиті кандидат вказує свій термін і індекс останнього запису в журналі;
- Інші сервери (послідовники) перевіряють, чи актуальні записи в журналі кандидата і чи не голосували вони вже за інший сервер у цьому терміні.

Якщо кандидату вдається отримати більшість голосів, він стає лідером. Новий лідер повинен регулярно надсилати сигнали живлення, щоб запобігти таймауту послідовників і їх перетворенню в кандидатів.

Кожен сервер зберігає натуральне число, термін, яке зростає монотонно з часом. Спочатку кожен сервер має поточний термін, рівний нулю. Термін

відправника включений у кожен RPC. Коли сервер отримує RPC, він спочатку перевіряє включений термін. Якщо термін відправника більший, ніж у сервера, сервер оновлює свій термін перед відповіддю на RPC, і, якщо сервер був кандидатом або лідером, перетворюється на послідовника. Якщо термін відправника дорівнює терміну сервера, сервер відповідає на RPC як зазвичай. Якщо термін відправника менший, ніж у сервера, сервер відповідає негативно, включаючи свій термін у відповідь. Коли відправник отримує таку відповідь, він перетворюється на послідовника й оновлює свій термін.

Коли лідер отримує операцію, він додає її до кінця свого журналу з поточним терміном. Потім лідер надсилає запити на додавання записів журналу всім іншим серверам з новим записом журналу. Кожен сервер веде `commit index`, який визначає, які записи журналу є безпечними для застосування до його машини стану, і відповідає лідеру, підтверджуючи успішне отримання нового запису журналу. Після того, як лідер отримує позитивні відповіді від більшості серверів, він оновлює свій `commit index` і застосовує операцію до своєї машини стану. Потім лідер включає оновлений `commit index` у наступні додавання записів журналу. Послідовник додає запис журналу або набір записів лише в тому випадку, якщо його журнал перед цим записом (або записами) є ідентичним журналу лідера. Це гарантує, що записи журналу додаються в правильному порядку, запобігаючи прогалинам у журналі та забезпечуючи правильне застосування записів журналу до машин стану послідовників.

Цей процес триває до тих пір, поки лідер не виходить з ладу, що вимагає обрання нового лідера. Paxos та Raft застосовують різні підходи до цього процесу, тому ми описуємо кожен з них окремо.

2.1.1 Paxos

Послідовник отримає тайм-аут після того, як не отримає недавній запит на додавання записів журналу від лідера. Він стає кандидатом і оновлює свій термін до наступного терміну, такого що

$$t \bmod n = s \quad (2.1)$$

де t — наступний термін, n — кількість серверів, а s — ідентифікатор сервера-кандидата. Кандидат надсилає запити на голосування іншим серверам. Цей RPC включає новий термін кандидата та `commit index`. Коли сервер отримує запити на голосування, він відповідає позитивно, якщо термін кандидата більший, ніж його власний. Ця відповідь також включає будь-які записи журналу, які сервер має у своєму журналі після `commit index` кандидата.

Після того, як кандидат отримує позитивні відповіді на запити на голосування від більшості серверів, кандидат повинен переконатися, що його журнал містить усі зафіксовані записи, перш ніж стати лідером. Він робить це наступним чином. Для кожного індексу після `commit index`, лідер переглядає записи журналу, які він отримав разом із власним журналом. Якщо кандидат побачив запис журналу для цього індексу, він оновлює свій власний журнал, додаючи цей запис із новим терміном. Якщо лідер побачив кілька записів журналу для одного й того ж індексу, він оновлює свій журнал, додаючи запис із найбільшим терміном і новим терміном. Кандидат може тепер стати лідером і почати реплікацію свого журналу на інші сервери.

2.1.2 Raft

Щонайменше один із послідовників отримає тайм-аут після того, як не отримає недавні додавання записів журналу від лідера [22]. Він стає кандидатом і збільшує свій термін. Кандидат надсилає запити на голосування іншим серверам.

Кожен з них включає термін кандидата, а також останній термін і індекс журналу кандидата. Коли сервер отримує запит запити на голосування, він відповідає позитивно, якщо термін кандидата більший або дорівнює його власному, він ще не голосував за кандидата в цьому терміні, і журнал кандидата є принаймні таким же актуальним, як і його власний. Це останнє правило можна перевірити, переконавшись, що останній термін журналу кандидата більший за термін сервера або, якщо вони однакові, що останній індекс кандидата більший за індекс сервера.

Після того, як кандидат отримує позитивні відповіді на запити на голосування від більшості серверів, він може стати лідером і почати реплікацію свого журналу.

Однак для безпеки Raft вимагає, щоб лідер не оновлював свій commit index, доки принаймні один запис журналу з нового терміну не буде зафіксований. Якщо кілька кандидатів у даному терміні розділять голоси таким чином, що жоден з них не отримає більшості, жоден з кандидатів не стане лідером. У цьому випадку кандидат отримує тайм-аут і починає нові вибори з наступним терміном.

2.1.3 Аналіз ефективності в умовах реальних сценаріїв

У Paxos, коли кілька серверів одночасно висувають свої кандидатури на роль лідера, перемогу здобуває кандидат із найвищим терміном. У Raft, у випадку, якщо кілька серверів одночасно стають кандидатами, їхні голоси можуть розподілитися, оскільки вони мають однаковий термін, і жоден з них не здобуде перемогу. Щоб знизити цей ризик, Raft змушує послідовників чекати додатковий час, який вибирається випадково після закінчення тайм-ауту виборів. Це може призвести до того, що процес вибору лідера в Raft буде тривалішим і з більшою варіативністю в часі.

Однак процес вибору лідера в Raft є простішим, ніж у Paxos. Raft дозволяє стати лідером тільки тому кандидату, у якого журнал актуальний, що усуває необхідність надсилати записи журналу під час виборів. У Paxos це не так: кожна позитивна відповідь на запит RequestVote включає записи журналу послідовника після commit index кандидата. Існують різні методи для зменшення кількості переданих записів журналу, але якщо журнал лідера неактуальний, все одно доводиться надсилати деякі записи.

Не тільки у відповіді на запит на голосування у Paxos передаються більше записів журналу, ніж у Raft. В обох алгоритмах, коли кандидат стає лідером, він передає свій журнал на всі інші сервери. У Paxos записи журналу можуть отримати новий термін від лідера, тому лідер може повторно надіслати той самий запис журналу серверу, який вже має його копію. У Raft це не відбувається, оскільки кожен запис журналу зберігає той самий термін протягом всього часу свого

існування. Хоча існують способи пом'якшення цієї проблеми в Paxos, вони виходять за межі спрощеного алгоритму Paxos.

Загалом, у порівнянні з Paxos, підхід Raft до вибору лідера є напрочуд ефективним для такого простого методу.

2.2 Методи оптимізації мережевого трафіку у Raft

У сучасних розподілених системах, де кілька серверів працюють разом для досягнення єдиної мети, мережевий трафік стає критичним компонентом, що визначає продуктивність і ефективність системи. У системах, які використовують Raft консенсус алгоритми, що передбачають постійну взаємодію між серверами для узгодження стану системи, як наслідок, вони створюють значний обсяг мережевого трафіку. Неефективне управління цим трафіком може призвести до затримок, збільшення часу відгуку, а також перевантаження мережі, що негативно вплине на загальну роботу системи. Оптимізація мережевого трафіку спрямована на зменшення обсягів переданих даних і мінімізацію затримок, що забезпечує швидший відгук на запити, знижує навантаження на мережу та скорочує витрати на обчислювальні ресурси.

Однією з головних переваг оптимізації мережевого трафіку є зменшення часу відгуку системи на запити користувачів або внутрішні запити між серверами. За допомогою ефективних стратегій зниження кількості та обсягу переданих даних, система може швидше обробляти запити, що є критичним у середовищах з високими вимогами до продуктивності.

Крім того, оптимізація дозволяє економити ресурси, необхідні для обробки мережевого трафіку. Зниження обсягів даних, що передаються, веде до зменшення використання пропускної здатності мережі, що, в свою чергу, знижує витрати на інфраструктуру і дозволяє більш ефективно використовувати доступні ресурси. Таким чином, оптимізація мережевого трафіку у консенсус алгоритмах не тільки покращує загальну ефективність системи, але й сприяє підвищенню її масштабованості та надійності, забезпечуючи стабільну роботу навіть в умовах

зростаючих навантажень і складності завдань. Існує кілька методів, які допомагають зменшити обсяг переданих даних і знизити затримки при передачі.

2.3 Перетворення Лапласа-Стільтєса

При розробці моделей необхідно враховувати, що час виконання операцій у процесорі, передачі даних між пристроями та оперативною пам'яттю, час передачі даних через мережу і т.д. - є випадковими величинами. Оперування великою кількістю випадкових величин і побудова результуючої функції розподілу ймовірностей $F(t)$ часу очікування чи обробки запиту до бази даних є трудомістким завданням [23]. Перехід від оригіналу $F(t)$ до його зображення $\varphi(s)$ – перетворення Лапласа на формі Стільтєса (ПЛС) – дозволяє суттєво спростити отримання моментів результуючої випадкової величини. У загальному вигляді ПЛС має такий вигляд

$$\varphi(s) = \int_0^{\infty} e^{-st} dF(t) \quad (2.2)$$

Надалі для скорочення говоритимемо про ПЛС функції розподілу ймовірностей випадкової величини або просто про ПЛС випадкової величини. ПЛС відповідної випадкової величини можна розрахувати, диференціюючи (2.1) в нулі

$$M_{\xi} = -\varphi^{(1)}(0), M_{\xi^2} = \varphi^{(2)}(0), \sigma_{\xi}^2 = M_{\xi^2} - M_{\xi}^2 \quad (2.3)$$

Нехай $\xi = \sum_{i=1}^k \xi_i$ – сума незалежних випадкових величин, тоді має місце рівність

$$\varphi(s) = \prod_{i=1}^k \varphi_i(s), \quad (2.4)$$

де $\varphi_i(s)$ – перетворення Лапласа-Стільтєса кожного з доданків ξ_i

Нехай до системи надходить пуасонівський потік заявок (вимог) з параметром λ Тоді функція числа заявок, що надійшли за випадковий інтервал часу ξ визначається виразом

$$w(z) = \sum_i p_i z^i = \varphi(\lambda(1 - z)), \quad (2.5)$$

Звідси одразу отримуємо, ймовірність, що за випадковий інтервал часу ξ не надійде однієї вимоги.

2.4 Метод оптимізації мережевого трафіку Semi-join

У розподілених базах даних оптимізація мережевого трафіку є одним із ключових аспектів забезпечення ефективної роботи системи. Одним із методів, що дозволяє зменшити обсяг даних, що передаються між вузлами системи, є Semi-join [24]. Цей метод дозволяє мінімізувати витрати на передачу даних, знижуючи загальний обсяг переданої інформації, що особливо важливо у випадках складних запитів.

2.4.1 Визначення та принцип роботи

Метод Semi-join є ефективним підходом для оптимізації виконання запитів у розподілених базах даних, особливо у випадках, коли дані зберігаються на різних вузлах системи. Використання Semi-join дозволяє зменшити обсяг даних, що передаються між вузлами, і, таким чином, мінімізувати витрати на мережевий трафік та час виконання запиту.

Напівз'єднання є специфічною алгебраїчною операцією, що поєднує в собі операції з'єднання та проєкції. Воно виконує неповне з'єднання двох реляційних таблиць, залишаючи тільки ті атрибути, які мають відповідність у результаті з'єднання. Це дозволяє уникнути передачі надлишкових даних, які не мають впливу на кінцевий результат.

Розглянемо дві таблиці, що зберігаються на різних вузлах розподіленої системи. Припустимо, що для обох таблиць існують атрибути, які потрібно з'єднати за певною умовою (наприклад, за значенням ключового поля). У традиційному підході для виконання запиту необхідно передати одну з таблиць на інший вузол, де зберігається друга таблиця, і виконати операцію з'єднання. Це

може потребувати значного обсягу передачі даних, що є ресурсомістким процесом.

На відміну від повного з'єднання, Semi-join виконує з'єднання в два етапи:

1. Спочатку виконується проєкція атрибуту з таблиці на вузлі, де вона зберігається. Результуючий набір значень передається на інший вузол. Ця операція залишає тільки ті значення, які будуть використані у з'єднанні, зменшуючи обсяг даних, що передаються.

2. Після отримання проєкції атрибуту на іншому вузлі виконується напівз'єднання, що залишає тільки ті рядки таблиці, які мають відповідність у проєкції.

2.4.1 Процес з'єднання та оцінка витрат при використанні Semi-join

Для оцінки ефективності використання Semi-join розглянемо процес з'єднання. Припустимо, що для таблиць R та S потрібно виконати запит, що включає з'єднання цих двох таблиць. У традиційному підході повного з'єднання (Full Join) необхідно передати одну з таблиць на вузол, де зберігається інша таблиця, і виконати операцію з'єднання.

Загальна вартість такої операції може бути виражена як

$$C_{nj} = C_0 + C_1 \times size(R) \times card(R) \quad (2.6)$$

де C_0 — початкові витрати на встановлення зв'язку між вузлами, C_1 — витрати на передачу одиниці даних, $size(R)$ — розмір переданої таблиці R , $card(R)$ — кількість рядків у таблиці R .

Використовуючи метод Semi-join, ми можемо значно скоротити ці витрати. Спочатку здійснюється проєкція атрибута B на таблицю S , після чого результуючий набір передається на вузол, де зберігається таблиця R . Передача цієї проєкції на інший вузол має вартість

$$C_{sj1} = C_0 + C_1 \times size(B) \times val(B[S]) \quad (2.7)$$

де $val(B[S])$ — кількість унікальних значень у стовпці B таблиці S . Після виконання напівз'єднання отриманий набір рядків таблиці R передається на вузол S для завершення з'єднання

$$C_{sj2} = C_0 + C_1 \times size(R) \times card(R') \quad (2.8)$$

Загальна вартість Semi-join тоді дорівнює

$$C_{sj} = C_{sj1} + C_{sj2} = 2C_0 + C_1 \times (size(B) \times val(B[S]) + size(R) \times card(R')) \quad (2.9)$$

Для того, щоб метод Semi-join був ефективним, його загальні витрати C_{sj} повинні бути меншими, ніж витрати на повне з'єднання C_{nj} .

2.4.3 Використання методу

Нехай у вас є дві таблиці R та S , які зберігаються на різних вузлах:

- таблиця R має 10000 рядків, кожен рядок має розмір 200 байт;
- таблиця S має 5000 рядків, кожен рядок має розмір 100 байт;
- вартість встановлення зв'язку між вузлами $C_0=0.5$ секунд;
- вартість передачі одиниці даних $C_1=0.001$ секунди за байт.

Вартість повного з'єднання Full Join:

Обчислимо загальний обсяг байтів, які потрібно передати

$$size(R) = 10000 \times 200 = 2000000 \text{ б}$$

Загальна вартість комунікацій

$$C_{nj} = 0,5 + 0,001 \times 2000000 = 0,5 + 2000 = 2000,5 \text{ с}$$

Оцінка витрат на з'єднання за допомогою Semi-join. Атрибут B в таблиці S має 1000 унікальних значень. Після проєкції атрибута B на таблицю S , розмір проєкції

$$size(B) \times val(B[S]) = 100 \times 1000 = 100000 \text{ б}$$

Результат напівз'єднання R' має 3000 рядків. Вартість передачі проєкції атрибута B від вузла з таблицею S до вузла з таблицею R

$$C_{sj1} = 0,5 + 0,001 \times 100000 = 100,5 \text{ с}$$

Вартість передачі результату напівз'єднання R на вузол S

$$size(R') = 3000 \times 200 = 600000 \text{ б}$$

$$C_{sj2} = 0,5 + 0,001 \times 600000 = 600,5 \text{ с}$$

Загальна вартість напівз'єднання

$$C_{sj} = 100,5 + 600,5 = 701,0 \text{ с}$$

Таким чином, Semi-join у цьому прикладі значно ефективніший, оскільки він зменшує загальну вартість комунікацій у майже три рази, що робить його кращим вибором для оптимізації мережевого трафіку у розподілених базах даних.

2.4.4 Алгоритм SDD-1 для оптимізації запитів

Класичний алгоритм SDD-1 використовує метод Semi-join для оптимізації запитів у розподілених базах даних [25]. Він складається з двох основних частин: базового алгоритму та після оптимізації. Базовий алгоритм обчислює вигоду від виконання кожного можливого Semi-join, вибираючи ті, що дають максимальну вигоду, та виконує їх до досягнення мінімальних витрат на передачу даних.

Основні кроки алгоритму SDD-1:

1. Обчислення вигоди кожного можливого Semi-join у графі запиту.
2. Виконання Semi-join з максимальною вигодою, після чого перерахування вигоди інших Semi-join.
3. Повторення цього процесу доти, поки не буде виконано всі Semi-join з позитивною вигодою.
4. Вибір вузла з мінімальною вартістю комунікації як остаточного місця виконання запиту та передача всіх необхідних даних на цей вузол для завершення з'єднання.

Semi-join має значні переваги у порівнянні з іншими методами оптимізації мережевого трафіку, такими як повне з'єднання або методи на основі хешування. Основною перевагою Semi-join є здатність значно зменшити обсяг переданих даних, що особливо важливо в умовах обмеженої пропускної здатності мережі. Однак, як і будь-який метод, Semi-join має свої обмеження, зокрема, у випадках, коли дані сильно фрагментовані або кількість записів є дуже великою.

Загалом, застосування методів на основі Semi-join дозволяє значно підвищити ефективність обробки запитів у розподілених базах даних, знижуючи витрати на комунікації та покращуючи час відповіді системи.

2.6 Метод мінімізації мережевого трафіку в Raft Consensus Algorithm

Raft є одним із найпоширеніших алгоритмів досягнення консенсусу, який забезпечує узгодженість даних через постійний обмін інформацією між вузлами системи. Цей процес часто призводить до значного мережевого навантаження, особливо в умовах великих обсягів даних та частих змін.

Для вирішення цієї проблеми було розроблено метод мінімізації мережевого трафіку, який базується на попередньому обміні ключовими векторами та кардинальностями між вузлами перед фактичним передаванням даних. Цей підхід дозволяє значно скоротити обсяг даних, що передаються, шляхом уникнення дублювання та обмеження передачі лише необхідними даними.

Основна логіка роботи методу полягає у попередньому обміні метаданими між вузлами. Кожен вузол спочатку надсилає свої ключові вектори, що представляють унікальні ключі даних, збережених на вузлі, іншим вузлам системи. Після цього здійснюється обмін кардинальностями, що відображають кількість записів для кожного ключа в наборі даних на вузлі. Використовуючи цю інформацію, кожен вузол може визначити, які саме дані необхідно передати, щоб уникнути дублювання та мінімізувати обсяг переданих даних.

Цей метод особливо ефективний для систем із рідкими змінами даних, де зниження мережевого навантаження є ключовим фактором для підвищення продуктивності. Завдяки цьому підходу вдається значно зменшити мережевий трафік, що сприяє підвищенню швидкості виконання запитів та зниженню загальних витрат на інфраструктуру.

2.5 Метод мінімізації мережевого трафіку в Raft

2.5.1 Логіка роботи методу

Розроблений метод мінімізації мережевого трафіку в алгоритмі консенсусу типу Raft ґрунтується на попередньому обміні ключовими векторами та

кардинальностями між вузлами системи з подальшою передачею цих даних лідеру. Такий підхід дозволяє значно зменшити обсяг даних, що передаються в мережі, уникнути дублювання і, таким чином, знизити навантаження на мережу.

Основна логіка методу полягає в тому, що ми спочатку ми використовуємо логіку [9], а саме кожен вузол системи передає лідеру свій ключовий вектор, який містить інформацію про унікальні ключі даних, збережених на цьому вузлі. Крім того, вузли передають лідеру кардинальності, які відображають кількість записів для кожного ключа в наборі даних. Лідер, отримавши ці метадані, аналізує їх та визначає оптимальний спосіб витягування даних з різних вузлів для виконання запиту.

Ключовим моментом є те, що лідер приймає рішення про те, які саме дані необхідно передати з вузлів для матеріалізації результатів запиту. В процесі прийняття рішення лідер використовує отримані ключові вектори та кардинальності для того, щоб уникнути передачі зайвих даних, що суттєво зменшує обсяг мережевого трафіку. Лідер визначає набір даних, які потрібно витягнути з кожного вузла, зважаючи на те, які ключі вже присутні в системі та які необхідні для виконання запиту.

Цей підхід дозволяє не тільки мінімізувати мережевий трафік, але й забезпечує високу ефективність системи в умовах розподілених баз даних, де зміни даних відбуваються відносно рідко. Крім того, завдяки централізованому прийняттю рішень лідером забезпечується узгодженість даних у системі, що є критично важливим для забезпечення надійності та коректності результатів запитів.

2.5.2 Математична модель

Розглянемо систему, що складається з N вузлів, де кожен вузол i зберігає набір даних, представлений у вигляді множини F_i . Для кожного вузла i визначається ключовий вектор KV_i що містить унікальні ключі даних, які зберігаються на цьому вузлі. Кардинальність C_i представляє кількість записів для кожного ключа в наборі даних F_i . Обмін ключовими векторами відбувається наступним чином: на першому етапі кожен вузол i передає свій ключовий вектор KV_i всім іншим вузлам системи

$$KV_{i \rightarrow j} = KV_i, \quad \forall i, j \in \{1, 2, \dots, N\} \quad (2.10)$$

де $KV_{i \rightarrow j}$ — ключовий вектор, який вузол i передає вузлу j . Після обміну ключовими векторами вузли передають свої кардинальності C_i , що дозволяє кожному вузлу отримати інформацію про кількість записів для кожного ключа

$$C_{i \rightarrow j} = C_i, \quad \forall i, j \in \{1, 2, \dots, N\} \quad (2.11)$$

де $C_{i \rightarrow j}$ — кардинальність, яку вузол i передає вузлу j . Після обміну метаданими лідер отримує всю необхідну інформацію для прийняття рішень щодо оптимального витягування даних. Лідер виконує наступні кроки

$$D_{i \rightarrow j} = F_i \cap KV_j, \quad \forall i, j \in \{1, 2, \dots, N\} \quad (2.12)$$

де $D_{i \rightarrow j}$ — дані, які потрібно передати з вузла i до вузла j на основі перетину ключових векторів. Загальний обсяг мережевого трафіку T_{ij} між вузлами i та j обчислюється за формулою

$$T_{ij} = \sum_{i=1}^N \sum_{j=1}^N (|KV_i| + |C_i| + |D_{i \rightarrow j}|) \quad (2.13)$$

де $|KV_i|$ — розмір ключового вектора, $|C_i|$ — розмір кардинальності, а $|D_{i \rightarrow j}|$ — розмір даних, які необхідно передати від вузла i до вузла j .

Запропонована математична модель дозволяє формалізувати процес мінімізації мережевого трафіку в розподілених базах даних, що використовують алгоритми консенсусу типу Raft. Вона забезпечує чітке визначення обсягу переданих даних на основі попереднього обміну ключовими векторами та кардинальностями. Такий підхід дозволяє знизити дублювання даних та зменшити обсяг мережевого трафіку, що особливо важливо для систем з великими обсягами даних або частими змінами.

2.5.3 Використання методу

Розглянемо систему онлайн-журналу студентів, яка складається з п'яти вузлів $N=5$. Кожен вузол зберігає інформацію про студентів у форматі ID, Прізвище, та Ім'я. Для простоти будемо вважати, що на кожному вузлі зберігається

від одного до трьох ключові вектори та з однією кардинальністю. Деякі з цих записів можуть дублюватися на різних вузлах.

Таблиця 2.1 – Таблиця вагових коефіцієнтів

Вузол(Node)	ID	Username
Вузол 1	1	Шевченко Олена
Вузол 1	2	Коваленко Іван
Вузол 2	2	Коваленко Іван
Вузол 2	3	Петренко Дарина
Вузол 3	4	Іваненко Олексій
Вузол 4	5	Гончаренко Марія
Вузол 4	1	Шевченко Олена
Вузол 5	6	Сергієнко Василь
Вузол 5	3	Петренко Дарина
Вузол 5	4	Іваненко Олексій

Кожен вузол передає лідеру ключові вектори та кардинальності. Ключовий вектор KV_i містить ID студентів, збережених на вузлі i , а кардинальність C_i визначає кількість записів для кожного ID:

- вузол №1: $KV_1 = \{1,2\}$, $C_1 = \{1,1\}$
- вузол №2: $KV_2 = \{2,3\}$, $C_2 = \{1,1\}$
- вузол №3: $KV_3 = \{4\}$, $C_3 = \{1\}$
- вузол №4: $KV_4 = \{5,1\}$, $C_4 = \{1,1\}$
- вузол №5: $KV_5 = \{6,3,4\}$, $C_5 = \{1,1,1\}$

Лідер аналізує отримані ключові вектори та кардинальності для визначення оптимального способу витягування даних. Лідер вирішує, які дані потрібно витягнути з різних вузлів, щоб мінімізувати дублювання та обсяг переданих даних. Нехай у рамках даного прикладу лідером буде задано вузол 1. Після отриманих даних від послідовників лідер приймає рішення про те, які дані передавати між вузлами для виконання запиту.

Вузол №1 (Лідер):

- Дані не передаються, оскільки Вузол 1 є лідером.

- Загальний трафік для Вузла 1: $T_1=0$

Вузол 2:

- $KV_2=\{2,3\}$
- $C_2=\{2:1,3:1\}$
- Вузол 2 має дані, які дублюються з Вузлом №1 (ID 2), тому ці дані не потрібно передавати.
- Передача $KV_2=\{3\}$ і відповідної кардинальності $C_2=\{3:1\}$, оскільки дані з ID 3 дублюються з Вузлом №5, але для Вузла №1 ці дані нові.
- Загальний трафік для Вузла №2: $T_2=2$

Вузол 3:

- $KV_3=\{4\}$
- $C_3=\{4:1\}$
- Дані унікальні для цього вузла.
- Передача $KV_3=\{4\}$ і відповідної кардинальності $C_3=\{4:1\}$, оскільки дані з ID 4 вже є на Вузлі №5, але для Вузла №1 ці дані нові.
- Загальний трафік для Вузла №3: $T_3=2$

Вузол 4:

- $KV_4=\{1,5\}$
- $C_4=\{1:1,5:1\}$
- Дані з ID 1 вже є на Вузлі 1, тому їх не потрібно передавати.
- Передача $KV_4=\{5\}$ і відповідної кардинальності $C_4=\{5:1\}$, оскільки дані з ID 5 унікальні для цього вузла.
- Загальний трафік для Вузла 4: $T_4=2$

Вузол 5:

- $KV_5=\{3,4,6\}$
- $C_5=\{3:1,4:1,6:1\}$ Вузол 5 має дані, які дублюються з Вузлом 2 (ID 3) і Вузлом 3 (ID 4), тому ці дані не потрібно передавати.
- Передача $KV_5=\{6\}$ і відповідної кардинальності $C_5=\{6:1\}$, оскільки дані з ID 6 унікальні для цього вузла.
- Загальний трафік для Вузла 5: $T_5=2$

Сумарний мережевий трафік для всієї системи розраховується як сума всіх переданих даних

$$T_{std} = T_1 + T_2 + T_3 + T_4 + T_5 = 0 + 2 + 2 + 2 + 2 = 8$$

Для порівняння сумарний мережевий трафік без використання методу мінімізації мережевого трафіку

$$T_{opt} = T_1 + T_2 + T_3 + T_4 + T_5 = 4 + 4 + 2 + 4 + 6 = 20$$

Відсоток оптимізації

$$\frac{T_{std} - T_{opt}}{T_{std}} \times 100\% = 60\% \quad (2.14)$$

Загальний мережевий трафік у системі складає 8 одиниць що на 60% краще у порівнянні з вхідними даними. Метод мінімізації мережевого трафіку дозволив уникнути дублювання даних і значно зменшити обсяг переданих даних у системі.

2.5.4 Локальне збереження результатів

Локальне збереження результатів є ключовим аспектом підвищення ефективності роботи розподілених систем, зокрема в контексті алгоритмів консенсусу типу Raft. Після виконання запиту в системі, результати запиту можуть бути матеріалізовані на вузлі, з якого був ініційований запит. Це дозволяє уникнути повторного виконання тих самих запитів і значно знижує мережевий трафік та навантаження на лідера.

Нехай запит Q був виконаний на вузлі Si , і результати запиту $R(Q)$ були збережені у локальному сховищі цього вузла

$$R(Q) = \{r_1, r_2, \dots, r_n\} \quad (2.15)$$

де r_i –окремий результат, що відповідає одному із записів у базі даних.

Щоб підвищити ефективність системи, результати матеріалізації також помічаються в кеші. Це дозволяє вузлу визначити, що дані актуальні і не потребують повторного запиту до лідера, якщо до них звертаються повторно в межах терміну дії кешу. Нехай кеш має термін дії T_{cache} , який визначає максимальний час, протягом якого результати вважаються актуальними

$$T_{cache} = t + \Delta t \quad (2.16)$$

де t – час запису результатів у кеш, а Δt – період, протягом якого дані вважаються актуальними.

Якщо новий запит Q' до системи стосується тих самих даних, що вже є в кеші, і $t \leq T_{cache}$, то система використовує кешовані результати $R_{cache}(Q)$

$$R_{cache}(Q) = R(Q), \text{ якщо } t \leq T_{cache} \quad (2.17)$$

Таким чином, якщо результати запиту все ще актуальні, система не звертається до лідера для повторного виконання запиту, що значно зменшує мережевий трафік і підвищує швидкість обробки запитів.

Кожен кеш має свій термін дії, після закінчення якого дані вважаються застарілими, і їх потрібно оновити. Якщо кешований запит більше не актуальний, система повинна звернутися до лідера для отримання оновлених даних і перезаписати результати в кеш

$$R_{cache}(Q) = R_{new}(Q), \text{ якщо } t > T_{cache} \quad (2.18)$$

де $R_{new}(Q)$ — результати, отримані після повторного виконання запиту.

Локальне збереження результатів і управління кешем є критично важливими елементами оптимізації роботи розподілених систем, що використовують алгоритм Raft. Завдяки збереженню результатів і ефективному використанню кешу, система може зменшити кількість звернень до лідера, оптимізувати мережевий трафік і підвищити загальну продуктивність. Правильне управління терміном дії кешу забезпечує актуальність даних і баланс між швидкістю доступу та точністю результатів.

2.6 Метод узгодженості даних на основі алгоритму Левенштейна

2.6.1 Опис методу

Виходячи з попереднього методу оптимізації мережевого трафіку з Raft [21] системах виникає необхідність синхронізації даних між вузлами, особливо коли дані змінюються. Традиційні методи синхронізації, що базуються на передачі повних копій даних, не є ефективними з погляду використання мережевого

трафіку. Замість цього можуть використовуватися методи, що передають лише зміни (дельти) між двома версіями даних. Метод узгодженості даних на основі алгоритму Левенштейна спрямований на забезпечення ефективної передачі та синхронізації даних у розподілених системах. У розподілених базах даних зміни, внесені в одному з вузлів системи, повинні бути оперативно передані та синхронізовані на всіх інших вузлах для підтримки консистентності даних. Однак, у великих системах з високою частотою змін та невеликою дельта зміною цей процес може створювати значне навантаження на мережу, збільшуючи час узгодження даних і знижуючи загальну продуктивність системи.

Одним з таких методів є модифікований метод Левенштейна з ваговими коефіцієнтами, який оптимізує передачу даних шляхом налаштування "вартості" різних операцій редагування. Модифікація методу Левенштейна з ваговими коефіцієнтами дозволяє розширити його застосування для різних типів даних, використовуючи коефіцієнти, які відображають "вартість" кожної операції.

2.6.2 Математична модель

Для оптимізації передачі даних через мережу в розподілених системах, метод Левенштейна був модифікований шляхом введення вагових коефіцієнтів. Ці коефіцієнти можуть бути адаптивними і залежать від таких факторів:

- частота змін - якщо певна операція редагування відбувається частіше, ваговий коефіцієнт може бути знижений для мінімізації витрат;
- типи даних - для різних типів даних (наприклад, текстові або числові) можуть задаватися різні вагові коефіцієнти для операцій;
- контекст - вага операцій може змінюватися залежно від того, де саме в рядку відбувається зміна (наприклад, на початку чи в кінці).

Нехай маємо два рядки даних S і T , що представляють стару і нову версію тексту відповідно. Відстань редагування $d(S, T)$ між рядками S і T визначається як мінімальна сума вартостей операцій вставки w_{ins} , видалення w_{del} та заміни w_{rep} , необхідних для перетворення S у T . Формально

$$d(S, T) = \min \left\{ \begin{array}{l} d(S_{1...i-1}, T_{1...j}) + w_{del} \\ d(S_{1...i}, T_{1...j-1}) + w_{ins} \\ d(S_{1...i-1}, T_{1...j-1}) + w_{del} * cons(S_i, T_j) \end{array} \right\} \quad (2.19)$$

де i та j є індексом символу в рядку S та T відповідно, $cost(S_i, T_j)$ — вартість заміни символу S_i на T_j , яка дорівнює 0, якщо $S_i = T_j$ та 1 — якщо $S_i \neq T_j$.

До переваг методу з ваговими коефіцієнтами можна віднести:

- економія мережевого трафіку, так як передаються лише необхідні мінімальні дельти, що зменшує обсяг переданих даних;
- гнучкість та адаптивність, вагові коефіцієнти можуть бути налаштовані для конкретної системи, зменшуючи вартість часто виконуваних операцій;
- підвищення ефективності синхронізації, використання вагових коефіцієнтів дозволяє знизити час і обчислювальні витрати на синхронізацію даних між вузлами

2.6.3 Визначення вагових коефіцієнтів

Для підбору оптимальних значень вагових коефіцієнтів необхідно мінімізувати загальну вартість редагування для великого набору пар рядків (S_k, T_k) . Таким чином, задача визначення вагових коефіцієнтів формується як задача оптимізації, а цільова функція мінімізує сумарну вартість редагування для набору пар рядків

$$\min \sum_{k=1}^N d(S_k, T_k, w_{del}, w_{ins}, w_{rep}) \quad (2.20)$$

де N — кількість пар рядків у навчальному наборі даних. Вагові коефіцієнти повинні задовольняти наступні обмеження:

- $w_{del} \geq 0$;
- $w_{ins} \geq 0$;
- $w_{rep} \geq 0$;
- Додаткові обмеження можуть бути введені залежно від специфіки даних, наприклад, $w_{ins} \leq w_{rep}$, якщо вставка є "дешевшою" операцією.

Для прикладу необхідно визначити оптимальне значення вагових коефіцієнтів для операцій вставки, видалення та заміни в методі Левенштейна для мінімізації мережевого трафіку під час узгодження даних у розподілених системах. Для дослідження використовувалися приклад даних для системи онлайн журналу, а саме для сутності Lesson поля homework, а саме зміна рядку *"Виконати вправу №200,201,204"* на *"Виконати вправу №200,201,204. Підготуватися до контрольної роботи"* та набір випадкових вагових коефіцієнтів, результати представлені у таблиці:

Таблиця 2.2 – Таблиця вагових коефіцієнтів

№	w_{del}	w_{ins}	w_{rep}	Відстань Левенштейна
1	1,5	3,0	2,0	111,0
2	1,0	2,0	1,0	74,0
3	2,5	2,0	3,0	74,0
4	1,0	2,0	3,0	74,0
5	1,5	3,0	1,0	111,0
6	2,5	1,0	1,0	37,0
7	2,5	2,0	1,0	74,0
8	2,0	3,0	1,5	111,0
9	2,0	2,5	1,5	92,5
10	1,5	2,0	3,0	74,0

Аналіз показує, що найменшу відстань Левенштейна (37) було досягнуто при використанні вагових коефіцієнтів

$$w_{del} = 2,5, w_{ins} = 1,0, w_{rep} = 1,0$$

Це свідчить про те, що в умовах, коли операції видалення є дорогими, а вставка та заміна дешевшими, оптимальною є мінімізація операцій видалення. Інші набори коефіцієнтів, де ваги видалення були меншими, призводили до вищих витрат на редагування, що може свідчити про надмірне використання видалення у даному контексті.

2.6.4 Використання методу

Для прикладу змодельюємо ситуацію в розподіленій базі даних з використання Raft консенсус алгоритму, коли вчитель вніс дані про заняття та після цього вирішив відреагувати дані про домашнє завдання дописавши певні зміни. Після того як дані будуть відправлені на збереження вони відправляються лідеру, де той в свою чергу відправляє дані на всі інші вузли та очікує відповіді більшості. Після повідомлення про успішне збереження вузол з якого було змінено дані зберігає зміни у себе. Вхідні дані:

- Було: *Виконати вправу №201,201,204. Підготуватися до контрольної роботи*
- Стало: *Виконати вправу №200,201,204. Підготуватися до самостійної роботи*

Вагові коефіцієнти: $w_{del} = 2,5$, $w_{ins} = 1,0$, $w_{rep} = 1,0$.

Для початку необхідно провести аналіз змін між рядками:

- у номері вправ після "№" перше "201" змінено на "200";
- у фразі "Підготуватися до контрольної роботи" слово "контрольної" замінено на "самостійної".

Дали необхідно визначити позицій змін:

- потрібно замінити символи на позиціях 17-19 з "201" на "200";
- потрібно замінити на позиціях 47-57 з "контрольної" на "самостійної".

Для визначення операції та номеру позиції змін у тексті ми використовуємо спеціальні ключі, в яких закодовано тип операції та відповідні позиції в рядку. Такий підхід забезпечує компактне та однозначне представлення змін, що необхідно для ефективної синхронізації даних у розподілених системах. Структура ключа має наступний формат:

- Перша частина — код операції:
 - 1 — вставка (Insertion) ;
 - 2 — видалення (Deletion) ;
 - 3 — заміна (Replacement);
- Друга частина — позиції в рядку:
 - для вставки одна позиція, після якої потрібно вставити новий текст;

- о для видалення та заміни необхідні початкова та кінцева позиції фрагмента, який потрібно видалити або замінити.

Використання таких ключів дозволяє: Однозначно ідентифікувати операцію та її місце застосування в тексті, та зменшити обсяг переданих даних, оскільки передається лише необхідна інформація про зміни.

При виконанні запиту передаються дані, а саме сутність яка має бути збережена в журнал змін на кожному вузлі розподіленої бази даних. Порівняймо мережевий трафік між двома варіантами передачі даних.

У першому варіанті (рис. 2.2) ми передаємо повністю оновлені дані. Загальна кількість байтів коли ми передаємо повністю всю зміну дорівнює 272 б.

```
{
  "data": {
    "model": "App/Models/Lesson",
    "model_id": 23235,
    "value": {
      "homework": "Виконати вправу №200,201,204. Підготуватися до контрольної роботи"
    }
  }
}
```

Рис. 2.2 Передача повного значення змінених даних

У другому варіанті (рис. 2.3) ми передаємо лише інформацію про зміни, використовуючи метод передачі дельт. Замість того, щоб пересилати весь оновлений об'єкт, ми передаємо тільки ті частини даних, які зазнали змін, а також інформацію про тип операції та позицію цих змін у рядку. Це суттєво зменшує обсяг мережевого трафіку. Загальна кількість байтів коли ми передаємо повністю всю зміну дорівнює 237 б.

```
{
  "data": {
    "model": "App/Models/Lesson",
    "model_id": 23235,
    "value": {
      "key": {
        "3_17_20": "200",
        "3_47_58": "самостійної"
      }
    }
  }
}
```

Рис. 2.3 Передача лише змінених значень, операцій та позицій

Передача тільки дельт зменшує обсяг мережевого трафіку з 272 байтів до 237 байтів. Це означає, що підхід із передачею лише змін скорочує у даному прикладі мережевий трафік на 12.86% у порівнянні з передачею повного значення. Такий підхід дозволяє суттєво зменшити використання мережі, особливо в системах, де зміни в даних незначні. Зменшення мережевого трафіку є критичним фактором для покращення продуктивності в розподілених базах даних. Менший обсяг даних, що передається між вузлами, безпосередньо впливає на зменшення часу виконання запитів, оскільки менше часу витрачається на передачу даних мережею. Це особливо важливо для РБД, де швидке узгодження і синхронізація даних між вузлами є ключовими для забезпечення узгодженості, доступності та надійності системи. Зменшення мережевого трафіку також дозволяє знизити затримки та підвищити загальну швидкодію системи, що є критично важливим для обробки великої кількості запитів у реальному часі.

Висновки до другого розділу

У цьому розділі приділено увагу протоколам консенсусу, таким як Paxos та Raft, а також методам оптимізації мережевого трафіку, що дозволяють покращити ефективність розподілених баз даних. Протоколи консенсусу, такі як Paxos та Raft, залишаються центральними методами забезпечення узгодженості даних у розподілених системах.

Одним із ключових аспектів оптимізації роботи розподілених систем є мінімізація мережевого трафіку, який є критичним компонентом, що визначає продуктивність і ефективність системи. Було розглянуто кілька методів для досягнення цієї мети. Було розроблено метод мінімізації мережевого трафіку в алгоритмах типу Raft, що базується на попередньому обміні ключовими векторами та кардинальностями між вузлами перед фактичним передаванням даних. Такий підхід дозволяє зменшити обсяг даних, які передаються, та уникнути дублювання, що особливо ефективно в умовах розподілених баз даних із рідкими змінами. Математична модель, представлена в розділі, формалізує процес мінімізації мережевого трафіку та дозволяє знизити дублювання даних, що зменшує обсяг мережевого трафіку до 60% у порівнянні зі стандартними методами.

Також було запропоновано модифікований метод узгодженості даних на основі алгоритму Левенштейна, який спрямований на оптимізацію передачі та синхронізації даних у розподілених системах. Модифікований метод Левенштейна з ваговими коефіцієнтами дозволяє враховувати специфіку різних типів даних та частоту змін для подальшої оптимізації. Цей підхід є особливо корисним для систем із високою частотою змін даних, де передача повних копій була б надмірно затратною.

Таким чином, у цьому розділі було проведено детальний аналіз сучасних методів узгодження даних і оптимізації мережевого трафіку у розподілених інформаційних системах. Вибір конкретного підходу або протоколу залежить від специфіки системи, вимог до продуктивності та ефективності використання ресурсів. Вивчення і порівняння протоколів, таких як Paxos і Raft, у поєднанні з методами оптимізації мережевого трафіку, дозволяє зробити розподілені системи більш масштабованими, надійними та продуктивними.

РОЗДІЛ 3 МЕТОДИ ПОКРАЩЕННЯ ПРОЦЕСУ РЕБАЛАНСУВАННЯ В ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ

3.1 Процеси ребалансування в розподілених базах даних

Ребалансування даних є однією з ключових задач у сфері управління розподіленими базами даних. У сучасних інформаційних системах, де обсяги даних постійно зростають, а вимоги до швидкості та надійності обробки інформації стають дедалі вищими, правильний розподіл даних між серверами та вузлами стає критично важливим [26]. Ефективний розподіл даних між вузлами безпосередньо впливає на продуктивність системи, її здатність швидко обробляти запити користувачів, а також на стійкість до високих навантажень, що можуть виникати під час пікових періодів використання. Основна мета процесу ребалансування полягає в оптимізації розподілу даних таким чином, щоб мінімізувати час виконання запитів, зменшити затримки в обміні інформацією між вузлами та скоротити витрати на передачу даних по мережі. Це дозволяє забезпечити більш ефективне використання ресурсів системи та підвищити загальну продуктивність.

Аналіз існуючих підходів до ребалансування показує, що жоден із методів не є універсальним рішенням, здатним задовольнити вимоги всіх типів розподілених систем. Кожен метод має свої переваги та недоліки, які проявляються в залежності від специфіки конкретної системи, характеру даних та типів запитів, що обробляються [27]. Тому постійне удосконалення методів розподілу даних і розробка нових алгоритмів є актуальною задачею. Впровадження більш ефективних стратегій ребалансування може суттєво підвищити швидкість виконання запитів, зменшити затримки, пов'язані з доступом до віддалених даних, та забезпечити більш рівномірне навантаження на вузли системи. Це особливо важливо для сучасних високонавантажених систем, які працюють у розподіленому середовищі та обслуговують велику кількість користувачів в реальному часі.

Проблема ребалансування даних тісно пов'язана з проблемою розподілу файлів у розподілених комп'ютерних системах. В обох випадках метою є

оптимізація розподілу ресурсів для покращення продуктивності системи. Однак, на відміну від розподілу файлів, де часто не враховується семантика їх обробки, задача розподілу даних в базах даних повинна брати до уваги складні взаємозалежності між різними фрагментами даних. Запити до баз даних часто вимагають доступу до декількох взаємопов'язаних фрагментів інформації, і нерозуміння цих взаємних залежностей може призвести до значних затримок та неефективного використання ресурсів. Врахування цих зв'язків дозволяє розробити більш ефективні алгоритми розподілу, які забезпечують скорочення часу обробки запитів та підвищення загальної продуктивності системи.

Таким чином, дослідження нових методів ребалансування, здатних враховувати специфіку роботи розподілених баз даних, є актуальною науковою та практичною задачею. Розробка таких методів потребує глибокого розуміння як технічних аспектів роботи системи, так і поведінки користувачів, типів запитів та структури даних. Вирішення цієї задачі сприятиме підвищенню ефективності сучасних інформаційних систем, забезпечуючи швидкий та надійний доступ до даних навіть при високих навантаженнях. Це особливо важливо в умовах постійного зростання обсягів даних та вимог до швидкості обробки інформації, що характерно для багатьох галузей, включаючи електронну комерцію, фінансові послуги, соціальні мережі та інші сфери, де розподілені бази даних є основою інформаційної інфраструктури.

Одним із перспективних методів ребалансування є метод використання біогеографічної оптимізації на основі біогеографії(ВБО) [28], яка моделює природні процеси міграції видів між різними середовищами проживання (рис. 3.1). Цей підхід дозволяє адаптивно та динамічно перерозподіляти фрагменти бази даних, забезпечуючи оптимальний баланс навантаження на вузли.

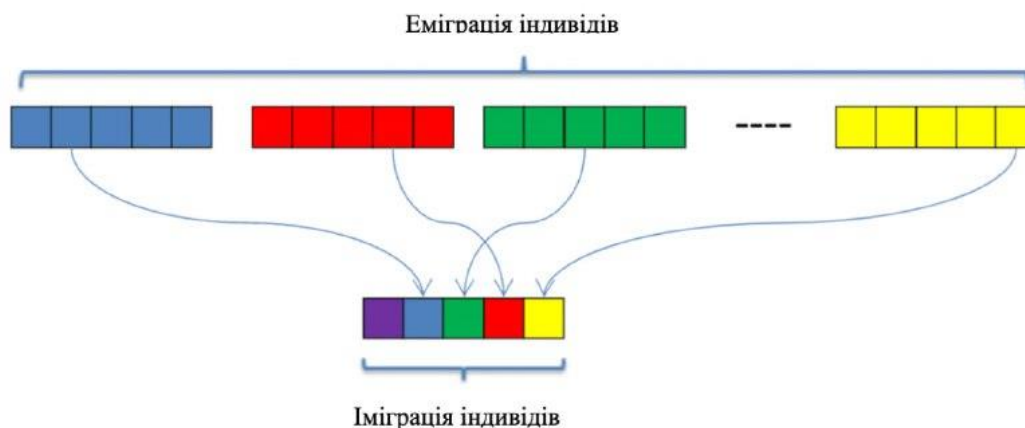


Рис. 3.1 Алгоритм методу використання біогеографічної оптимізації

Метод ВВО імітує природний процес, де міграція видів між середовищами проживання використовується як аналог переміщення фрагментів даних між вузлами. Кожен вузол розглядається як окреме середовище, яке має свої характеристики щодо обчислювальних ресурсів і потужностей. Завдяки цим принципам, метод забезпечує адаптивний розподіл даних, враховуючи поточне завантаження вузлів і зміни в робочих навантаженнях, що є особливо важливим для великих розподілених систем з високою динамікою запитів. До переваг даного методу можна віднести:

- адаптивність, метод дозволяє швидко адаптувати розподіл даних відповідно до змін у навантаженні та запитах системи;
- ефективність, метод значно знижує витрати на передачу даних між вузлами, що покращує загальну продуктивність системи;
- глобальна оптимізація, забезпечує пошук глобальних оптимумів завдяки механізмам міграції та мутації.

До недоліків можна віднести:

- висока обчислювальна складність, метод може вимагати значних обчислювальних ресурсів, що ускладнює його застосування в реальних умовах з великим обсягом даних;
- залежність від початкових умов, початкові конфігурації розподілу можуть суттєво впливати на кінцевий результат, що робить метод чутливим до початкових налаштувань

Метод біогеографічної оптимізації є гарним підходом для ребалансування розподілених баз даних, оскільки він забезпечує динамічний розподіл даних з

урахуванням поточного завантаження вузлів і змін у робочих навантаженнях. Завдяки моделюванню природних процесів міграції видів, метод дозволяє досягти оптимального балансу між продуктивністю і витратами на передачу даних. Проте, висока обчислювальна складність та чутливість до початкових умов вимагають подальших вдосконалень для ефективного застосування в реальних умовах. ВВО застосовується для оптимізації розподілу завдань у хмарних обчисленнях, де він допомагає мінімізувати час виконання та вартість передачі даних між серверами.

3.2 Ребалансування даних на основі генетичних алгоритмів

Проблема ребалансування є складною через взаємозалежність між стратегією виконання запитів (визначається оптимізатором запитів) і розподілом фрагментів. Оптимальний розподіл фрагментів залежить від заданої стратегії виконання запитів, а оптимальна стратегія виконання запитів залежить від фіксованої матеріалізації фрагментів (тобто, визначення місця розташування фрагментів, до яких звертається запит). Основна проблема у визначенні оптимального розподілу полягає у відсутності моделі, яка б представляла залежності між фрагментами, до яких звертається запит. Ці залежності виникають через розподіл відношень на фрагменти та/або доступ до кількох відношень.

Щоб розкласти розподілений запит на фрагментований набір запитів слід використовувати алгоритми розподіленого розкладу запитів і локалізації даних [30]. Цей розкладений запит включає залежності між фрагментами. Ці залежності моделюють бінарні операції (такі як з'єднання, об'єднання) між фрагментами, які потрібно обробити для виконання розподіленого запиту (рис. 3.2).

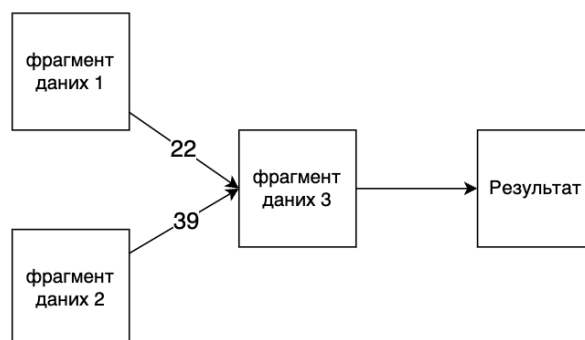


Рис. 3.2 Приклад виконання запиту в РБД

Ми оцінюємо розміри проміжних відношень, що генеруються після виконання угарних і бінарних операцій, використовуючи статистику бази даних (наприклад, кардинальність і довжини кортежів фрагментів/відношень), доступну з системного каталогу. Оскільки ми інтегруємо розподілену обробку запитів, етапи розкладання запитів і локалізації даних усувають доступ до неактуальних фрагментів і генерують стисло дерево запитів на фрагментах. У розподіленій оптимізації запитів і на етапі виконання оптимальні порядки виконання бінарних операцій базуються на стратегії виконання запитів. Стратегія виконання запитів може бути:

1. Move-Small - якщо бінарна операція включає два фрагменти даних, розташовані на двох різних сайтах, тоді менший фрагмент даних пересилається на сайт більших даних.
2. Query-Site - переслати всі фрагменти даних на сайт походження запиту та виконати запит.

Зазначимо, що метою розподілу даних є мінімізація загальної вартості передачі даних для обробки всіх запитів, використовуючи одну з наведених вище стратегій виконання запитів.

Метою розподілу даних є максимізація локальності фрагментів для виконання запитів. Другою метою є інтеграція стратегії виконання запитів, коли запиту потрібно отримати доступ до фрагментів з кількох сайтів, і зменшення загальної вартості передачі даних для обробки всіх запитів. Граф залежностей фрагментів даних кожного запиту моделює два типи вартості передачі даних. Перший тип витрат обумовлений переміщенням даних з сайтів, де розташовані фрагменти даних, до сайту, де запит ініційовано. Другий тип витрат обумовлений переміщенням даних із сайту, де розташований один фрагмент даних, до сайту, де розташований інший фрагмент. У цьому випадку розмір фрагмента даних, необхідний для кожного сайту, не змінюється залежно від розташування інших фрагментів даних, оскільки немає залежності між фрагментами даних, до яких звертається запит. Це справедливо для стратегії обробки запитів query-site і верхнього рівня графів залежностей фрагментів даних. Нехай r_{xj} визначається як розмір даних фрагмента O_j , які потрібно передати на сайт, де ініційовано запит q_x .

Відповідна матриця R має розмір $n \times k$. Зверніть увагу, що це включає верхній рівень графу залежностей фрагментів даних. Нехай q_x буде запитом, ініційованим із сайту S_i , a_{ix} разів за одиничний інтервал часу. Нехай U буде матрицею розміром $m \times k$, де u_{ij} дає кількість даних, яку потрібно передати із сайту, де фрагмент даних O_j розподілений на сайт S_i , де ініційовані запити

$$d_{jj'} = \sum_{x=0}^{n-1} \left(\sum_{i=0}^{m-1} a_{ix} \right) \delta_{jj'}^x, \text{ та } D = \sum_{x=0}^{n-1} \left(\sum_{i=0}^{m-1} \nabla^x \right) \quad (3.1)$$

Нехай $site(O_j)$ позначає сайт, де розташований фрагмент даних O_j . Тоді загальна вартість транспортування t визначається як

$$t = \sum_{j=0}^{k-1} \sum_{j'=0}^{k-1} c_{site(O_j), site(O_{j'})} * d_{jj'} + \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} c_{site(O_j)} * u_{ij} \quad (3.2)$$

де перший доданок представляє вартість передачі даних, яка виникає при обробці бінарних операцій між фрагментами даних, розташованими на різних сайтах, а другий доданок представляє вартість передачі даних для результатів бінарних операцій фрагментів даних до сайту, де ініційований запит. Метою у задачі розподілу даних є мінімізувати t шляхом зміни функції $site(O_j)$ (яка відображає фрагмент даних на сайт).

Для ілюстрації, розглянемо сценарій, показаний раніше (рис. 3.2), в якому чотири фрагменти даних мають бути розподілені на мережу з чотирьох сайтів, зважаючи на те, що є два запити. Частоти доступу запиту 0 та запиту 1 для чотирьох сайтів задані як

$$A = \begin{bmatrix} 0 & 49 \\ 50 & 43 \\ 0 & 0 \\ 45 & 0 \end{bmatrix}$$

Таким чином, загальні частоти доступу для кожного запиту становлять 95 і 92 відповідно, отже, матриця вартості залежностей фрагментів даних задається як

$$D = \begin{bmatrix} 0 & 1196 & 0 & 0 \\ 0 & 0 & 0 & 2090 \\ 3128 & 0 & 0 & 3705 \\ 0 & 4600 & 0 & 0 \end{bmatrix}$$

Припустимо, що матриця розмірів передачі даних для запитів задається як

$$R = \begin{bmatrix} 0 & 0 & 95 & 13 \\ 0 & 15 & 49 & 0 \end{bmatrix}$$

Таким чином, матриця вартості розподілу фрагментів даних задається як

$$U = A \times R = \begin{bmatrix} 0 & 49 \\ 50 & 43 \\ 0 & 0 \\ 45 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 95 & 13 \\ 0 & 15 & 49 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 735 & 2401 & 0 \\ 0 & 645 & 6857 & 650 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 4275 & 585 \end{bmatrix} \quad (3.3)$$

Метод пошуку на основі генетичних алгоритмів натхненні механізмами природної генетики, що ведуть до виживання найбільш пристосованих індивідів. Генетичні алгоритми маніпулюють популяцією потенційних рішень для задачі оптимізації. Зокрема, вони працюють із закодованими представленнями рішень, еквівалентними генетичному матеріалу індивідів у природі, а не безпосередньо з самими рішеннями. У найпростішій формі рішення у популяції кодуються як бінарні рядки. Як і в природі, механізм відбору забезпечує необхідну рушійну силу для кращих рішень, щоб вони вижили. Кожне рішення має значення придатності *fitness*, яке відображає, наскільки воно хороше у порівнянні з іншими рішеннями у популяції. Чим вище значення придатності індивіда, тим вища ймовірність його виживання в наступному поколінні. Рекомбінація генетичного матеріалу в генетичних алгоритмах імітується за допомогою механізму кросовера, що обмінює частини між рядками. Інша операція, звана мутацією, спричиняє спорадичне і випадкове чергування бітів у рядках. Мутація також має пряму аналогію з природою і відіграє роль у відновленні втраченого генетичного матеріалу.

У запропонованому генетичному алгоритмі (рис. 3.3) для проблеми розподілу даних ми кодуємо розподіл кожного фрагмента даних у бінарному представленні.

Наприклад, якщо фрагмент даних призначений сайту 3, тоді його значення призначення дорівнює 11. Значення призначення всіх фрагментів даних об'єднуються в бінарний рядок. Кожен бінарний рядок представляє потенційне рішення задачі розподілу даних. Придатність рядка — це просто вартість розподілу. Механізм відбору реалізований як проста пропорційна схема відбору: рядок із придатністю f призначається $f/(f)$ потомків, де f — це середнє значення придатності популяції. Рядок зі значенням придатності, вищим за середнє,

призначається більш ніж один потомок, тоді як рядок зі значенням придатності, нижчим за середнє, призначається менше одного нащадку.

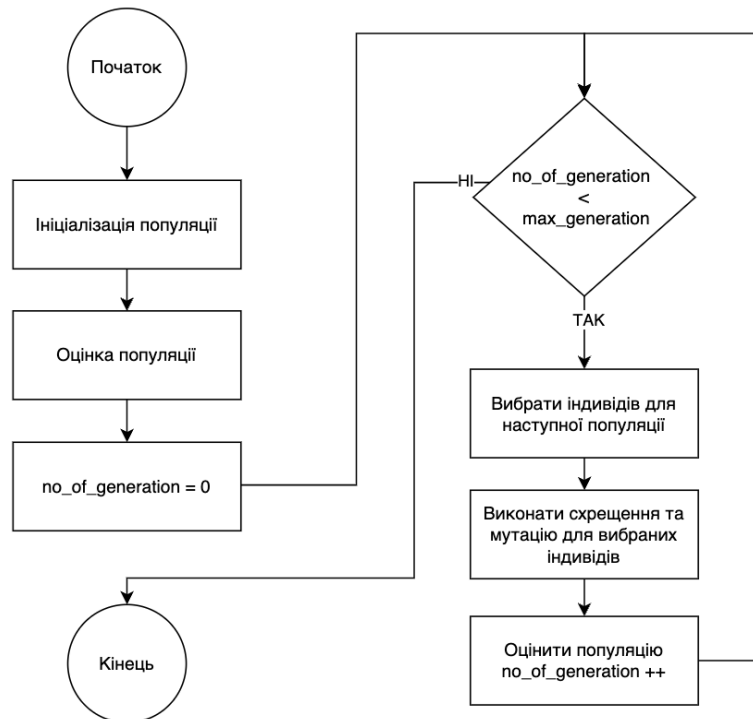


Рис. 3.3 Алгоритму розподілу даних за допомогою генетичних алгоритмів

Кросовер – це ще одна важлива операція ГА [31,32,33]. Пари рядків випадковим чином вибираються з популяції для кросоверу. Ми використовуємо простий одно точковий кросоверний підхід. Припускаючи, що L – це довжина рядка, алгоритм випадковим чином вибирає точку кросоверу, яка може приймати значення в діапазоні від 1 до $L - 1$. Частини двох рядків за межами цієї точки кросоверу обмінюються для утворення двох нових рядків. Точка кросоверу може приймати будь-яке з можливих значень $L - 1$ з однаковою ймовірністю. Зауважте, що кросовер виконується лише тоді, коли випадково згенероване число в діапазоні більше за вказану частоту кросоверу P_c (також звану ймовірністю кросоверу); в іншому випадку рядки залишаються незмінними. Значення P_c знаходиться в діапазоні від 0 до 1. У великій популяції P_c дає частку рядків, які фактично пройшли кросовер.

Після кросоверу рядки піддаються мутації. Мутація біта – це інверсія біта. Так само як P_c контролює ймовірність кросоверу, інший параметр, P_m – частота мутації, дає ймовірність того, що біт буде інвертовано. Біт рядка мутує незалежно.

Тобто мутація одного біта не впливає на ймовірність мутації інших бітів. Мутація розглядається як вторинний оператор із роллю відновлення втраченого генетичного матеріалу. Генетичний алгоритм для задачі розподілу даних описано на блок схемі.

Складність за часом підходу розподілу даних за допомогою генетичного алгоритму визначається як

$$O(GN_p(k^2 + km)) \quad (3.4)$$

де G – це кількість поколінь, а N_p – розмір популяції. Для задачі малого розподілу даних, яка була показана раніше на рисунку 2, початкових хромосом генетичного алгоритму зображено на рисунку (рис. 3.4а). Після ітеративного застосування генетичних операторів, остаточне рішення представлено хромосомою, показаною на рисунку (рис. 3.4б), що відображає розподіл:



Рис. 3.4 Рішення розподілу

Фрагмент даних 0 до вузла 2, фрагмент даних 1 до вузла 1, фрагмент даних 2 до вузла 1 і фрагмент даних 3 до вузла 2. Загальна вартість розподілу становить 67378 байт. Час виконання на робочій станції Sun Ultra складає 474 секунди.

3.3 Ребалансування з елітарністю та адаптивним схрещенням

Основні виклики ребалансування в розподілених базах даних включають мінімізацію часу виконання запитів та зменшення витрат на передачу даних між вузлами. Розподіл даних повинен забезпечувати рівномірне навантаження на сервери, мінімізуючи при цьому затримки та підвищуючи загальну продуктивність системи. Традиційні методи часто не справляються з цими задачами в умовах великих обсягів даних та динамічних змін у навантаженні. Генетичні алгоритми, завдяки своїй гнучкості та здатності адаптуватися до складних умов, стають перспективним підходом для вирішення цих проблем.

3.3.1 Механізм елітарності в генетичних алгоритмах

Елітарність — це один з ключових механізмів у генетичних алгоритмах, який забезпечує збереження найкращих індивідів з покоління в покоління, що дозволяє підтримувати високий рівень якості рішень. Це досягається шляхом прямого перенесення одного або певної кількості найкращих індивідів до наступного покоління без змін. Застосування елітарності гарантує, що навіть у разі несприятливих мутацій чи невдалих операцій схрещення, найкращі індивіди будуть захищені від деградації і завжди будуть частиною поточної популяції. Це значно знижує ризик випадкової втрати оптимальних або близьких до оптимальних рішень, що особливо важливо на пізніх етапах еволюції алгоритму, коли наближається глобальний мінімум або максимум.

Роль елітарності полягає не лише у захисті найкращих рішень, але й у прискоренні процесу конвергенції, оскільки найкращі рішення залишаються в популяції та можуть бути використані для подальшого вдосконалення через мутації або схрещення з іншими індивідами (рис. 3.5). Це дозволяє ефективніше та швидше знайти оптимальне рішення, зменшуючи кількість ітерацій, необхідних для досягнення високих результатів. У задачах, що стосуються ребалансування даних у розподілених базах, елітарність є критично важливим елементом, оскільки втрата вже знайдених ефективних розподілів може призвести до значного погіршення продуктивності системи.

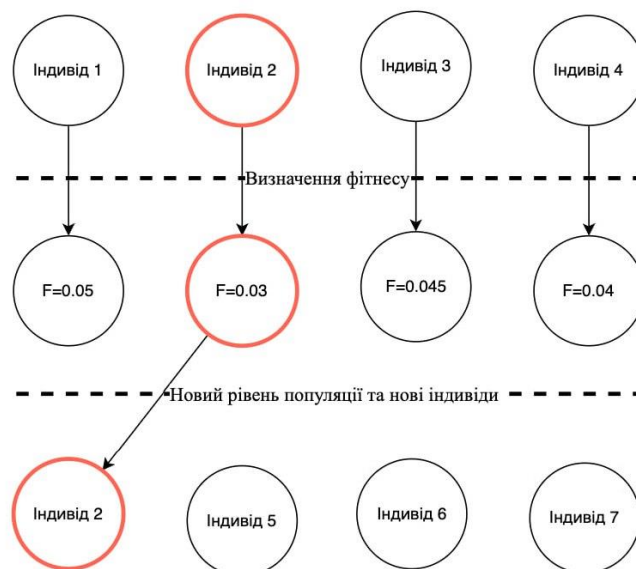


Рис. 3.5 Механізм елітарності в генетичних алгоритмах

У задачах ребалансування даних це особливо важливо, оскільки втрата ефективних розподілів може призвести до значних втрат у продуктивності системи.

3.3.2 Адаптивне схрещення

Адаптивне схрещення — це підхід, при якому параметри схрещення в генетичному алгоритмі динамічно змінюються залежно від поточного стану популяції (рис. 3.6). На відміну від традиційних методів, де параметри залишаються постійними протягом усього процесу еволюції, адаптивне схрещення дозволяє алгоритму краще пристосовуватися до різних фаз пошуку. Основні принципи адаптивного схрещення включають:

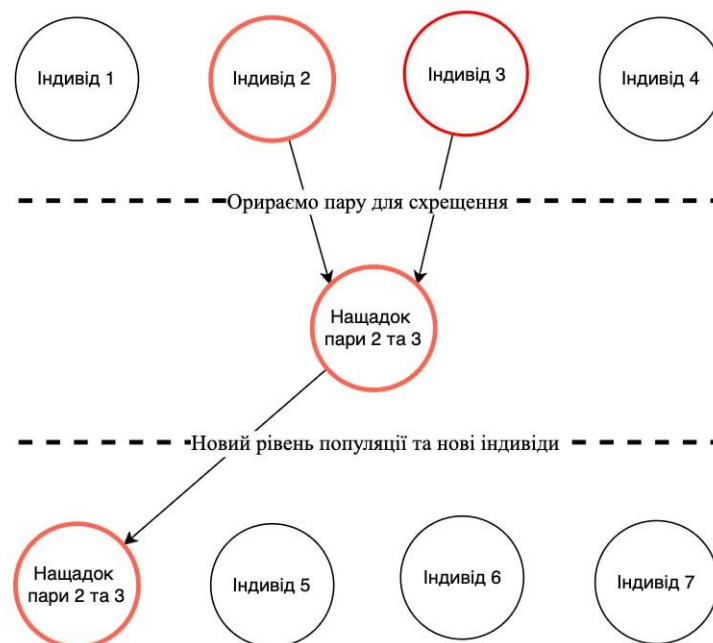


Рис. 3.6 Механізм адаптивного схрещення в генетичних алгоритмах

- Динамічна зміна ймовірності схрещення - параметри схрещення коригуються на основі різноманітності популяції або швидкості конвергенції. Наприклад, на початкових етапах алгоритму може використовуватися висока ймовірність схрещення для забезпечення різноманітності, а на пізніших етапах — зменшуватися для тонкого налаштування рішень.

- Вибір схеми схрещення - можуть застосовуватися різні методи схрещення (одно точкове, багато точкове, уніформне) залежно від ефективності на поточному етапі еволюції.

Адаптивне схрещення підвищує ефективність алгоритму, дозволяючи більш ефективно досліджувати простір пошуку та уникати локальних мінімумів.

3.4 Процеси генетичного алгоритму з урахуванням елітарності та адаптивності

З метою підвищення ефективності ребалансування даних та мінімізації часу виконання запитів, було розроблено генетичний алгоритм, що інтегрує механізми елітарності та адаптивності (рис. 3.7). Елітарність забезпечує збереження найкращих знайдених рішень протягом еволюції, а адаптивність дозволяє динамічно змінювати параметри схрещення та мутації залежно від поточного стану популяції.

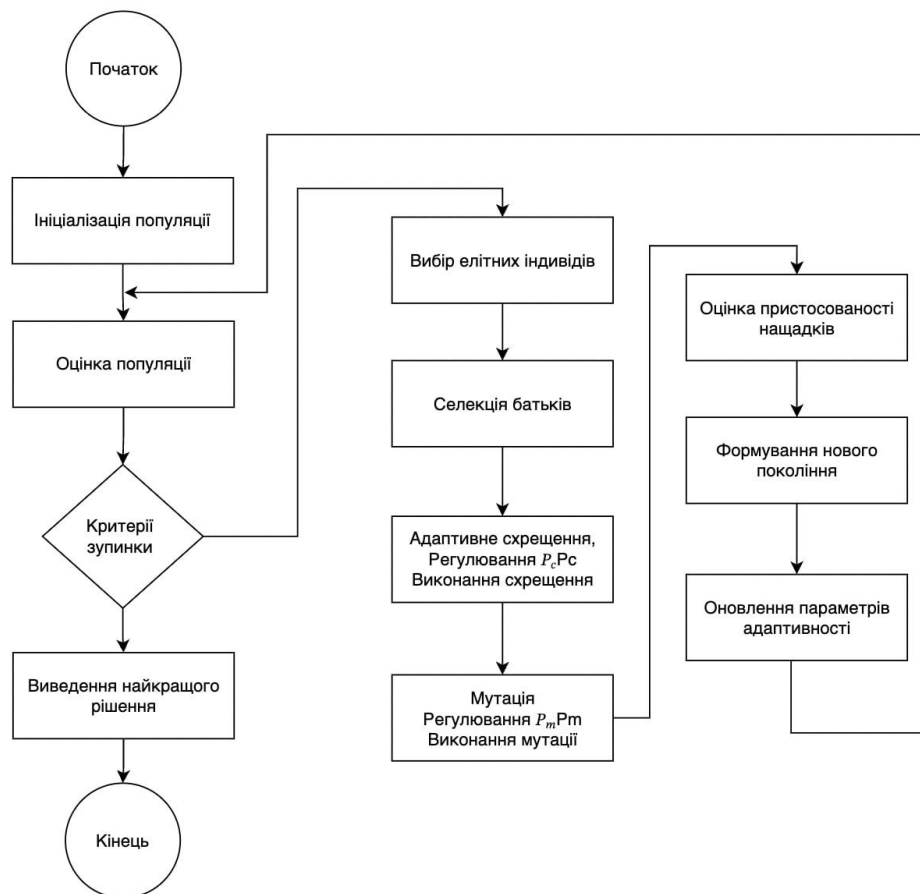


Рис. 3.7 Алгоритму ГА з урахуванням елітарності та адаптивності

Процес ребалансування складається з наступних кроків:

- Ініціалізація популяції - початкова популяція формується випадковим чином або з використанням евристик для забезпечення початкової різноманітності.

- Відбір - застосовується метод відбору (наприклад, турнірний відбір або метод рулетки) для вибору батьківських індивідів на основі їх пристосованості, яка оцінюється за функцією вартості $C(S)$.
- Схрещення з адаптивністю, що включає в себе адаптивну ймовірність схрещення та методи схрещення. Адаптивна ймовірність схрещення – процес коли ймовірність схрещення P_c змінюється залежно від середньої пристосованості популяції або рівня різноманітності. Наприклад, якщо різноманітність низька, P_c збільшується для уникнення застою. Використовуються різні методи: одно точкове, доточкове, уніформне, вибір яких може змінюватися адаптивно.
- Процес мутації - випадкова зміна значення гену (перепризначення даних на інший вузол) для підтримання генетичного різноманіття. Адаптивна ймовірність мутації - ймовірність мутації P_m також коригується залежно від стану популяції. Висока ймовірність мутації на початку допомагає дослідити простір пошуку, тоді як на пізніх етапах вона зменшується для стабілізації результатів.
- Елітарність - збереження найкращих індивідів. Визначена кількість найкращих індивідів без змін переноситься до наступного покоління. Елітарність забезпечує постійне покращення найкращого знайденого рішення та запобігає його втраті.
- Заміна поколінь забезпечує формування нового покоління - поєднання нащадків, отриманих після схрещення та мутації, з елітними індивідами та контролю розміру популяції, де забезпечується постійна чисельність популяції шляхом відбору необхідної кількості індивідів.
- Критерії зупинки:
 - o Максимальна кількість поколінь: Алгоритм завершується після досягнення заданої кількості поколінь.
 - o Конвергенція: Зупинка при відсутності покращення найкращого рішення протягом певної кількості поколінь.
 - o Досягнення цільової пристосованості: Якщо значення функції вартості стає нижчим за заданий поріг.

3.5 Формалізація задачі ребалансування

У розподілених базах даних ефективний розподіл даних між серверами є критичним для мінімізації часу виконання запитів. Неправильний розподіл може призвести до перевантаження окремих серверів, збільшення затримок та витрат на передачу даних між вузлами[34]. Метою є оптимальний розподіл даних, який забезпечує баланс навантаження та мінімізує загальний час обробки запитів.

Визначення основних компонентів:

- множина серверів $S=\{s_1, s_2, s_M\}$ - сервери або вузли, доступні для зберігання та обробки даних, $M = 3$;
- множина даних $D=\{d_1, d_2, d_N\}$ - це набір фрагментів бази даних або окремих таблиць, які необхідно розподілити між серверами: $N = 5$;
- множина запитів $Q=\{q_1, q_2, q_K\}$ - запити до бази даних, які потребують доступу до певних даних з D , $K = 3$.

Головна мета методу — це мінімізувати загальний час виконання всіх запитів[35]. Оскільки нас цікавить загальний час виконання всіх запитів, ми підсумовуємо час виконання кожного запиту. Та задаємо матрицю розподілу даних X яка визначає, на яких серверах розміщені дані, і відповідно, впливає на час доступу до цих даних. Матриця $X=[x_{ij}]$ має розмірність $N \times M$, де $x_{ij}=1$, якщо дані d_i розміщені на сервері s_j та $x_{ij}=0$, якщо дані d_i не розміщені на сервері s_j . Тобто цільову функцію формулюємо як

$$\min(F(X) = \sum_{k=1}^K T_k(X) \quad (3.6)$$

де $F(X)$ — загальний час виконання всіх запитів при розподілі даних, заданому матрицею X , та $T_k(X)$ — час виконання окремого запиту q_k при розподілі X .

Моделювання часу виконання запитів та час виконання запиту $T_k(X)$ залежить від кількох факторів:

- Час обробки на серверах - час, необхідний серверу для виконання частини запиту.
- Час передачі даних між серверами, якщо запит потребує даних з різних серверів, необхідно врахувати витрати на передачу цих даних.

Час обробки частини запиту q_k на сервері s_j можна позначити як

$$\max(t_{\text{обробки}}(q_k, s_j)) \quad (3.7)$$

Цей час враховує обробку частини запиту на конкретному сервері. Припускається, що частини запиту виконуються паралельно на різних серверах. Оскільки запит виконується паралельно, загальний час обробки визначається як максимальний з часу обробки всіх частин запиту на різних серверах. Тоді час виконання запиту $T_k(X)$ можна визначити за формулою

$$T_k(X) = \max_{s_j \in S_k} \left(t_{\text{обробки}}(q_k, s_j) + \sum_{(s_j, s_l) \in P_k} t_{\text{передачі}}(s_j, s_l) \right) \quad (3.8)$$

де:

- S_k – множина серверів, на яких знаходяться необхідні для запиту дані.
- P_k – множина пар серверів, між якими передаються дані;
- $t_{\text{prcs}}(q_k, s_j)$ – час обробки частини запиту на сервері s_j ;
- $t_{\text{trns}}(s_j, s_l)$ – час передачі даних між серверами s_j .

Також обов'язково необхідно враховувати обмеження ємності серверів, так як неможливо розмістити більше даних, ніж дозволяє ємність сервера. Кожен сервер має обмежену ємність C_j , що можна визначити за формулою

$$\sum_{i=1}^N x_{ij} * w_i \leq C_j, \quad \forall j = 1, \dots, M \quad (3.9)$$

де w_i – розмір даних d_i .

Генетичний алгоритм використовується для пошуку оптимального розподілу даних X , який мінімізує цільову функцію $F(X)$. Кодування використовується для опису можливих розподілів даних між кількома серверами в рамках генетичного алгоритму. Кожен індивід в популяції представляє окремий варіант такого розподілу, де кожен ген вказує на сервер, на якому розміщені конкретні дані d_i

$$c = [c_1, c_2, \dots, c_N], \text{ де } c_i \in \{1, 2, \dots, M\} \quad (3.10)$$

де $c_i=j$ означає, що дані d_i розміщені на сервері s_j .

Початкова популяція генерується випадковим чином, враховуючи обмеження ємності серверів. Випадковість забезпечує різноманітність початкових рішень, що підвищує ймовірність знайти глобальний оптимум. Для оцінки якості кожного індивіда в популяції в рамках генетичного алгоритму використовують фітнес-функцію з штрафами(додаткові значення, які додаються до T_k у випадку порушення обмежень). Тому фітнес-функцію можна визначати як

$$\text{Фітнес}(c) = \frac{1}{T_k + \text{Штрафи}} \quad (3.11)$$

Далі відбувається процес селекції, де випадково вибирається кілька індивідів, і найкращий з них переходить до наступного покоління. Імовірність вибору індивіда пропорційна його фітнесу.

В генетичних алгоритмах схрещення є одним із ключових операторів, який дозволяє комбінувати генетичну інформацію двох батьків для отримання нових нащадків. Адаптивне схрещення означає, що ймовірність схрещення P_c не є фіксованою, а змінюється залежно від стану популяції або характеристик індивідів. Це дозволяє алгоритму динамічно регулювати процес пошуку оптимального рішення. Визначення максимального та мінімального значень $P_{c_{max}}$ та $P_{c_{min}}$.

$P_{c_{max}}$ - максимальне значення ймовірності схрещення, яке застосовується, коли необхідно підвищити різноманітність популяції. $P_{c_{min}}$ - мінімальне значення, яке використовується, щоб зберегти хороші рішення та уникнути руйнування високопродуктивних індивідів. Параметри алгоритму $P_{c_{max}}$ та $P_{c_{min}}$ встановлюються на початку та залишаються постійними протягом виконання алгоритму.

Ймовірність схрещення P_c змінюється в залежності від стану популяції може бути розрахована за формулою

$$P_c = P_{c_{min}} + (P_{c_{max}} - P_{c_{min}}) * \frac{f_{max} - f_{\text{середнє}}}{f_{max} - f_{min}} \quad (3.12)$$

де f_{max} , f_{min} , $f_{\text{середнє}}$ — відповідно, максимальний, мінімальний та середній фітнес у популяції.

Адаптивна мутація означає, що ймовірність мутації P_m змінюється динамічно під час виконання алгоритму залежно від стану популяції [36]. Це дозволяє алгоритму ефективно балансувати між пошуком нових рішень та експлуатацією вже знайдених хороших рішень. Адаптивна ймовірність мутації P_m може бути визначена за допомогою наступної формули

$$P_m = P_{m_{min}} + (P_{m_{max}} - P_{m_{min}}) * \frac{f_{\text{середнє}} - f_{min}}{f_{max} - f_{min}} \quad (3.13)$$

Наступним етапом є процес елітарності. Елітарність є важливою стратегією в генетичних алгоритмах, яка гарантує, що найкращі рішення з поточного покоління не будуть втрачені в процесі мутацій або кросоверів. Ця стратегія допомагає зберігати високу якість рішень і сприяє прискоренню пошуку оптимальних рішень.

Детальне математичне моделювання методу ребалансування даних на основі генетичних алгоритмів з елітарністю та адаптивним схрещенням показує, як кожен елемент алгоритму сприяє досягненню головної мети — мінімізації часу виконання запитів. Розроблена модель враховує специфіку розподілених баз даних та дозволяє ефективно знаходити оптимальні розподіли даних, забезпечуючи баланс між пошуком нових рішень та експлуатацією вже знайдених.

3.6 Використання запропонованого методу ребалансування

Для успішного застосування розробленого методу ребалансування необхідно зібрати відповідні дані, які слугуватимуть вхідними параметрами для алгоритму та дозволять оцінити його ефективність. До збору даних можна віднести наступні кроки:

- Дані про деталі таблиці, їх розміри, кількість записів, зв'язки між таблицями.
- Дані про запити, що виконуються в системі, а саме частоту виконання, час виконання, дані або таблиці використовує запит;

- характеристики вузлів, ємність зберігання(максимальний обсяг даних, який може зберігати кожен вузол), обчислювальна потужність(процесорні характеристики, обсяг оперативної пам'яті), мережеві характеристики (пропускна здатність, затримки між вузлами)

Для прикладу використання методу ребалансування даних за допомогою генетичних алгоритмів з елітарністю та адаптивним схрещенням буде використано наступні вхідні параметри:

- фрагменти бази даних (табл. 3.1) $N = 5$;
- запити та їх вимоги (табл. 3.2)
- кількість серверів $M = 3$;
- ємність серверів $C_1 = C_2 = C_3 = 1000 \text{ МБ}$;
- час обробки 1 МБ даних 0.01 секунди.
- швидкість передачі між серверами: 100 МБ/с.
- коефіцієнт затримки мережі: 0.1 секунди на кожне з'єднання між серверами;
- коефіцієнт штрафу за перевищення ємності α : 1000.

Таблиця 3.1 – Таблиця даних та їх розмірів

Номер даних d_i	Опис	Розмір w_i МБ
d_1	Користувачі	500
d_2	Оцінки	400
d_3	Заняття	300
d_4	Розклад	350
d_5	Групи	150

Таблиця 3.2 – Таблиця запитів та їх вимог

Запит q_k	Опис	Необхідні дані d_k
q_1	Отримати журнал оцінок	$\{d_1, d_2\}$
q_2	Завантаження звіту	$\{d_1, d_5\}$
q_3	Перегляд розкладу	$\{d_3, d_4, d_5\}$
q_4	Перегляд щоденника	$\{d_1, d_3\}$

Початкова популяція генетичного алгоритму починається з генерації індивідів [37,38]. Кожен індивід представляється вектором довжиною N , де кожен елемент вектора c_i відповідає серверу, на якому розміщені дані d_i . Припустимо, розмір популяції $P=4$. Початкова популяція генерується випадковим чином з урахуванням обмежень ємності серверів. Отримуємо випадковий набір індивідів:

- Індивід 1: [1,1,2,2,3]
- Індивід 2: [2,1,3,2,1]
- Індивід 3: [3,2,1,1,2]
- Індивід 4: [1,1,1,1,1]

Далі необхідно зробити обчислення фітнес-функції для кожного індивіда[39]. Для цього спочатку зробимо розрахунок часу виконання запитів для кожного індивіда. Почнемо з перевірки обмежень ємності серверів

Сервер s_1 : $350+550=900$ МБ

Сервер s_2 : $250+200=450$ МБ

Сервер s_3 : 100 МБ

В результаті бачимо що всі обмеження виконані. Наступний етап - це розрахунок часу виконання запитів.

Для запиту запит q_1 необхідні дані d_1 на s_1 , d_2 на s_1 . Отримуємо дані:

Час обробки : $350+550 \times 0,01 = 9$ с.

Час передачі: обидва дані на $s_1 = 0$ с.

Загальний час: 9,0 с;

Для запиту запит q_2 необхідні дані d_1 на s_1 , d_5 на s_3 . Отримуємо дані:

Час обробки: $s_1=3$ с, $s_3=1$ с.

Час передачі: $100/100+0,1=1,1$ с.

Загальний час: $\max(3.5, 1.0)+1,1= 4,6$ с.

Для запиту запит q_3 необхідні дані d_3, d_4 на s_2 , d_5 на s_3 . Отримуємо дані:

Час обробки: $s_2=(200+250) \times 0,01=4,5$ с, $s_3=1$ с.

Час передачі: $100/100+0.1=1,1$ с.;

Загальний час: $\max(4.5, 1)+1,1=5,6$ с;

Для запиту запит q_4 необхідні дані d_1 на s_1 , d_3 на s_2 . Отримуємо дані:

Час обробки: $s_2=3,5$ с, $s_3=2,5$ с.

Час передачі: $250/100+0,1=2,6$ с.

Загальний час: $\max(3.5, 2.5)+2,6=6,1$ с.

В результаті отримуємо сумарний час виконання запитів та фітнес для індивіда №1:

$$F_1 = 9,0+4,6+5,6+6,1=25,3 \text{ с.}$$

$$\text{Фітнес}_1 = \frac{1}{F_1} \approx 0,0395$$

Повторюємо даний процес для індивідів №2,3 та отримуємо:

$$F_2 = 9,1+4,6+6,2+6,1=26 \text{ с.}$$

$$\text{Фітнес}_2 = \frac{1}{F_2} \approx 0,0385$$

$$F_3 = 9,1+4,6+5,6+6,1=25,4 \text{ с.}$$

$$\text{Фітнес}_3 = \frac{1}{F_3} \approx 0,0394$$

Для індивіда №4 є виняток. Для цього необхідно зробити перевірку на обмеження ємності серверів за формулою 3.9: $350+550+250+200+100=1450$ МБ, що перевищує обмеження ємності сервера на $1450-1000=450$ МБ. В результаті даний індивід отримує P_n штраф [40] з заданим коефіцієнтом штрафу

$$P_n = \alpha \times \Delta = 450 \times 1000 = 450000,$$

це робиться для того, щоб сильно зменшити фітнес індивіда з порушеннями. Загальний час та фітнес-функції зі штрафом буде дорівнювати

$$F_4 = 9,0+4,5+5,5+6,0=25 \text{ с.}$$

$$\text{Фітнес}_4 = \frac{1}{F_4 + P_n} \approx 2,2222 \times 10^{-6} \approx 0$$

Розрахувавши загальний час та фітнес функцію, для кожного індивіда отримуємо наступні значення фітнес функції враховуючи штрафи (табл. 3.3):

Таблиця 3.3 – Підсумок фітнес-функцій

Індивід	Сумарний час F (с)	Фітнес
1	25,3	0,0395
2	26	0,0385
3	25,4	0,0394
4	450025	0,000002

Після отримання всіх фітнес функцій ми обираємо найкращі індивіди для еліти. Для поточного покоління було обрано індивід зі значенням 0.0395, який є найкращим серед усіх індивідів. До решти індивідів застосовуємо турнірний відбір, для селекції батьків - процес вибору індивідів з поточної популяції для участі в операціях схрещення з метою отримання нащадків для наступного покоління припустимо, що обрані пари Індивід №2 та Індивід №3.

Далі відбувається обчислення параметрів адаптивного схрещення. У генетичних алгоритмах ймовірність схрещення P_c є важливим параметром, який визначає, з якою ймовірністю обрані батьківські пари будуть піддаватися операції схрещення для створення нащадків. В адаптивному схрещенні P_c не є фіксованою величиною, а змінюється залежно від стану популяції або характеристик індивідів. Параметри $P_{c_{max}}$ та $P_{c_{min}}$ визначають максимальне та мінімальне значення цієї ймовірності. $P_{c_{max}}$ зазвичай встановлюється в діапазоні від 0,8 до 1, а $P_{c_{min}}$ зазвичай знаходиться в діапазоні від 0,5 до 0,7. У нашому прикладі використовуються параметри $P_{c_{max}} = 0,9$ та $P_{c_{min}} = 0,6$. Вхідні параметри для адаптивного схрещення:

- $P_{c_{max}} = 0,9$
- $P_{c_{min}} = 0,6$
- $f_{max} = 0,0395$
- $f_{min} = 0,0385$
- $f_{\text{середнє}} = \frac{0,0395 + 0,0385 + 0,0394}{3} \approx 0,0391$
- $f_{\text{поточний}} = \frac{0,0385 + 0,0394}{2} \approx 0,03895$ – для пари індивід №2 та індивід №3.
- $P_m = P_{m_{min}} + (P_{m_{max}} - P_{m_{min}}) * \frac{f_{\text{середнє}} - f_{min}}{f_{max} - f_{min}}$

Ймовірність схрещення за формулою 3.12 буде дорівнювати

$$P_c = 0,765$$

Далі генеруємо випадкове число $r \in [0,1]$, якщо $r < P_c$, виконуємо схрещення. Припустимо, $r = 0,5 < 0,6$, тому схрещення відбувається. Використовуємо одно точкове схрещення випадково обираючи точку розрізу після другого гена. В результаті отримуємо такий набір: батьки - індивід №1: [2,1,3,1,2] та індивід №2: [3,2,1,1,2] та нащадок №1: [2,1,1,1,2] та нащадок №2: [3,2,3,2,1].

Наступний етап - адаптивна мутація. Для адаптивної мутації визначимо параметри мінімальної та максимальної ймовірності мутації. Мінімальна ймовірність мутації це найменше значення ймовірності мутації, до якого може знизитися. Використовується у випадку коли популяція досягає високого рівня фітнесу, невелика ймовірність мутації дозволяє підтримувати мінімальний рівень генетичної різноманітності. У вашому прикладі $P_{m_{min}} = 0,01$ (1%). Максимальна ймовірність мутації встановлює верхню межу для ймовірності мутації. Використовується для підвищення різноманітності, коли популяція має низький рівень пристосованості або застрягає в локальних мінімумах, збільшення ймовірності мутації допомагає дослідити нові області простору рішень. У нашому прикладі $P_{m_{max}} = 0,1$ (10%). Визначити ймовірності мутації за формулою 3.13

$$P_m = 0,0775.$$

Генеруємо випадкові числа для кожного гена. Припустимо, що ген c_2 мутує, в результаті змінюємо c_2 з 1 на 2. Результат: [2, 2, 1, 1, 2].

Наступним етапом формуємо нове покоління з отриманих даних, що складається еліти (індивід №1), оновленого нащадку №1, нащадку та індивід №2: [1,1,2,2,3], [2,2,1,1,2], [3,2,3,2,1], [2,1,3,2,1]. Тепер необхідно заново повторити алгоритм для нового рівня популяції.

Для оновленого нащадку №2 – індивід [2,2,1,1,2] сумарний час виконання запитів для буде дорівнювати

$$F = 9,1 + 4,6 + 5,6 + 6,1 = 25,2 \text{ с.}$$

$$\text{Фітнес} = \frac{1}{F} \approx 0,0397$$

У результаті генетичного алгоритму з елітарністю та адаптивним схрещенням було отримано найкращий індивід з загальним часом виконання запитів $F = 25,2$ секунд, а також знайдено варіант, що зменшує кількість вузлів з трьох до двох, що підтверджує ефективність використання даного методу, для задачі ребалансування даних у розподілених базах даних.

Висновки до третього розділу

У даному розділі було детально розглянуто та проаналізовано механізми ребалансування даних у високонавантажених системах, що мають на меті

зниження часу виконання запитів та оптимізацію використання її ресурсів. Огляд існуючих методів ребалансування включав використання статичних та динамічних підходів, зокрема біогеографічної оптимізації (BBO) та класичних генетичних алгоритмів. Ці методи дозволяють розподіляти дані між серверами таким чином, щоб зменшити час виконання запитів і знизити навантаження на окремі сервери, проте часто мають високу обчислювальну складність і можуть вимагати значної кількості обчислювальних ресурсів.

На основі оглянутих методів було удосконалено метод ребалансування даних, який використовує генетичні алгоритми з елітарністю та адаптивним схрещенням. У цьому підході елітарність гарантує збереження найкращих рішень у популяції, а адаптивне схрещення та мутація дозволяють динамічно вдосконалювати розподіл даних.

Такий підхід демонструє високу ефективність і доцільність для використання у високонавантажених розподілених системах, що вимагають швидкої та стабільної обробки запитів із мінімальними витратами на передачу даних і обчислювальні ресурси.

РОЗДІЛ 4 ПРОГРАМНІ ЗАСОБИ ЗАПРОПОНОВАНИХ МЕТОДІВ

4.1 Опис програмних засобів та структурних складових системи

4.1.1 Використані програмні засоби

Для дослідження дисертаційної роботи підвищення ефективності виконання запитів у високонавантажених системах було спроектовано та реалізовано програмне забезпечення з використанням програмних засобів, які інтегрують розроблені в межах дисертаційної роботи рішення. З метою забезпечення надійності та гнучкості розгортання ми застосували сучасний підхід на базі контейнеризації, що використовує технології контейнеризації Docker, а також мережеву архітектуру для взаємодії між сервісами бекенду, фронтенду та систем керування даними. В даному контексті наше середовище побудоване на Docker Compose(docker_compose.yaml) [41] та використовує низку сервісів, серед яких MySQL, Redis, Nginx, Laravel, а також додаткові технологічні рішення для забезпечення розподілу навантаження та кешування. Таке поєднання рішень дає змогу створити високо ефективне, масштабоване та ізольоване середовище для розробки та розгортання високонавантажених систем.

В процесі реалізації були використанні наступні програмні засоби:

- Docker – інструмент для контейнеризації, де на відміну від віртуальних машин, контейнери є менш затратними, оскільки вони спираються на ядро хостової операційної системи, але при цьому забезпечують достатню ізоляцію між додатками. Це суттєво спрощує розгортання та масштабування додатків, оскільки немає потреби в розгортанні окремих віртуальних машин для кожного сервісу.
- Nginx використовується як зворотній проксі-сервер і статичний веб-сервер. У типовій архітектурі Nginx приймає HTTP-запити від клієнтів та проксує їх на PHP-FPM. Nginx конфігурується через свій конфігураційний файл default.conf, який монтується у контейнер. Він визначає правило маршрутизації, статичні файли віддаються

безпосередньо Nginx, а динамічні запити до .php-файлів направляються на відповідний порт PHP-контейнера 9000.

- MySQL – реляційних систем керування базами даних. Розгорнувши її у контейнері, ми досягаємо ізоляції та повторюваності середовища. Том `db_data` закріплюється за контейнером MySQL, що дозволяє зберігати дані постійно, незалежно від перезапуску контейнера. Це забезпечує персистентність даних та простоту міграції. MySQL Container слухає на стандартному порту 3306.
- Laravel – PHP-фреймворк, орієнтований на швидку розробку веб-додатків з акцентом на чітку архітектуру MVC, зручні інтегровані засоби для маршрутизації, роботи з БД, керування сесіями та кешем. Laravel в контейнері означає, що весь бекенд-застосунок розміщено у власному оточенні зі своїми залежностями. У файлі Docker Compose Laravel-контейнер часто залежить від бази даних і Redis, тому що йому потрібні ці сервіси для повноцінної роботи. Він відображає свій порт 9000, що зазвичай використовується для зв'язку з Nginx, PHP-FPM прослуховує цей порт.
- Redis – ключ-значення сховище, завдяки йому можна зберігати тимчасові дані, токени сесій, опрацьовувати черги завдань та підвищувати продуктивність системи. Запуск Redis у окремому контейнері дозволяє ізолювати кеш від інших компонентів. Через змінні середовища або стандартні параметри Redis контейнер зберігає свої дані в томі `redis_data` використовуючи порт 6379.
- Supervisor використовується для керування воркерами черг. Черги дозволяють виконувати завдання у фоновому режимі без блокування основного потоку запитів користувача. Supervisor дозволяє автоматично перезапускати робітників у разі помилки, а також динамічно змінювати кількість працюючих процесів в залежності від навантаження.

Кожен контейнер виконується в ізольованому середовищі, що мінімізує вплив несправностей одного сервісу на інші рис 4.1. Крім того, використання

індивідуальних мереж Docker та закриття зовнішніх портів за межами необхідних, покращує безпеку системи, унеможливаючи несанкціонований доступ до внутрішніх сервісів.

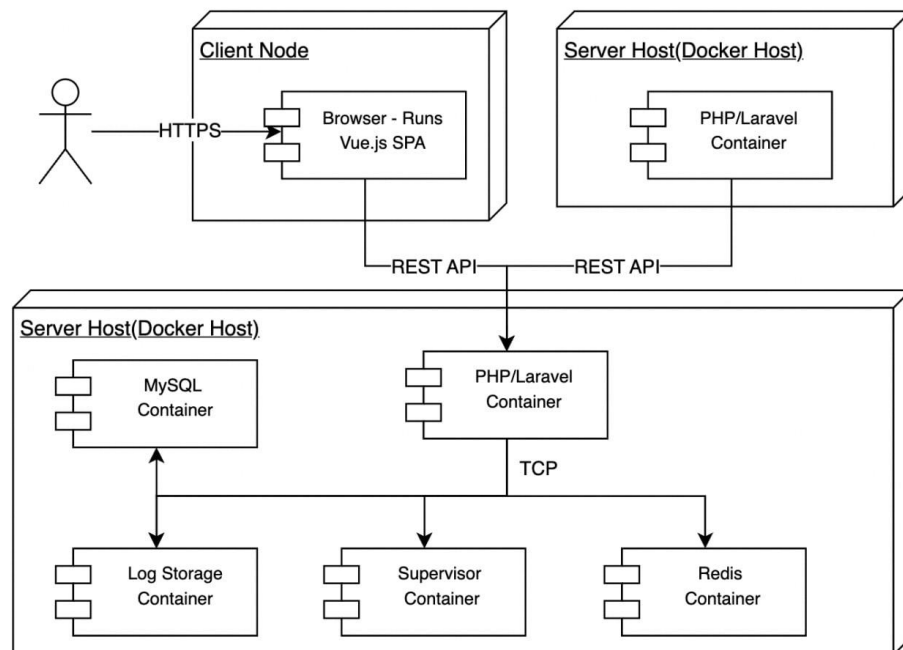


Рис 4.1 – UML діаграма розгортання

Запропонована архітектура дає змогу легко масштабувати компоненти. Наприклад, якщо навантаження зростає, можна розгорнути декілька контейнерів з PHP/Laravel та балансувати навантаження через Nginx. Redis можна конфігурувати у кластерному режимі задля підвищення відмовостійкості та обробки великих обсягів даних. Подібним чином MySQL можна замінити або доповнити репліками. Використання Docker та його екосистеми інструментів дозволяє вирішувати складні питання розміщення контейнерів, оновлень без простою, а також зменшення часу на розгортання.

4.1.2 Використані програмні засоби

Розподілена система складається з застосунку для користувачів та серверного додатку, який нараховує мінімум три вузли. Слід зазначити, що структура розподіленої системи буде змінюватися залежно від використовуваних методів. Для перших двох методів необхідне застосування Raft консенсус алгоритму, де всі вузли бази даних працюють узгоджено для досягнення консенсусу щодо стану системи.

Операції читання можуть виконуватися з будь-якого вузла, що підвищує продуктивність системи. У разі виходу лідера з ладу, алгоритм Raft автоматично обирає нового лідера з доступних вузлів, що забезпечує безперебійну роботу та високу доступність системи. На попередньому рис. 4.1 можна побачити UML діаграму розгортання, для дослідження першого та другого наукового результату.

Наступна UML діаграму розгортання відрізнятися від попередньої, оскільки тут застосовується метод ребалансування даних (рис. 4.2).

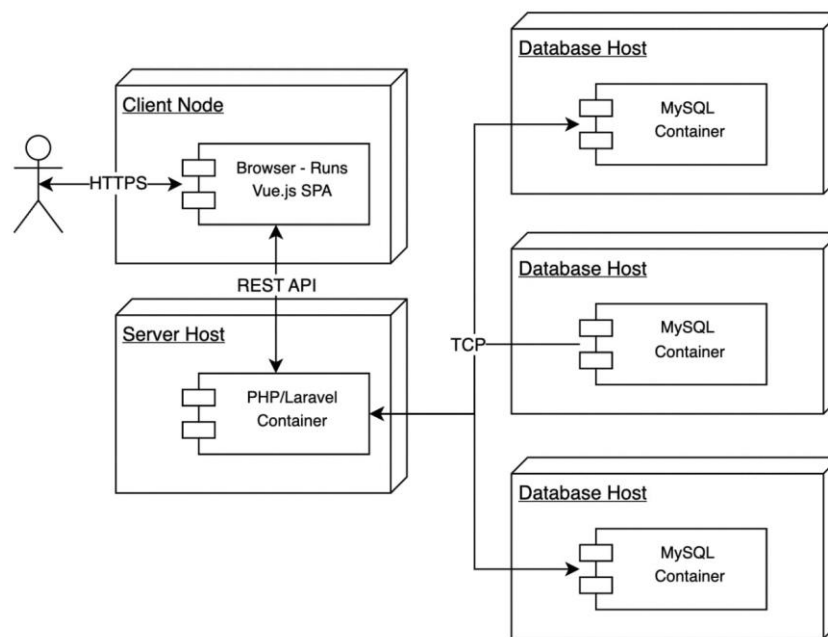


Рис 4.2 – Структурна схема з використанням методу ребалансування

Сервери підключаються до різних реплік, а самі репліки взаємодіють між собою утворюючи мережу.

4.1.3 Проєктування інтегрального середовища

Архітектура системи була ретельно спроектована, для забезпечення високої надійності, масштабованості та максимальної продуктивності в умовах інтенсивного використання. У своїй основі вона складається з чотирьох ключових рівнів (рис. 4.3), причому кожен із них утворює єдине цілісне середовище обробки даних. Завдяки такому підходу вдається ефективно розподілити навантаження між різними компонентами системи, зменшити ризик відмови окремих модулів і підвищити загальну стійкість. Крім того, чітке структурування за рівнями сприяє гнучкості та спрощує процес модернізації, що є надзвичайно важливим у

швидкозмінних умовах сучасних інформаційних систем. У результаті формується єдина система, здатна ефективно обслуговувати великі обсяги запитів, водночас зберігаючи стабільний рівень доступності й продуктивності для кінцевих користувачів.

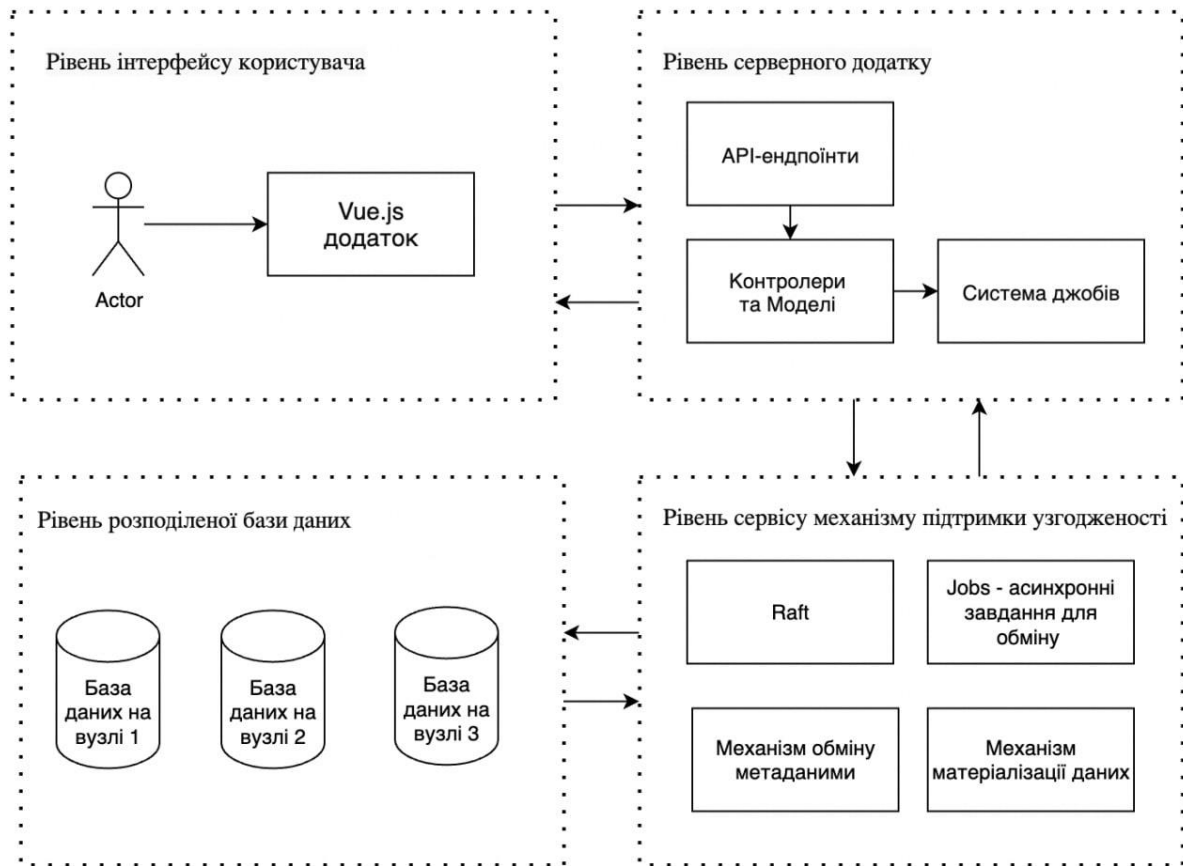


Рис 4.3 – Рівні розподіленої системи

Рівень розподіленої бази даних - на цьому рівні зберігаються всі дані. Розподілена база даних складається з кількох екземплярів баз даних, розміщених на різних вузлах системи. Такий підхід забезпечує:

- високу доступність даних - у випадку відмови одного з вузлів інші продовжують обслуговувати запити;
- масштабованість - додавання нових вузлів дозволяє обробляти більшу кількість запитів без втрати продуктивності;
- реплікацію даних - кожен вузол містить копію критично важливих даних, що забезпечує їх збереження та швидкий доступ.

Для узгодження даних між вузлами використовується Raft Consensus Algorithm, який гарантує консистентність та цілісність даних у розподіленому середовищі.

Рівень сервісу механізму підтримки узгодженості - цей шар відповідає за реалізацію механізмів, що забезпечують узгодженість даних між різними вузлами системи. Основні компоненти цього рівня:

- механізм консенсусу Raft використовується для досягнення узгодженості даних між вузлами шляхом вибору лідера, який координує всі записи та зміни;
- Jobs - асинхронні процеси, які забезпечують обмін повідомленнями та даними між вузлами. Вони відповідають за: попередній обмін метаданими та синхронізацію даних.

Механізм підтримки узгодженості працює прозоро для інших шарів системи, забезпечуючи цілісність даних без необхідності внесення змін до бізнес-логіки або інтерфейсу користувача.

Рівень серверного застосунку реалізує всю бізнес-логіку системи та відповідає за обробку запитів від користувачів. Основні функції:

- обробка CRUD-операцій - створення, читання, оновлення та видалення даних (оцінок, завдань, звітів тощо);
- валідація даних та перевірка коректності та повноти введеної інформації;
- аутентифікація та авторизація: забезпечення безпечного доступу користувачів до системи відповідно до їхніх ролей.

Рівень інтерфейсу користувача відповідає за взаємодію кінцевих користувачів із системою. Він реалізований за допомогою Vue.js, що забезпечує:

- динамічність та реактивність - швидке оновлення інтерфейсу без необхідності перезавантаження сторінки;
- зручність користування та інтуїтивно зрозумілий інтерфейс, для різних категорій користувачів;
- можливість роботи в реальному часі, оперативне відображення змін, внесених іншими користувачами.

Інтерфейс користувача взаємодіє з серверним додатком через API, що забезпечує чітке розмежування між фронтендом та бекендом і спрощує розвиток та підтримку системи.

Кожен з рівнів архітектури тісно пов'язаний з іншими, утворюючи цілісну систему. Інтерфейс користувача надсилає запити до серверного застосунку, який обробляє їх та взаємодіє з рівнем розподіленої бази даних для збереження або отримання необхідних даних. Серверний застосунок через рівень сервісу механізму підтримки узгодженості забезпечує синхронізацію даних між різними вузлами системи, використовуючи Raft Consensus Algorithm та Jobs. Рівень сервісу механізму підтримки узгодженості здійснює обмін метаданими та матеріалізацію результатів, що дозволяє мінімізувати мережевий трафік, зменшити час виконання запитів та підвищити швидкодію роботи системи.

Побудова системи на основі чотирьох рівневої архітектури забезпечує чітке розмежування функціональних обов'язків між різними компонентами. Кожен рівень взаємодіє з іншими через визначені інтерфейси, що сприяє модульності та гнучкості системи. Особливо важливим є шар серверного застосунку, який реалізує всю бізнес-логіку та відповідає за обробку запитів від користувачів.

Для детальнішого розуміння того, як саме реалізовано бізнес-логіку та взаємодію між компонентами, розглянемо UML-діаграму основних класів серверного застосунку. На цій діаграмі представлені основні контролери та їхні взаємозв'язки, що дозволяє краще зрозуміти структуру та функціональність системи.

В застосунку використано наступні класи:

- RaftController – реалізує обробку запитів, пов'язаних з роботою алгоритму консенсусу Raft. Він відповідає за координацію процесів узгодженості даних між вузлами системи.
- NodeController – керує взаємодією між вузлами системи. Забезпечує обмін повідомленнями, синхронізацію станів та обробку відмов.
- MetadataController – відповідає за обмін метаданими між вузлами. Це дозволяє мінімізувати обсяг переданих даних та оптимізувати роботу мережі.

- TeachersController – відповідає за всі операції, пов’язані з викладачами. Він обробляє запити на створення нових викладачів, оновлення їхньої інформації, призначення предметів та управління розкладом занять.
- StudentsController – займається обробкою запитів, пов’язаних зі студентами. Контролер дозволяє реєструвати нових студентів, оновлювати їхні дані, переглядати успішність та керувати їхнім доступом до системи.
- ParentsController – забезпечує функціональність для батьків студентів. За допомогою цього контролера батьки можуть отримувати інформацію про академічну успішність своїх дітей, переглядати оцінки та спілкуватися з викладачами.
- AdminsController – контролер адміністратора, який має доступ до управління всією системою. Він може створювати та редагувати користувачів, налаштовувати системні параметри, генерувати звіти та статистику.
- ScheduleController – відповідає за управління розкладом занять. Дозволяє створювати та редагувати розклади, призначати аудиторії та забезпечувати доступність інформації для студентів та викладачів.
- GroupsController – займається управлінням академічними групами. Контролер дозволяє створювати групи, додавати студентів, призначати кураторів та керувати пов’язаною інформацією.
- DiaryController – реалізує функціонал електронного щоденника. Він дозволяє викладачам додавати домашні завдання, а студентам – переглядати та здавати їх. Також забезпечує зворотний зв’язок між викладачами та студентами.
- LessonController – контролер для роботи з заняттями. Викладачі можуть створювати заняття з відповідним типом, а також вносити домашнє завдання.
- ScoresController – контролер для роботи з оцінками. Викладачі можуть виставляти оцінки, а студенти та батьки – переглядати їх.

4.2 Програмна реалізація методів

4.2.1 Модифікований метод Raft Consensus Algorithm

Для дослідження було реалізовано програмні засоби, що забезпечують ефективне виконання запитів шляхом попереднього обміну метаданими між вузлами та локальним збереженням отриманих результатів (рис. 4.4). Система складається з декількох вузлів, кожна з яких має локальну базу даних MySQL та кеш-пам'ять Redis. Вузли взаємодіють між собою через мережу та використовують Leaderless Replication та алгоритм Raft для досягнення консенсусу та забезпечення узгодженості даних. Основні компоненти системи:

- лідер та послідовники - у рамках алгоритму Raft один з вузлів виконує роль лідера, інші — послідовників;
- журнал використовується для збереження команд, які мають бути виконані на всіх вузлах, забезпечуючи узгодженість даних.
- кешування за допомогою Redis використовується для зберігання метаданих та міток стану, що дозволяє зменшити кількість звернень до лідера.

У системі реалізовано механізм автоматичного вибору лідера згідно з алгоритмом Raft. При запуску всі вузли є послідовниками. Якщо послідовник не отримує heartbeats від лідера протягом визначеного тайм-ауту, він переходить у стан кандидата та ініціює вибори. Лідер періодично надсилає heartbeats до послідовників для підтримки лідерства та синхронізації журналів. Таким чином, Raft надає надійну й водночас відносно просту модель керування станами у розподілених системах. Постійні heartbeats лідера і чітко визначені механізми перевірки тайм-аутів та термів дають змогу швидко реагувати на збої або втрату зв'язку з одним із вузлів, ініціювати вибори нового лідера та відновлювати нормальне функціонування кластера без суттєвої затримки чи ризику дублювання ролей.

Алгоритм роботи Raft показано на рис., де детально ілюструється послідовність станів вузлів, обмін повідомленнями та механізми узгодження роботи Raft зображено на рис .

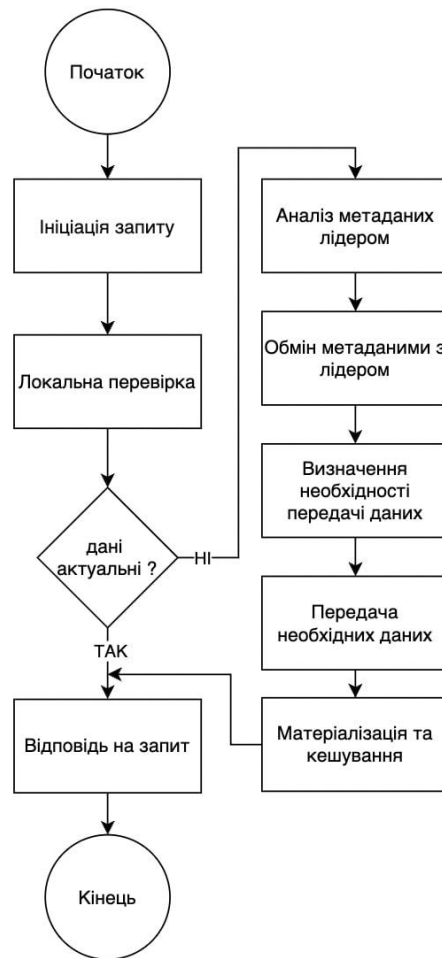


Рис 4.4 – Алгоритм роботи Raft розподіленої системи

Метод базується на ідеї, що перед передачею значних обсягів даних між вузлами доцільно спочатку обмінятися метаданими про їхній стан. На основі таких метаданих можна виявити, які саме фрагменти даних потребують актуалізації, та визначити мінімально необхідний обсяг передачі. Це дозволяє суттєво скоротити мережевий трафік, оскільки у мережу надсилається лише та частина даних, що зазнала змін чи доповнень.

Для забезпечення ефективною мінімізації мережевого трафіку в алгоритмі Raft використовуються різні типи метаданих, включаючи хеш-суми, кардинальність, мітки часу та версії даних наведені в таблиці 4.1. Метадані можуть бути передані в заголовках headers HTTP-запиту або через спеціалізовані повідомлення між вузлами. Хеш-суми дозволяють швидко перевіряти цілісність та актуальність даних між вузлами без необхідності передавати повні дані. Використання метаданих про кардинальність допомагає визначати різницю в кількості записів, що сигналізує про можливі зміни.

Таблиця 4.1 – Дані для методу мінімізації мережевого трафіку

Метадані	Опис	Заголовки
Хеш-сума	Хеш-сума даних для перевірки цілісності та актуальності інформації.	X-Raft-Hash: {хеш_сума}
Кардинальність	Кількість записів у відповідному сегменті даних (для порівняння кількості даних).	X-Raft-Cardinality: {кількість_записів}
ID вузла	Унікальний ідентифікатор вузла, яка відправляє дані.	X-Raft-Node-ID: {ID_вузла}
Мітка UTC часу	Час останнього оновлення даних, що дозволяє визначити актуальність інформації.	X-Raft-Timestamp: {мітка_часу}

Механізм прийняття рішень на основі порівняння цих метаданих забезпечує точну передачу лише тих даних, які дійсно потребують синхронізації, що суттєво знижує навантаження на мережу.

4.2.2 Модифікований метод узгодженості на основі алгоритму Левенштейна

Виходячи з реалізації оптимізації мережевого трафіку в системах Raft, основним завданням залишається забезпечення узгодженості даних між вузлами, особливо в умовах високої частоти змін. Традиційні підходи до синхронізації передбачають передачу повних копій даних між вузлами, що призводить до значного навантаження на мережу. У відповідь на це була розроблена метод оптимізації, який використовує передачу лише необхідних дельт між версіями даних, а не повного обсягу інформації. Це значно знижує навантаження на мережу і підвищує ефективність роботи розподілених систем.

Модифікований метод узгодженості на основі алгоритму Левенштейна з ваговими коефіцієнтами був інтегрований у попередню архітектуру, де за

допомогою Observers спочатку визначаються дані які необхідно змінити, а далі з допомогою Job виконується передача даних, що потребують оновлення рис 4.5.

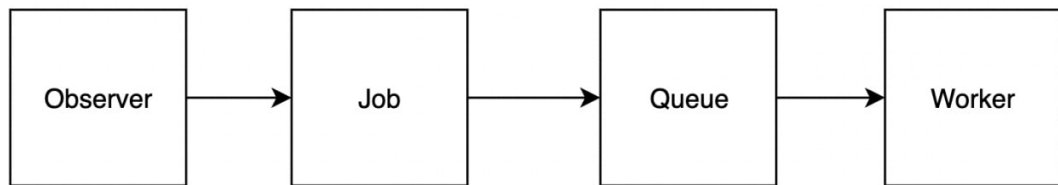


Рис 4.5 – Схема оновлення даних

У цьому підході використовується додатковий рівень контролю над "вартістю" операцій редагування за допомогою вагових коефіцієнтів(4.4), що дозволяє налаштувати оптимальний баланс між операціями вставки, видалення і заміни. Коли дані змінюються в одному з вузлів, Job передає лише ті сегменти даних, які зазнали змін, використовуючи оптимізований метод Левенштейна з налаштованими ваговими коефіцієнтами. Це дозволяє уникнути передачі зайвих даних і знижує обсяг мережевого трафіку, необхідного для синхронізації.

Журналізація є ключовим компонентом алгоритму Raft, який забезпечує узгодженість даних між вузлами в розподіленій системі. Завдяки журналізації кожен вузол зберігає послідовність команд, які були виконані або мають бути виконані, забезпечуючи можливість відновлення стану системи у випадку відмови або втрати даних. Основною метою журналу є зберігання всіх операцій, які виконуються системою, щоб кожен вузол могла застосовувати ці операції у правильному порядку. Це гарантує, що всі вузли мають однаковий стан і можуть виконувати запити користувачів, навіть якщо одна або кілька вузлів виходять з ладу.

Було встановлено, що для різних типів даних найефективнішими є різні вагові коефіцієнти при обчисленні відстані Левенштейна. Зокрема, числові дані, такі як оцінки, текстові поля на зразок домашніх завдань, або незначні зміни у прізвищах користувачів потребують різних налаштувань вартості операцій вставки, видалення та заміни. Щоб забезпечити максимальну ефективність передачі змін і зменшити навантаження на мережу, було прийнято рішення розробити функціонал для динамічного визначення та зберігання оптимальних

вагових коефіцієнтів залежно від типу даних і моделі, у якій ці дані змінюються. Це дозволяє адаптувати алгоритм узгодженості до особливостей кожного типу даних, забезпечуючи високу продуктивність і ефективне використання мережевого трафіку.

Для реалізації підходу з динамічним визначенням вагових коефіцієнтів для різних типів даних було прийнято рішення створити спеціальну структуру зберігання та інструмент для автоматичного розрахунку оптимальних значень.

Зокрема, було створено таблицю `weight_coefficients` з полями: `id`, `model`, `weight_delete`, `weight_insert`, `weight_replace`. В цій таблиці зберігаються вагові коефіцієнти для різних моделей даних. Кожен запис у цій таблиці включає назву моделі і відповідні вагові коефіцієнти для операцій вставки, видалення та заміни. Це забезпечує можливість зберігати окремі налаштування для кожного типу даних, враховуючи специфіку його змін.

З метою автоматизації процесу визначення оптимальних вагових коефіцієнтів у розподілених системах була використана функціональність фреймворка `Laravel`, зокрема розширені можливості інструменту `Artisan` для створення спеціальної команди *`OptimizeWeightCoefficients`* (рис. 4.6). Ця команда в режимі реального часу виконує складні обчислення вагових коефіцієнтів для різних категорій даних і типів моделей, попередньо враховуючи низку параметрів, включно зі структурою бази даних, частотою оновлень та інтенсивністю запитів. Подібна гнучкість у конфігуруванні забезпечує високу адаптивність системи, оскільки дозволяє швидко змінювати значення ваг, реагуючи на зміну умов навантаження або специфічні вимоги експлуатації.

Оперативне оновлення коефіцієнтів у сукупності з надійними механізмами розподіленої синхронізації сприяє більш точному узгодженню даних між вузлами мережі, підтримуючи високий рівень консистентності навіть за умови значних обсягів транзакцій. Завдяки цьому підходу система не лише демонструє кращі показники продуктивності та стійкості, але й здатна максимально ефективно впроваджувати додаткові оптимізації без потреби суттєвої перебудови архітектури або ручного втручання адміністратора.

```

private function computeLevenshteinWithWeights($oldString, $newString, $w_del, $w_ins, $w_rep): array
{
    $oldLen = mb_strlen($oldString);
    $newLen = mb_strlen($newString);

    $matrix = array_fill( start_index: 0, count: $oldLen + 1, array_fill( start_index: 0, count: $newLen + 1, value: 0));

    for ($i = 0; $i <= $oldLen; $i++) {
        for ($j = 0; $j <= $newLen; $j++) {
            if ($i == 0) {
                $matrix[$i][$j] = $j * $w_ins;
            } elseif ($j == 0) {
                $matrix[$i][$j] = $i * $w_del;
            } else {
                $cost = ($oldString[$i - 1] == $newString[$j - 1]) ? 0 : $w_rep;
                $matrix[$i][$j] = min(
                    value: $matrix[$i - 1][$j] + $w_del,
                    ...values: $matrix[$i][$j - 1] + $w_ins, |
                    $matrix[$i - 1][$j - 1] + $cost
                );
            }
        }
    }

    return $matrix[$oldLen][$newLen];
}

```

Рис 4.6 – Частина коду реалізації команди визначення вагових коефіцієнтів

4.2.3 Метод ребалансування з використанням генетичних алгоритмів

З урахуванням змін в архітектурі розподіленої системи, де тепер кожна сутність зберігається на різних вузлах, виникає необхідність оптимізації розподілу даних та ефективної синхронізації між вузлами. Для вирішення цієї проблеми застосовується метод оптимізації на основі генетичного алгоритму з елітарністю та адаптивним схрещенням. Цей метод спрямований на знаходження оптимального розподілу сутностей по вузлах з метою мінімізації загального часу виконання запитів та зменшення мережевого навантаження.

Генетичний алгоритм з елітарністю та адаптивним схрещенням використовує елітарність для збереження найкращих рішень у кожному поколінні, що забезпечує швидшу конвергенцію до оптимального розподілу. Адаптивні параметри схрещення та мутації дозволяють динамічно змінювати ймовірності генетичних операторів в залежності від поточного стану популяції, підвищуючи ефективність пошуку та уникнення локальних оптимумів.

У новій архітектурі, де сутності розподілені по різних вузлах, запити можуть вимагати доступу до кількох сутностей одночасно. Оптимальний розподіл даних дозволяє мінімізувати кількість між вузлових взаємодій, що знижує затримки та

навантаження на мережу. Крім того, метод забезпечує ефективну синхронізацію змін між вузлами, передаючи лише необхідні дельти, а не повні копії даних.

Таким чином, використання генетичного алгоритму з елітарністю та адаптивним схрещенням у новій архітектурі розподіленої системи сприяє:

- Підвищенню продуктивності системи за рахунок швидшого виконання запитів.
- Зменшенню мережевого трафіку шляхом оптимізації передачі даних між вузлами.
- Покращенню масштабованості системи, що є критичним для великих розподілених баз даних.
- Забезпеченню консистентності даних через ефективну синхронізацію змін між вузлами.

Цей підхід дозволяє більш ефективно використовувати ресурси системи та покращує загальний користувацький досвід завдяки швидшому доступу до даних та стабільній роботі системи навіть при високому навантаженні.

4.3 Генерація тестових даних

Для генерування тестових даних було використано програмне забезпечення на основі фреймворку Laravel, зокрема механізми Seeders та Factories, які дозволяють швидко та зручно створювати великі обсяги фіктивної інформації, для різних сутностей системи.

Спочатку за допомогою Factories визначалися шаблони створення сутностей, такі як користувачі, групи, заняття чи оцінки. У цих класах визначалися поля, типи даних, випадкові значення та залежності. Це давало змогу легко генерувати тисячі екземплярів сутностей із різними параметрами.

Наступним кроком було використання Seeders, які безпосередньо виконують масове створення записів у базі даних на основі описаних у Factories шаблонів. Через seed-команди можна створити різні сценарії наповнення даними: наприклад, згенерувати 1000 користувачів, 2000 оцінок або 300 уроків, швидко змінюючи лише параметри у коді класу.

Окремою важливою задачею було розподіл згенерованих даних між різними вузлами системи. Завдяки гнучкому керуванню конфігураціями Laravel та специфічним artisan-командам, можна було налаштувати наповнення не лише локальної бази, а й розподіл даних між віддаленими вузлами. Таким чином, кожен вузол отримував певну кількість сутностей, що дозволяло емулювати реальну роботу розподіленої системи, де певні дані зберігалися на одному вузлі, інші — на другому чи третьому. Це дозволило в умовах тестування відтворити реальні сценарії роботи системи, такі як різний ступінь навантаження, перевірку узгодженості даних та ефективності методів оптимізації мережевого трафіку.

Висновки до четвертого розділу

Представлений комплексний підхід до проєктування та реалізації розподіленої системи, орієнтованої на високонавантажені умови роботи, демонструє ефективну інтеграцію сучасних програмних засобів та методів оптимізації.

Використання контейнеризації (Docker), балансування навантаження, реляційної СКБД, кешування дозволило створити гнучке, масштабоване та ізольоване середовище для розробки, розгортання та експлуатації системи.

Структурне розділення архітектури на логічні рівні – від розподіленої бази даних, що забезпечує високу доступність та цілісність, до рівня сервісу механізму підтримки узгодженості (Raft та Jobs, серверного застосунку з чіткою бізнес-логікою та інтерфейсу користувача на Vue.js – сприяє кращій керованості та модульності проєкту.

Застосування Raft Consensus Algorithm та ефективне використання метаданих дало змогу мінімізувати мережевий трафік і покращити узгодженість даних, що особливо важливо для великих розподілених систем.

Динамічне визначення вагових коефіцієнтів для алгоритму Левенштейна, а також впровадження генетичного алгоритму з елітарністю та адаптивним схрещенням для ребалансування даних забезпечили не лише оптимізацію

синхронізації інформації, але й підвищену продуктивність та ефективність роботи системи загалом.

Ці підходи дозволяють краще використовувати мережеві та обчислювальні ресурси, адаптуватися до умов постійно зростаючих обсягів даних, частих оновлень та високого навантаження. Таким чином, розроблене середовище та впроваджені методики не лише покращують час відгуку системи та стабільність її роботи, але й створюють основу для подальшого розвитку, модифікацій та розширень. Результати дослідження мають потенціал для масштабування та адаптації до інших класів розподілених систем, сприяючи підвищенню ефективності сучасних інформаційних технологій та забезпеченню високоякісного користувацького досвіду.

РОЗДІЛ 5 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЗАПРОПОНОВАНИХ МЕТОДІВ

5.1 Розробка інтегрального середовища системи онлайн-журналу

Проведення досліджень з використанням розподіленої системи онлайн-журналу обумовлено тим, що вона має максимально та надійно працювати, підтримувати узгодженість даних та бути відмово стійкою, оскільки це впливає на оперативність навчального процесу як для викладачів, так і для студентів та батьків. Система онлайн-журналу розподілена на різних куточках країни, по різних навчальних закладах, що вимагає спеціалізованих методів та технологій для вузлу системи. Ведення розкладу, виставлення оцінок, внесення домашнього завдання в кінці кожного уроки можуть відбуватися одночасно з різних пристроїв та локацій, але це не повинно впливати на швидкість відображення цих даних для всіх ролей системи. Внесення нової інформації має відбуватися якомога швидше, щоб підтримувати безперервність та стабільність навчального процесу.

Системи онлайн-журналів, як було згадано раніше, повинні відповідати високим стандартам. Це зумовлено не лише потребами кінцевих користувачів, а й необхідністю забезпечення швидкого доступу до інформації та швидкої реакції на зміни в освітньому процесі. Розподіл даних і реплікація дозволяють створити окремі вузли з копіями даних для розподілу навантаження, що покращує продуктивність системи.

Одним із можливих підходів до реалізації такої системи є використання архітектури на базі алгоритму консенсусу Raft. Raft забезпечує розподілений консенсус у кластерах, що дозволяє досягти високої доступності та узгодженості даних. У цій архітектурі система складається мінімум з трьох вузлів, де один виконує роль лідера, а інші є послідовниками або кандидатами. Лідер відповідає за прийом і обробку запитів від користувачів.

Для онлайн-журналу це означає, що всі операції внесення чи оновлення інформації проходять через лідера та узгоджуються між усіма вузлами. Якщо лідер виходить з ладу, алгоритм Raft автоматично обирає нового лідера з послідовників,

що забезпечує безперебійну роботу системи. Це підвищує надійність і стійкість системи до відмов.

Клієнтський застосунок взаємодіє з лідером, але в разі його недоступності може перенаправити запити до нового лідера без втрати даних чи функціональності. Такий підхід дозволяє забезпечити швидкий доступ до актуальної інформації, зберігаючи при цьому узгодженість даних у розподіленому середовищі. Це особливо важливо для освітніх установ, де оперативність і точність інформації є критичними для ефективного навчального процесу.

5.1.1 Бізнес-логіка інтегрального середовища системи онлайн-журналу

Бізнес-логіка електронного онлайн-журналу починається з того, що в системі реєструється користувач з роллю адміністратора. Під час реєстрації він вносить свої персональні дані, такі як прізвище, ім'я, по батькові, електронну адресу та надійний пароль для доступу до системи. Адміністратор є ключовою фігурою в налаштуванні платформи, оскільки саме він відповідає за початкове заповнення бази даних та надання доступу іншим користувачам.

Після успішної реєстрації адміністратор переходить до створення навчальних закладів та облікових записів для їх керівників. Він вводить детальну інформацію для кожного керівника, включаючи їхні контактні дані, посаду та інші важливі відомості. Додатково адміністратор системи вносить інформацію про всі предмети, які викладаються в навчальних закладах, та їхні типи. Це можуть бути контрольні роботи, практичні заняття, лекції, семінари та інші форми навчання. Він також додає інформацію про канікули, державні свята та інші важливі дати на поточний навчальний рік. Це допомагає в коректному плануванні розкладу та уникненні конфліктів при проведенні занять.

Керівники після активації своїх облікових записів отримують доступ до персонального кабінету, де можуть керувати інформацією свого закладу та налаштовувати систему відповідно до потреб установи. Отримавши доступ до системи, керівники навчальних закладів починають роботу у своїх особистих кабінетах. Першим кроком вони вносять детальну інформацію про всі класи або групи, залежно від структури навчання в їхньому закладі. Наступним етапом є

внесення користувачів з відповідною інформацією для кожної ролі. Керівники додають вчителів, студентів та батьків, забезпечуючи кожному користувачу відповідний рівень доступу та функціонал у системі.

Для вчителів вводяться їхні спеціалізації та предмети, які вони викладають. Для студентів — особисті дані, група та інша важлива інформація. Батьки отримують облікові записи, пов'язані з їхніми дітьми, що дозволяє їм слідкувати за успішністю та відвідуваністю. Потім керівники переходять до створення розкладу для кожної групи чи класу(відповідно до навчального закладу). Вони детально налаштовують розклад занять, вказуючи дні тижня, час проведення уроків, призначених викладачів та аудиторії. Система дозволяє налаштовувати розклад з можливістю варіації по тижнях (наприклад, з першого по четвертий тиждень), що забезпечує гнучкість у плануванні та врахуванні особливостей навчального процесу.

Важливим етапом є підв'язка студентів до їхніх батьків. Це здійснюється шляхом встановлення зв'язку між обліковими записами студентів та батьків у системі. Завдяки цьому, батьки, заходячи у свій особистий кабінет, можуть отримувати актуальну інформацію про успішність, відвідуваність, отримані оцінки та домашні завдання своїх дітей. Вони також можуть спілкуватися з викладачами та адміністрацією через систему повідомлень.

Після виконання всіх вищезазначених кроків система готова до повноцінного використання, і можна розпочинати навчальний процес. Вчителі отримують можливість вносити оцінки, створювати та призначати домашні завдання, а також спілкуватися зі студентами та батьками. Студенти можуть переглядати свій розклад, отримувати завдання та слідкувати за своєю академічною успішністю. Батьки мають змогу бути в курсі навчання своїх дітей, що сприяє більш тісній взаємодії між навчальним закладом та сім'єю. Таким чином, детальне налаштування та заповнення системи на початкових етапах забезпечує її ефективне функціонування надалі. Короткий варіант функціональних схем зображено на рис.5.1.



Рис 5.1 – Функціональна схема роботи системи

5.1.2 Реалізація серверного застосунку

Серверний застосунок можна умовно поділити на три основні шари: шар веб-сервісу для взаємодії, сервісний шар та шар взаємодії з розподіленою базою даних. Шар веб-сервісу виступає інтерфейсом для взаємодії із системою та забезпечує клієнт-серверне спілкування за допомогою протоколу прикладного рівня HTTP. Для передачі корисного навантаження використовується формат JSON, що дозволяє легко та зручно оперувати даними.

До головних маршрутів можна віднести наступні:

- *admins, students, teachers, parents, managers*. Цей маршрут використовується для управління інформацією про користувачів з відповідною роллю у системі. До ключової інформації відносяться прізвище, ім'я, по-батькові, електронна пошта та інші додаткові поля відповідно до кожної ролі. При використанні методів GET, PUT та DELETE необхідно вказати ID об'єкта в заголовках HTTP запиту. Підтримує HTTP методи GET, POST, PUT, DELETE.
- *groups*. Використовується для управління інформацією про групи у системі. До ключової інформації відносяться назва, номер класу, ідентифікатор викладача та ідентифікатор пов'язаної групи. При використанні методів GET, PUT та DELETE необхідно передавати ID об'єкта в заголовках HTTP-запиту. Підтримує HTTP-методи GET, POST, PUT, DELETE.

- *lessons*. Використовується для управління інформацією про заняття у системі. До ключової інформації відносяться ідентифікатор групи, ідентифікатор дисципліни, назва, дата, зміст, домашнє завдання, позначка нової теми та ідентифікатор типу. При використанні методів GET, PUT та DELETE необхідно передавати ID об'єкта в заголовках HTTP-запиту. Підтримує HTTP-методи GET, POST, PUT, DELETE.
- *disciplines, lesson-types, holidays, posts, instructions*. Використовується для управління інформацією про довідники: дисципліни, типи уроків, канікули, новини та інструкції у системі. До ключової інформації відноситься назва та інші додаткові поля. При використанні методів PUT, GET та DELETE необхідно передавати ID об'єкта в заголовках HTTP-запиту. Підтримує HTTP-методи POST, GET, PUT, DELETE.

Шар взаємодії з розподіленою базою даних реалізовано у двох варіантах для кожного методу (алгоритму консенсусу Raft та методу ребалансування). Це необхідно, оскільки кожен з цих методів має власну логіку взаємодії.

Шар сервісу в нашому проєкті виконує функцію інтерфейсу для внутрішньої взаємодії застосунку та реалізації бізнес-логіки системи. Він спроектований таким чином, щоб забезпечити гнучкість у використанні різних методів підвищення ефективності виконання запитів до розподіленої бази даних. Це необхідно, оскільки система призначена для дослідження двох різних архітектурних рішень.

5.1.3 Графічний інтерфейс реалізованого середовища

Графічний інтерфейс системи було розроблено з використанням сучасного JavaScript-фреймворку Vue.js, який забезпечує високу продуктивність, гнучкість та зручність у створенні інтерактивних та динамічних веб-додатків. Завдяки Vue.js користувачі отримують швидкий відгук інтерфейсу, що підвищує загальний рівень задоволеності роботою з додатком. Для створення розмітки та стилізації компонентів використовуються HTML і CSS, що дозволяє зберігати структурованість та послідовність у розташуванні візуальних елементів. Це також сприяє адаптивності інтерфейсу для коректного відображення на різних пристроях.

На рисунку 5.2 та 5.3 представлено інтерфейс електронного щоденника для студента, де відображається розклад занять на тиждень.

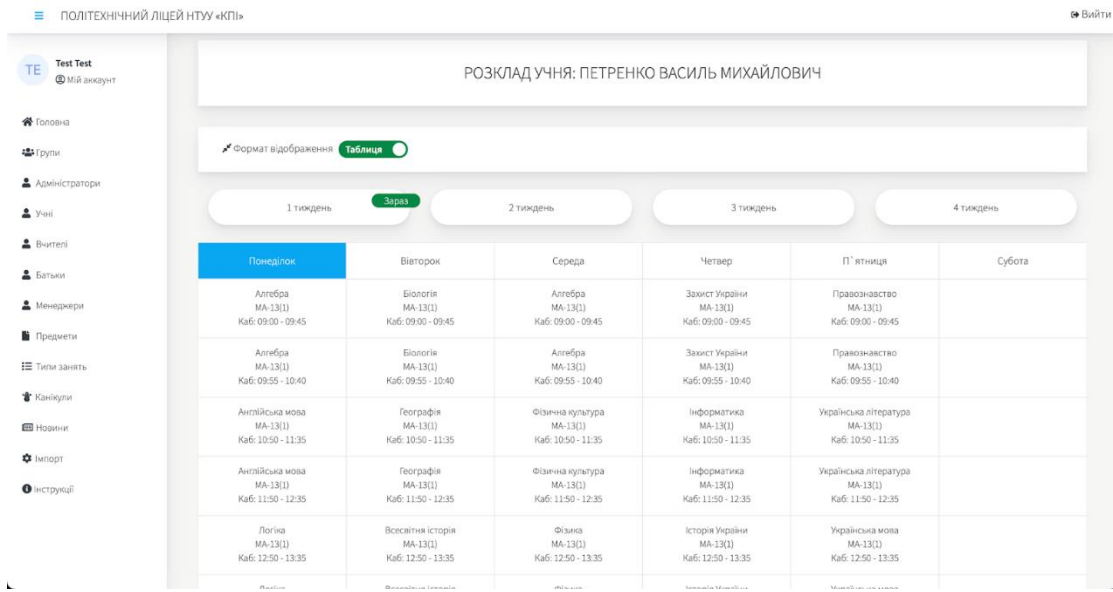


Рис 5.2 – Сторінка розкладу

Кожен урок показано із зазначенням предмету, часу проведення та аудиторії. Інтерфейс також включає можливість перемикатися між тижнями, а формат відображення може змінюватися за допомогою відповідного перемикача. Ліворуч розташоване меню навігації, яке забезпечує швидкий доступ до основних розділів системи, таких як групи, учні, вчителі та інші.

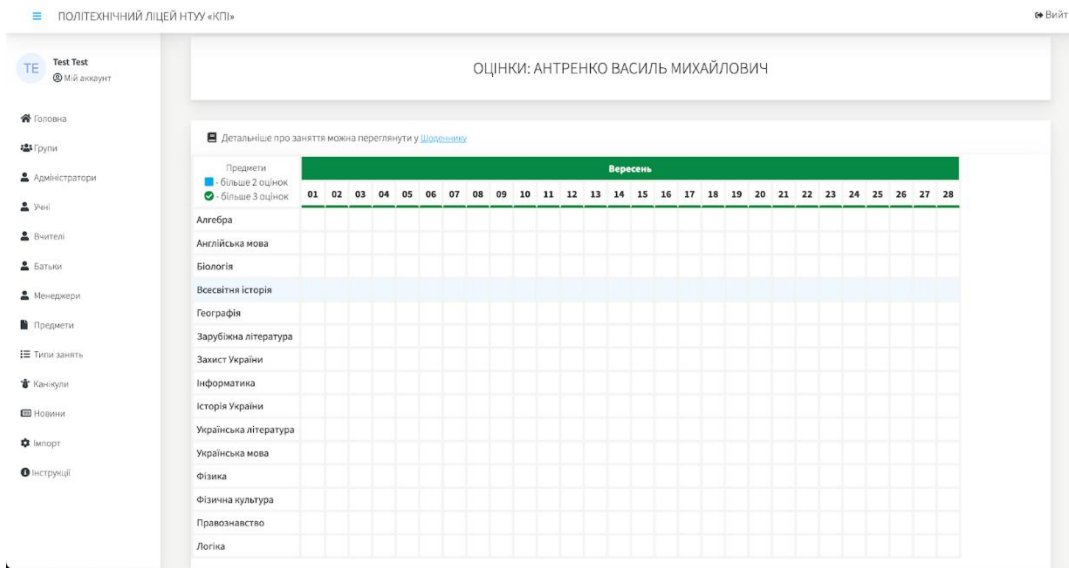


Рис 5.3 – Сторінка журналу успішності

На рисунку 5.4 показано журнал успішності студента, який демонструє результати його оцінок за предметами протягом місяця. У верхній частині

представлений календар із числами місяця (вересень), де відображаються дати оцінок.

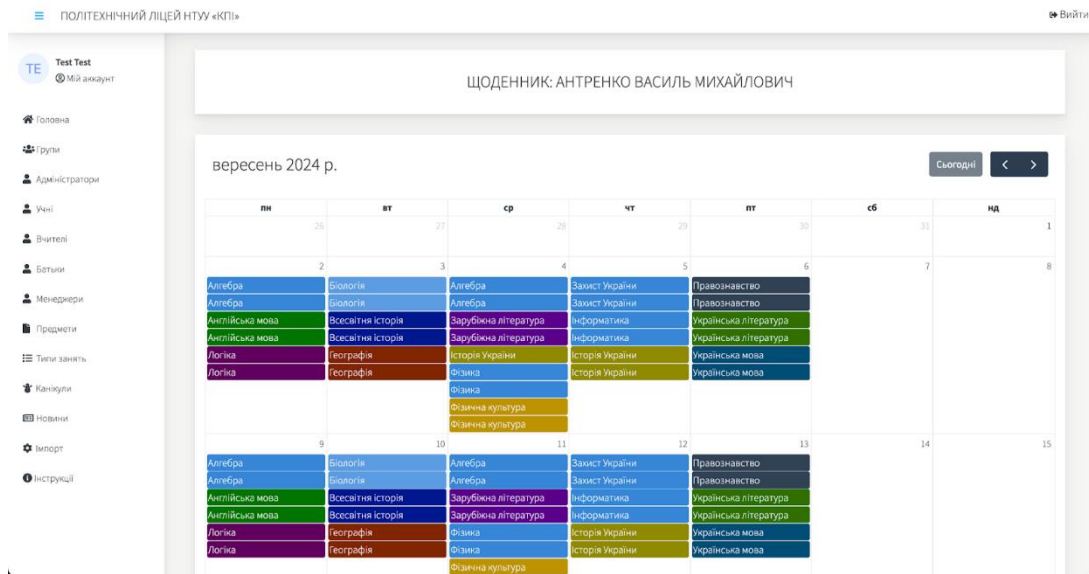


Рис 5.4 – Сторінка інформації про щоденник

По вертикалі вказані предмети. Інтерфейс дозволяє легко відстежувати кількість оцінок за кожним предметом. Журнал має інтуїтивно зрозумілу структуру, що полегшує перегляд академічних результатів студента за конкретний період часу.

5.2 Експериментальні дослідження методів

5.2.1 Метод мінімізації мережевого трафіку в Raft

Для більш глибокої та об'єктивної оцінки ефективності розробленого методу мінімізації мережевого трафіку при виконанні запитів у розподіленій системі, що застосовує алгоритм консенсусу Raft, було проведено розширену серію з 10 експериментів. У кожному з цих експериментів варіювалися параметри, пов'язані з кількістю унікальних груп на кожному вузлі, а також з параметрами запитів до системи. Обрана стратегія вимірювань передбачала випадкове формування обсягів і складу даних на різних вузлах та використання різних запитів, що імітують різноманітні сценарії роботи реальних розподілених систем. Такий підхід дозволяє не лише узгодити результати з різноплановими умовами експлуатації, але й виявити особливості поведінки системи при високому навантаженні, нерівномірному розподілі даних між вузлами та раптових змінах конфігурації.

Випадковість у виборі величин і характеристик даних створює умови, максимально наближені до реальних, де системи часто зазнають непередбачуваних коливань навантаження, варіацій у кількості активних користувачів та змін структури даних. Завдяки цьому можна точніше простежити динаміку впливу обраного методу на середній час відповіді, стабільність передачі даних та ефективність використання мережевих ресурсів.

Тестове середовище складається з трьох реплік та для тестування використовувалась наступна процедура з однією з головних сутностей Group:

- кількість унікальних груп на кожному вузлі: випадкове число від 50 до 200;
- продубльовано деякі групи на інших вузлах;
- ініціалізується запит на отримання всіх груп;
- методи виконання запитів - без використання методу(пряма передача повних даних від лідера до вузлі-ініціатора) та з використанням методу(попередній обмін метаданими, визначення необхідних даних та подальшому локальному збереженні результатів на вузлі-ініціаторі);
- отримати дані та загальну інформацію по запиту.

Результати експериментів наведено на рисунку 5.5.

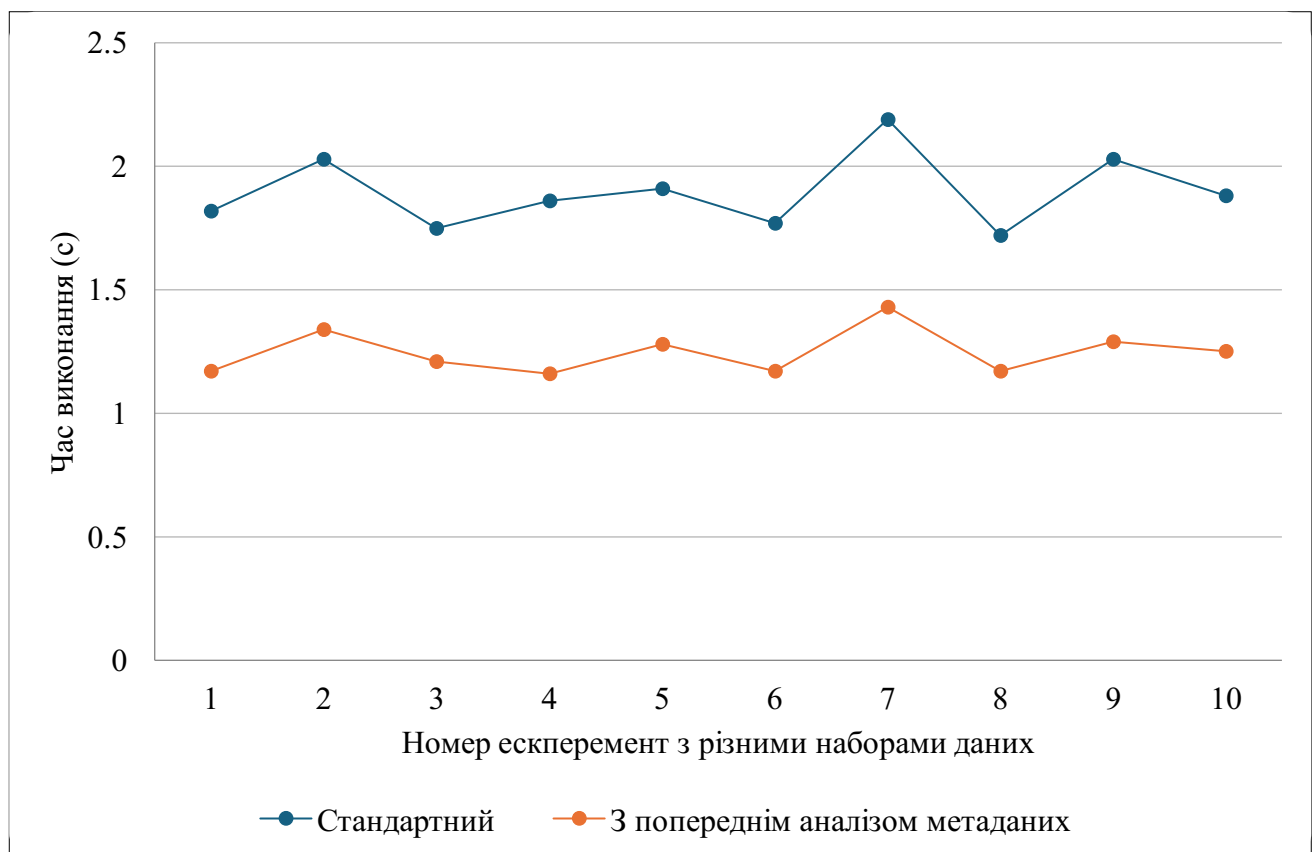


Рис 5.5 – Середній час виконання запитів для 3 вузлів

Крім попередньо описаних випробувань у середовищі з трьома вузлами, було проведено додаткову серію експериментів, у яких кількість вузлів у розподіленій системі збільшено до п'яти. Використання п'яти вузлів дозволило дослідити такі аспекти, як стійкість алгоритму до потенційних дисбалансів у наборі груп, зміна тенденцій у кількості передаваних даних і вплив цього на загальний час отримання відповідей.

Такий підхід підкреслює здатність методу ефективно функціонувати за умов розширеної інфраструктури, в якій, порівняно з трьома вузлами, збільшується різноманітність сценаріїв та потенційна складність узгодження даних. Аналіз результатів цих додаткових експериментів дозволяє ширше зрозуміти можливості та обмеження розробленого методу в реальних високонавантажених середовищах, наближених до промислових застосувань. (рис. 5.6).

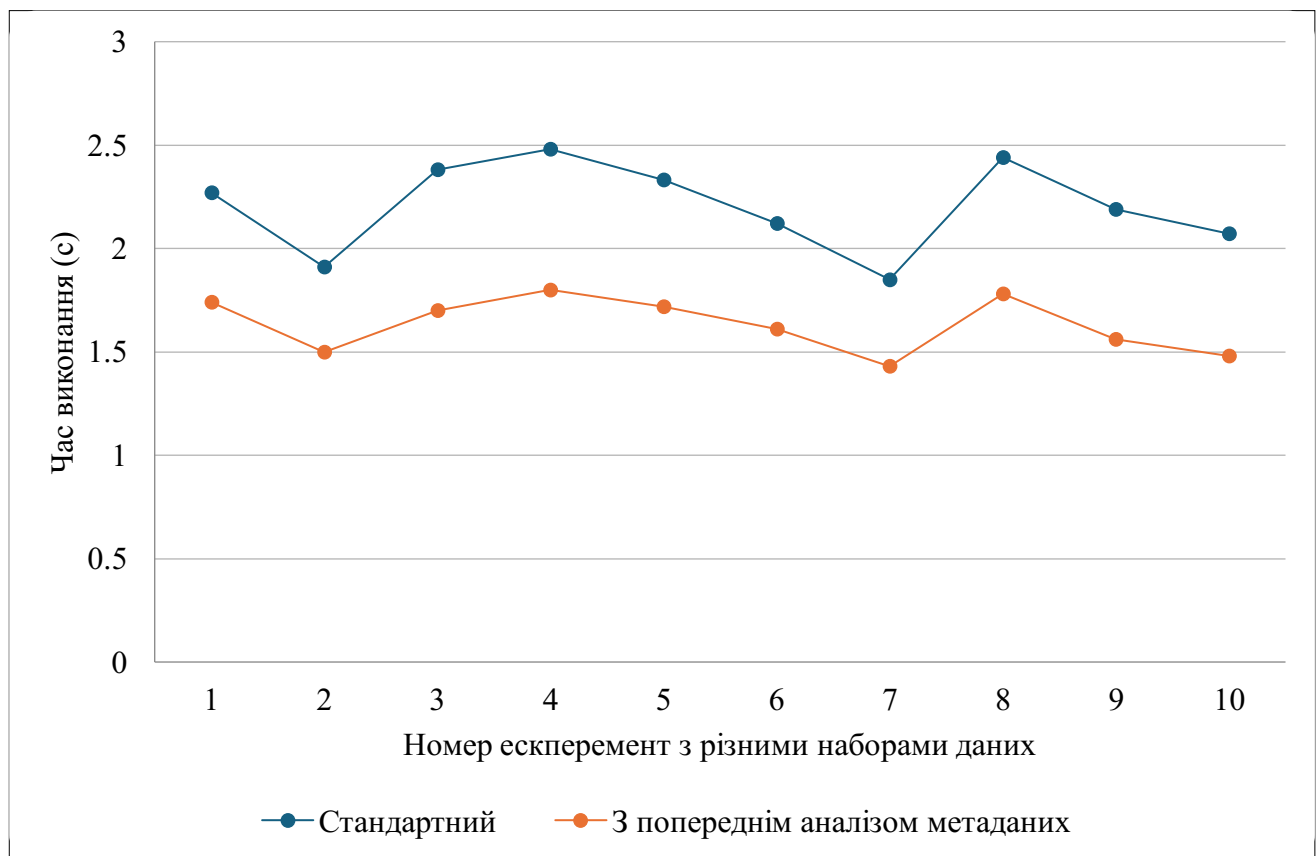


Рис 5.6 – Середній час виконання запитів для 5 вузлів

Після отримання результатів ми можемо обчислити середнє значення для кожного експерименту, що дозволить визначити загальний ефект від використання методу оптимізації. Для цього необхідно обчислити середні значення для кожного показника як з використанням оптимізації, так і без неї. Це дозволить отримати

базові показники, для визначення оптимізації. Розрахувати відсоток оптимізації за кожним показником, використовуючи формулу

$$k = \frac{T_{std} - T_{opt}}{T_{std}} \times 100\% \quad (4.1)$$

Відповідно для трьох вузлів $T_{std} = 1,896$ та $T_{opt} = 1,247$ ми отримуємо оптимізацію у 34.22%, та для п'яти вузлів $T_{std} = 2,204$ та $T_{opt} = 1,632$ ми отримуємо 25,95%.

Також було проведено експеримент з використанням оперативної пам'яті, для оцінки впливу методу оптимізації на ресурси системи. Основна мета цього експерименту полягала у визначенні, як змінюється обсяг споживання пам'яті при виконанні запитів із використанням методу мінімізації мережевого трафіку в порівнянні зі стандартним підходом.

На графіку (рис. 5.7) показано динаміку використання оперативної пам'яті для кожного з десяти повторень експерименту. Синя та зелена лінії відображають споживання пам'яті без застосування оптимізації, тоді як оранжева та блакитна лінії демонструють використання пам'яті з урахуванням методу оптимізації. Графік дає змогу візуально оцінити незначне збільшення обсягу пам'яті (близько 3-5%) при використанні оптимізованого підходу, що пов'язано з додатковими обчисленнями, необхідними для підтримки актуальності кешованих даних та обробки метаданих.

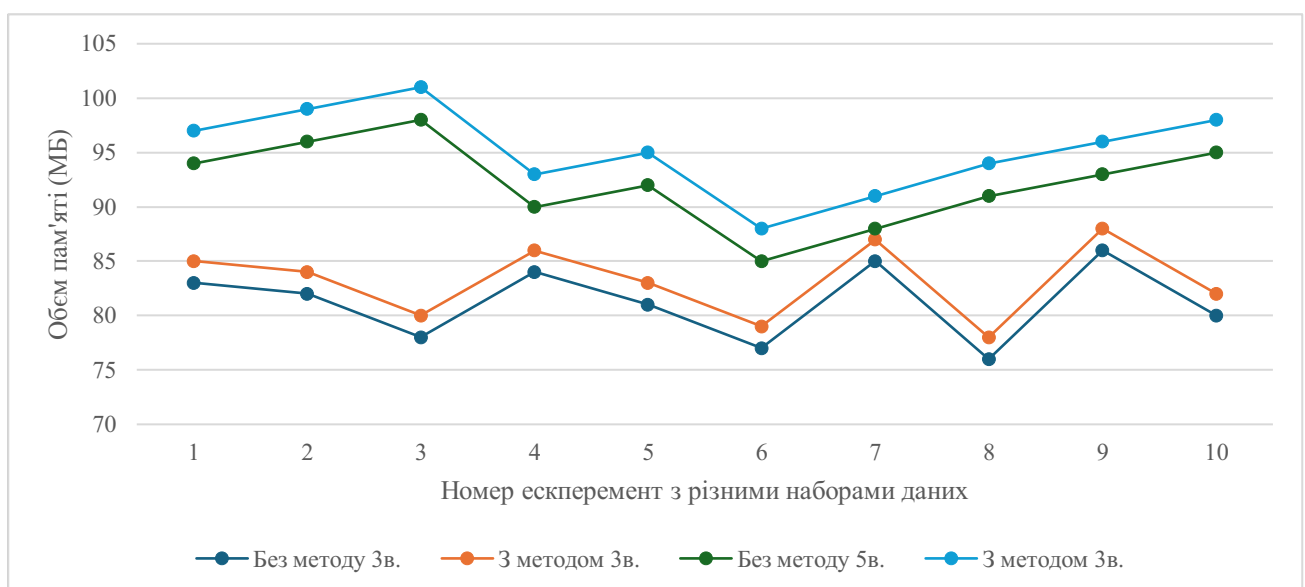


Рис 5.7 – Використання оперативної пам'яті при виконанні запитів

Як результат, можна зробити висновок, що запропонована оптимізація значно покращує час виконання запитів. Це означає, що система обробляє запити швидше, що особливо важливо для високонавантажених розподілених систем. Однак, одночасно з цим оптимізація призводить до невеликого, але помітного збільшення використання оперативної пам'яті, яке становить близько 3-5%. Це підвищення обумовлене додатковими обчисленнями та необхідністю зберігати метадані для підтримки актуального стану системи, що є важливим для забезпечення швидкої роботи та ефективного використання мережевого трафіку.

5.2.2 Модифікований метод узгодженості даних на основі методу Левенштейна

Було обрано різних наборів даних для експерименту з різними типами змін у рядках:

- Зміна частини рядка(заміна домашнього завдання) (рис. 5.8):

Було: *Виконати вправу №200,201,204;*

Стало: *Виконати вправу №200,201,204. Підготуватися до контрольної роботи;*

Найкращий варіант: $w_{ins}=0,5$, $w_{del}=2,5$, $w_{rep}=1,0$

- Зміна всього рядка(виправлення оцінки) :

Було: *10;*

Стало: *9;*

Найкращий варіант: $w_{ins}=1,5$, $w_{del}=1,0$, $w_{rep}=2,0$

- Зміна частини рядка (заміна прізвища):

Було: *Марія Ковальчук;*

Стало: *Марія Коваленко;*

Найкращий варіант: $w_{ins}=1,0$, $w_{del}=2$, $w_{rep}=1,5$

- Зміна частини рядка (подвійна заміна домашнього завдання):

Було: *Виконати вправу №200,201,204;*

Стало: *Виконати вправу №199,201,205;*

Найкращий варіант: $w_{ins}=1,0$, $w_{del}=2,0$, $w_{rep}=1,5$

```

Weights (Insertion: 1, Deletion: 2, Replacement: 1.5) - Distance: 37
Weights (Insertion: 0.5, Deletion: 2.5, Replacement: 1) - Distance: 18.5
Weights (Insertion: 1.5, Deletion: 1, Replacement: 2) - Distance: 55.5
Weights (Insertion: 1, Deletion: 3, Replacement: 1) - Distance: 37
Weights (Insertion: 0.8, Deletion: 2.2, Replacement: 1.1) - Distance: 29.6
Weights (Insertion: 1.2, Deletion: 2.5, Replacement: 0.9) - Distance: 44.4
Weights (Insertion: 1.1, Deletion: 1.8, Replacement: 1.4) - Distance: 40.7
Weights (Insertion: 1.3, Deletion: 2.3, Replacement: 1.2) - Distance: 48.1
Weights (Insertion: 0.9, Deletion: 1.5, Replacement: 1.7) - Distance: 33.3
Weights (Insertion: 1.4, Deletion: 2.1, Replacement: 1.6) - Distance: 51.8

Best Weights:
Insertion: 0.5, Deletion: 2.5, Replacement: 1
Minimum Distance: 18.5

```

Рис 5.8 – Реалізація визначення вагових коефіцієнтів

Для підтвердження ефективності запропонованого методу узгодженості даних на основі алгоритму Левенштейна з ваговими коефіцієнтами було проведено експеримент, для різних типів даних: з невеликим, середнім та великим розміром та відповідно з різною динамікою змін. Метою експерименту було дослідити, наскільки метод підходить для кожного типу текстових даних і як ефективно він оптимізує процес обробки запитів при різних розмірах текстів (рис 5.9).

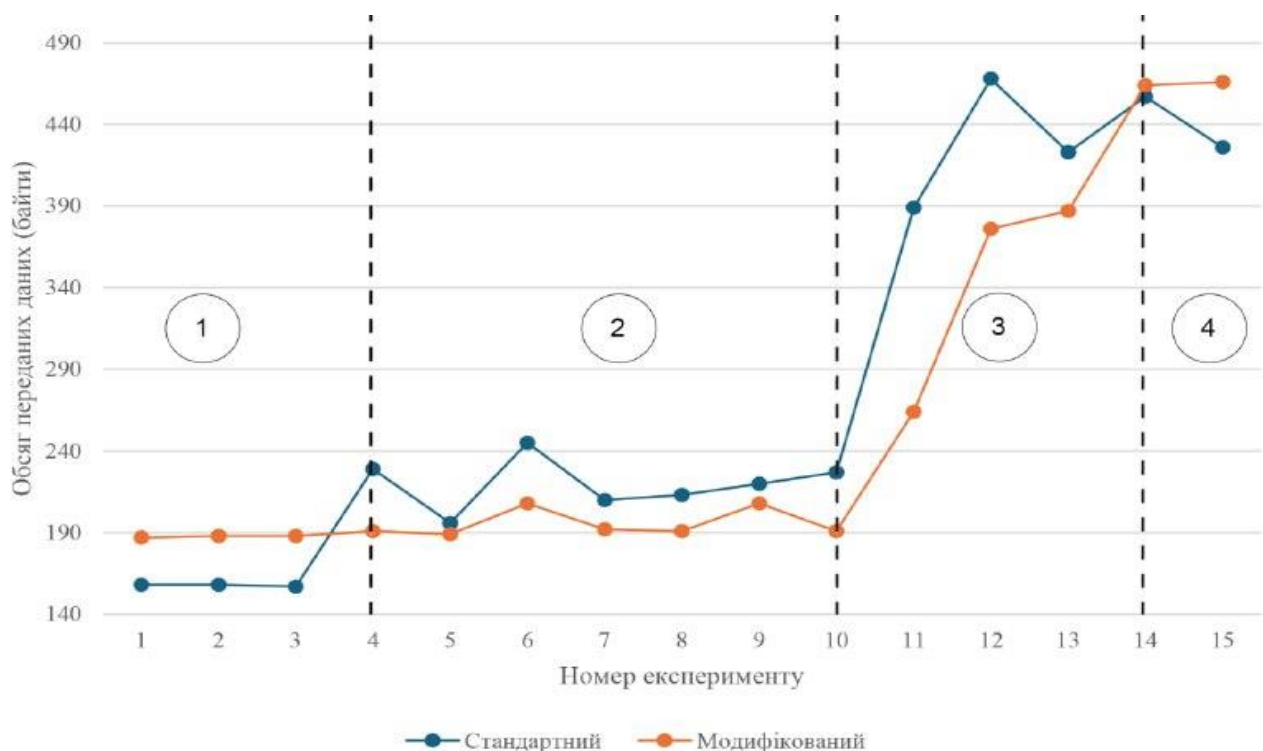


Рис 5.9 – Вартість передачі даних

З графіку видно, що у першій області де текстова зміна є невелика (наприклад значення оцінки) результати показують, що обсяг переданих даних як для стандартного методу, так і для модифікованого методу залишається

стабільним із незначними коливаннями. Важливо зазначити, що модифікований метод демонструє трохи нижчий обсяг переданих даних, ніж стандартний метод. Це свідчить про менш ефективне використання запропонованого методу, для невеликих текстових даних.

У другій області проводилося дослідження з невеликими реченнями та змінами у них. В результаті спостерігається значне зростання обсягу переданих даних у стандартному методі, тоді як модифікований метод демонструє більш стабільні результати із поступовим зростанням. Перевага модифікованого методу стає більш очевидною, оскільки він забезпечує менший обсяг переданої інформації навіть у випадку більшої кількості змін. Це свідчить про ефективність методу при невеликих змінах.

У третій області відбувалося дослідження з великими текстовими даними та незначних змінах у них, як результат обидва методи демонструють різке зростання обсягу переданих даних. Однак модифікований метод зберігає помітно нижчий рівень передачі, ніж стандартний. Це ще раз підтверджує перевагу модифікованого підходу при обробці значних обсягів змін у даних. Завдяки адаптивним коефіцієнтам алгоритму вдається оптимізувати передачу даних і знизити навантаження на мережу.

У фінальній четвертій області де використовувались великі текстові дані з великими змінами, модифікований метод продовжує демонструвати кращі результати, забезпечуючи менший обсяг переданих даних порівняно зі стандартним методом. Хоча різниця між методами стає менш значною, ефективність модифікованого підходу залишається вищою. Це підтверджує стабільність запропонованого методу в умовах великого навантаження та високої частоти змін.

Як результат середнє значення часу виконання для стандартного методу становить 278,40, тоді як для модифікованого методу — 259,33. Це свідчить про зменшення середнього часу виконання приблизно на 19,07 одиниць (або приблизно на 6,85%). Така оптимізація є помітною і вказує на покращення ефективності запитів при використанні модифікованого методу.

Згідно з отриманими результатами та зазначеними змінами, видно, що модифікований метод показує себе ефективним для роботи з середніми та великими текстовими даними, особливо в умовах, коли зміни у текстах є частковими, а не повними. Це видно з того, що різкі зміни в часі виконання у стандартному методі пов'язані з обробкою великих текстів, тоді як модифікований метод стабільніше виконує обробку, зменшуючи середній час виконання запитів навіть при великих обсягах даних.

Такий результат є особливо значущим для високо навантажених систем, де ефективність і стабільність часу виконання запитів є критично важливими. Модифікований метод, який демонструє зниження часу виконання при часткових змінах у великих текстах, може суттєво зменшити навантаження на систему. Він дозволяє обробляти тільки змінені частини даних, а не весь текст, що знижує обсяг оброблюваної інформації і, відповідно, час синхронізації.

Таким чином, модифікований метод можна рекомендувати для систем, які обробляють великі текстові блоки і часто стикаються з частковими змінами в них. У таких випадках метод не тільки зменшує середній час виконання запитів, але й забезпечує більш стабільну роботу системи при високому навантаженні. Це може бути корисно, наприклад, для систем управління контентом, де великі текстові блоки редагуються частково, або в базах даних із динамічними текстовими полями, де зміни відбуваються в окремих частинах тексту.

5.2.3 Ребалансування на основі генетичних алгоритмів елітарністю та адаптивним схрещенням

Для підтвердження ефективності запропонованого методу було проведено низку експериментальних досліджень, результати яких наведено у графічній формі (рис. 5.10). Початкові дані для експерименту були взяті з третього розділу, де детально описано характеристики вхідних даних і параметри системи, а також надано обґрунтування вибору обсягів, структур та форматів досліджуваних даних. Експерименти виконувалися в конфігурації з трьома вузлами розподіленої системи, що забезпечує реалістичні умови зберігання та обробки інформації, а також дозволяє виявити потенційні переваги та недоліки методу за умови множинної взаємодії між репліками.

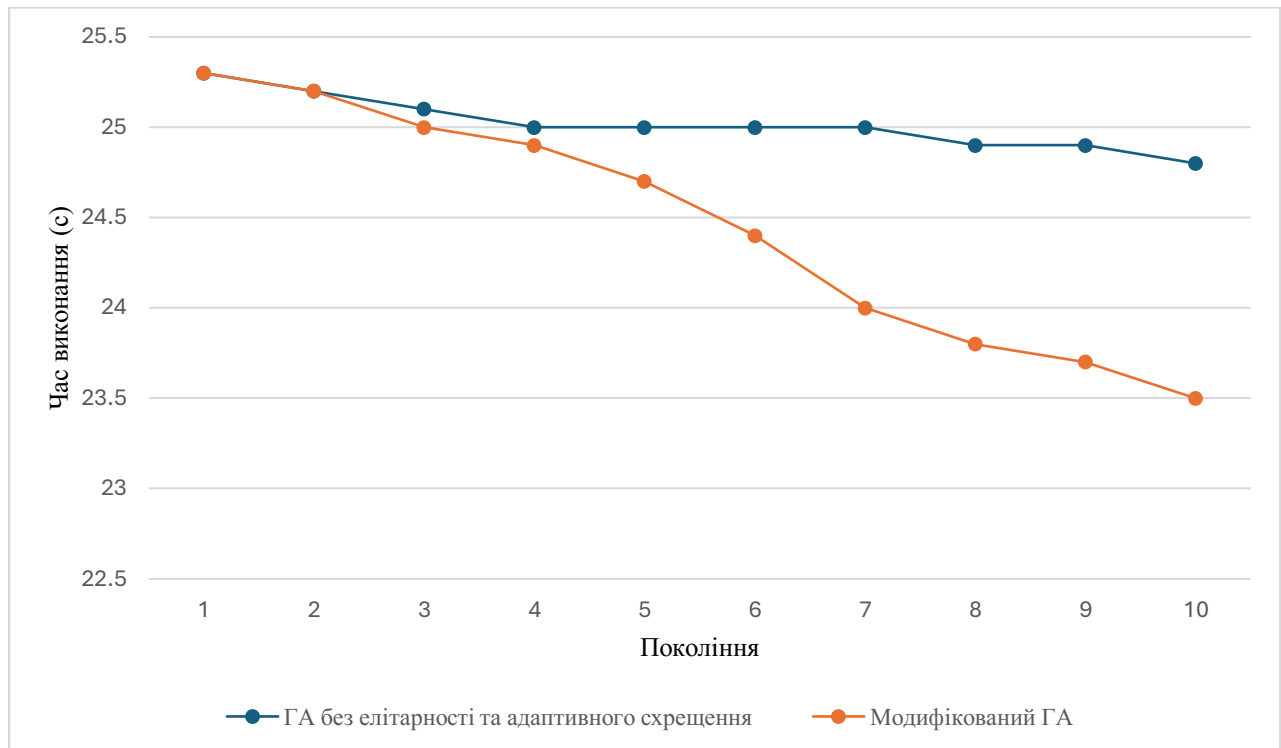


Рис 5.10 – Час виконання запитів по поколіннях для 3х вузлів

На графіку представлено порівняння найкращого часу виконання запитів по поколіннях для двох генетичних алгоритмів: без елітарності та адаптивного схрещення (синя лінія) та з ними (оранжева лінія).

З графіку можна зробити наступні висновки:

- Початкові покоління (1-3) обидва методи показують схожий час виконання запитів на початку, із незначною перевагою. Це може бути пов'язано з відсутністю значного впливу елітарності та адаптивності на ранніх етапах оптимізації.
- Середні покоління (4-7) ГА з елітарністю та адаптивним схрещенням починають демонструвати помітну перевагу в часі виконання. Його кривий зниження стає крутішим порівняно зі звичайним. Це вказує на те, що елітарність та адаптивне схрещення сприяють швидшій конвергенції до кращих рішень, дозволяючи зберігати найкращі індивіди та зменшувати кількість непотрібних обчислень.
- Кінцеві покоління (8-10) ГА з елітарністю та адаптивним схрещенням досягає стабільного поліпшення часу виконання, опускаючись до значення

близько 23,5 секунд. У той же час звичайний демонструє менш ефективне зменшення часу виконання, залишаючись на рівні приблизно 24,8 секунд. Цей результат показує, що ГА з елітарністю та адаптивним схрещенням є більш ефективним для задачі мінімізації часу виконання запитів у розподіленій системі.

Починаючи з третього покоління, модифікований ГА показує зниження часу виконання, опускаючись з 25,3 секунд до 23,5 секунд до десятого покоління. Це становить приблизно 7,1% зменшення часу виконання запитів у порівнянні зі стандартним ГА, який демонструє лише незначне зменшення, з 25,3 секунд до 24,8 секунд у десятому поколінні.

Крім того, для оцінки масштабованості та продуктивності методу було проведено додатковий експеримент із п'ятьма вузлами у розподіленій системі (рис. 5.11). Збільшення кількості вузлів дозволяє проаналізувати ефективність методу в умовах більшого навантаження на мережу та складнішого розподілу даних, що є важливим для підтвердження його стабільності та адаптивності у великих розподілених системах.

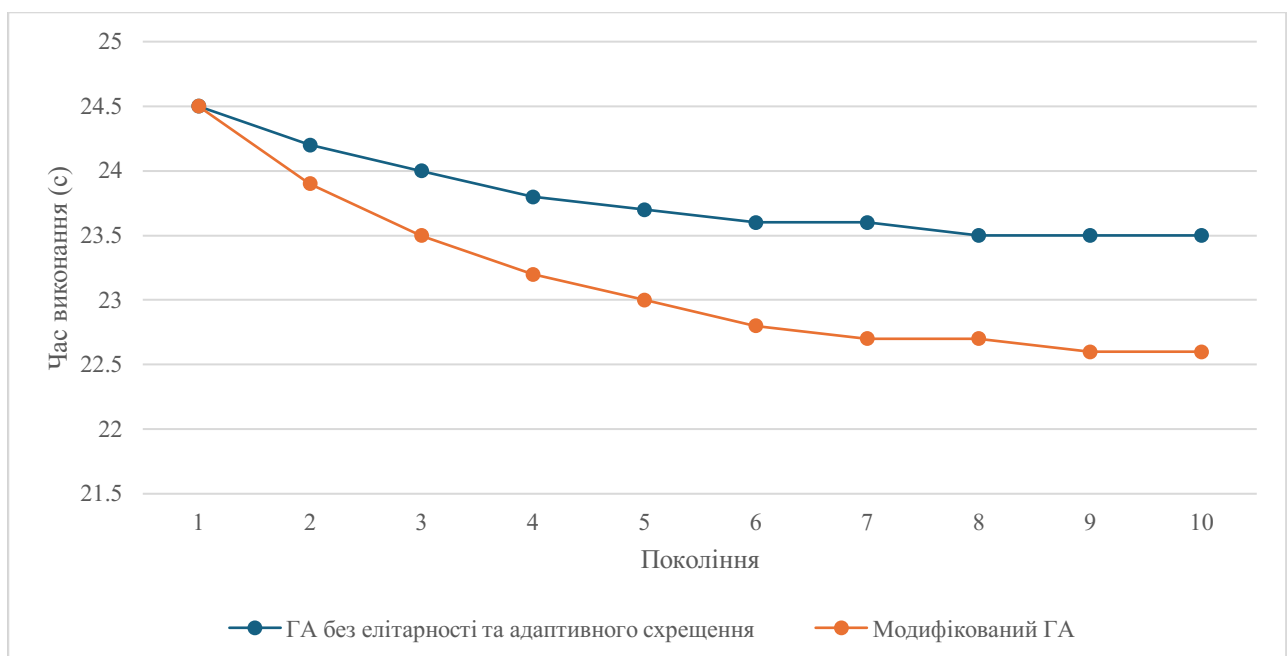


Рис 5.11 – Час виконання запитів по поколіннях для 5х вузлів

У випадку розподіленої системи з п'ятьма вузлами модифікований ГА з елітарністю та адаптивним схрещенням демонструє ще більш виражені переваги у порівнянні зі стандартним ГА. Збільшення кількості вузлів дозволяє методу

ефективніше використовувати адаптивні параметри для оптимізації розподілу даних, що сприяє значному зниженню часу виконання запитів. У порівнянні зі стандартним ГА, модифікований забезпечує швидшу конвергенцію до оптимального рішення навіть у більш складних умовах, підтверджуючи свою ефективність і стабільність у масштабованих розподілених системах.

Модифікований ГА також показує помітну перевагу над стандартним методом. У той час як стандартний поступово зменшує час виконання з 24,5 секунд до 23,5 секунд, модифікований досягає кращих результатів, знижуючи час виконання з 24,5 до 22,5 секунд у десятому поколінні. Це зменшення становить приблизно 8,2% у порівнянні зі стандартним ГА, що підтверджує ефективність модифікованого підходу в умовах більшої кількості вузлів.

Висновки до п'ятого розділу

Основною метою дослідження було практично перевірити запропоновані методи оптимізації виконання запитів у високонавантажених системах та порівняти їх з існуючими рішеннями. Для цього було створено наближене до реальних умов середовище—електронний онлайн-журнал для освітніх закладів, що дозволило провести дослідження та оцінити поведінку системи при впровадженні різних методів. Архітектура розподіленої системи електронного онлайн-журналу, побудована на чотирьох основних шарах, забезпечує ефективну та надійну роботу в умовах високих навантажень та розподіленого середовища. Використання сучасних технологій, таких як Raft Consensus Algorithm, Laravel, Vue.js та Docker, дозволило створити систему, яка відповідає вимогам сучасних освітніх закладів та забезпечує зручний доступ до інформації для всіх категорій користувачів.

У ході дослідження були розроблені та впроваджені такі методи:

- Метод мінімізації мережевого трафіку в алгоритмі консенсусу Raft. Шляхом попереднього обміну метаданими між вузлами та подальшої матеріалізації результатів вдалося зменшити обсяг мережевого трафіку та прискорити обробку запитів. Це забезпечило ефективніше використання мережевих ресурсів та підвищило швидкодію системи. Як результат було оптимізовано

час виконання запитів на 34,22% для системи що містить три вузли та 25,95%, для системи, що містить п'ять вузлів.

- Модифікований метод узгодженості даних на основі алгоритму Левенштейна з ваговими коефіцієнтами. Проведені дослідження підтвердили ефективність методу. Згідно з отриманими результатами, середнє значення часу виконання для стандартного методу становило 278,40 байт, тоді як для модифікованого методу—259,33 байт. Це свідчить про зменшення середнього часу виконання на 19,07 байт, або приблизно на 6,85%, що є суттєвим показником оптимізації при великій кількості запитів. Даний метод добре підходить для узгодженості великих текстових даних з невеликими змінами.
- Модифікований метод на основі генетичних алгоритмів з елітарністю та адаптивним схрещенням демонструє значні переваги над стандартним у контексті оптимізації часу. Використання цих методів дозволяє досягти швидшої конвергенції до оптимальних рішень, зменшити час виконання запитів та підвищити продуктивність розподілених систем. Для системи з трьома вузлами модифікований ГА забезпечив оптимізацію часу виконання запитів на 7,1%, знижуючи середній час з 25,3 до 23,5 секунд, тоді як для системи з п'ятьма вузлами оптимізація становила 8,2%, зменшуючи час з 24,5 до 22,5 секунд. Це особливо важливо для високонавантажених систем, де ефективність і стабільність роботи є критичними. Результати дослідження підтверджують доцільність впровадження модифікованого ГА для задач ребалансування даних у масштабованих розподілених середовищах.

ВИСНОВКИ

Дисертаційне дослідження було спрямоване на підвищення ефективності обробки запитів у високонавантажених розподілених базах даних, що є актуальною проблемою для сучасних інформаційних систем. Для досягнення означеної мети було сформульовано й розв'язано низку науково-прикладних завдань, спрямованих на виявлення та усунення недоліків у використанні мережевих ресурсів, узгодженні даних та ребалансуванні інформаційних фрагментів.

Виходячи з актуальності проблеми, було сформульовано такі основні завдання дослідження:

1. Проаналізувати існуючі методи та технології, спрямовані на підвищення продуктивності високонавантажених розподілених систем, виявити їх недоліки та визначити напрями вдосконалення.
2. Розробити метод зниження мережевого трафіку, який дозволить скорочуючи час відповіді тим самим оптимізувати використання мережевих ресурсів.
3. Удосконалити процес узгодження даних, запропонувавши підхід, що дозволить мінімізувати обсяг переданих змін при частих та невеликих оновленнях, особливо для текстових даних.
4. Розробити вдосконалений метод ребалансування, який дозволить динамічно оптимізувати розподіл фрагментів бази даних між вузлами, знижуючи середній час виконання запитів.

У процесі розв'язання вказаних завдань було отримано наступні наукові та практичні результати:

- Вперше розроблено новий метод мінімізації мережевого трафіку який базується на поєднанні принципів, притаманних як Raft, так і Leaderless Replication. Цей метод дозволяє зменшити обсяг передаваних даних між вузлами шляхом попереднього обміну метаданими та подальшої матеріалізації результатів. Експериментальні результати показали оптимізацію часу виконання запитів на 34,22% для системи з трьома вузлами та на 25,95% для системи з п'ятьма вузлами, що підтверджує ефективність методу в реальних умовах експлуатації.

- Удосконалено метод узгодженості даних на основі модифікованого алгоритму Левенштейна з ваговими коефіцієнтами. Запропонований підхід зменшує середній час виконання запитів на 6,85%. Це досягається за рахунок передачі лише необхідних дельт між версіями даних, що особливо ефективно при узгодженні великих текстових даних з незначними змінами. Метод показав високу ефективність у високо навантажених системах, де стабільність та швидкість узгодження даних є критично важливими.
- Удосконалено метод ребалансування даних на основі генетичних алгоритмів, додавши функції елітарності та адаптивного схрещення. Використання цього методу дозволило покращити розподіл даних між вузлами та знизити час виконання запитів. Для системи з трьома вузлами було досягнуто оптимізації часу виконання запитів на 7,1%, зменшуючи середній час з 25,3 до 23,5 секунд. Для системи з п'ятьма вузлами оптимізація становила 8,2%, знижуючи час з 24,5 до 22,5 секунд. Це підтверджує доцільність використання модифікованого генетичного алгоритму для задач ребалансування в масштабованих розподілених системах.

В результаті виконання поставлених завдань дало змогу виявити та усунути недоліки, пов'язані з надмірним мережевим трафіком, недостатньою точністю узгодження даних та оптимальним розподілом ресурсів у високонавантажених розподілених базах даних, а також розробити та експериментально підтвердити ефективність комплексних підходів, які покращують продуктивність систем, підвищують надійність їх роботи та забезпечують швидку реакцію на зміну умов експлуатації.

Отримані наукові результати мають потенціал подальшого розвитку та можуть бути адаптовані до інших типів високонавантажених систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Özsu, M. T., & Valduriez, P. (2020). Principles of Distributed Database Systems. Springer International Publishing. <https://doi.org/10.1007/978-3-030-26253-2>;
2. What is MongoDB? - MongoDB Manual. (n.d.). <https://docs.mongodb.com/manual/>;
3. RFC 2616: Hypertext Transfer Protocol. IETF Datatracker. <https://tools.ietf.org/html/rfc2616>;
4. Quinn, M. J. (2003). Parallel programming in C with MPI and OpenMP. <http://www.inf.puc-rio.br/~noemi/cd-06/cd3.pdf>;
5. Coulouris, G. F., & Dollimore, J. (1988). Distributed Systems: Concepts and design. <http://cdk5.net/errata/Errata.pdf>;
6. Белоус, Р., Крилов, Є., & Анікін, В. (2021). Методи оптимізації запитів розподілених БД. В Адаптивні системи автоматичного управління (Вип. 2, Issue 39, с. 3–11). Kyiv Polytechnic Institute. <https://doi.org/10.20535/1560-8956.39.2021.247364>;
7. Belous, R. V., & Krylov, Ye. V. (2024). Optimization of Raft usage in distributed systems with databases. В Scientific notes of Taurida National V.I. Vernadsky University. Series: Technical Sciences (Вип. 1, Issue 3, с. 37–41). Publishing House Helvetica (Publications). <https://doi.org/10.32782/2663-5941/2024.3.1/>;
8. Belous, R., & Krylov, Ye. (2024). Minimisation of network traffic in the Raft-like consensus algorithm. В Municipal economy of cities (Вип. 4, Issue 185, с. 2–6). O.M.Beketov National University of Urban Economy in Kharkiv. <https://doi.org/10.33042/2522-1809-2024-4-185-2-6>;
9. Mukhin, V., Zavgorodnii, V., Kornaga, Y., Zavgorodnya, A., Krylov, I., Rybalochka, A., Kornaga, V., & Belous, R. (2021). Devising a method to identify an incoming object based on the combination of unified information spaces. В Eastern-European Journal of Enterprise Technologies (Вип. 3, Issue 2 (111), с. 35–44). Private Company Technology Center. <https://doi.org/10.15587/1729-4061.2021.229568>;
10. Belous, R., & Krylov, I. (2023). Time optimization of process of data consistency in NOSQL. В Herald of Khmelnytskyi National University. Technical sciences (Вип. 321, Issue 3, с. 37–42). Khmelnytskyi National University. <https://doi.org/10.31891/2307-5732-2023-321-3-37-42> Mhawes A., and Hassan M.

Methods to Implement Homogeneous and Heterogeneous Distributed Database.

<https://www.noveltyjournals.com/upload/paper/paperpdf-1596030814.pdf>;

11. Okardi, B., & Asagba, O. P. (2021). Overview of distributed database system. *International Journal of Computer Techniques*, 8(1), 83–100. <http://www.ijctjournal.org/volume8/issue1/ijct-v8i1p8.pdf>;
12. Patidar, P. (2024, April 22). Understanding database partitioning in distributed systems: Rebalancing partitions. *Medium*. <https://medium.com/the-developers-diary/understanding-database-partitioning-in-distributed-systems-rebalancing-partitions-fa7fee542fd3>;
13. Chapter 6. Partitioning - Shichao's notes. <https://notes.shichao.io/dda/ch6/> ;
14. Database Replication. (2009). *B Encyclopedia of Database Systems* (c. 723–723). Springer US. https://doi.org/10.1007/978-0-387-39940-9_2426;
15. Types of database replication. GeeksforGeeks. <https://www.geeksforgeeks.org/types-of-database-replication-system-design> ;
16. Jogalekar, P., & Woodside, M. (2000). Evaluating the scalability of distributed systems. *B IEEE Transactions on Parallel and Distributed Systems* (Вип. 11, Issue 6, c. 589–603). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/71.862209>;
17. Burckhardt, S. (2015). Consistency in Distributed Systems. *B Lecture Notes in Computer Science* (c. 84–120). Springer International Publishing. https://doi.org/10.1007/978-3-319-28406-4_4;
18. Gomes, V. B. F., Kleppmann, M., Mulligan, D. P., & Beresford, A. R. (2017). Verifying strong eventual consistency in distributed systems. *B Proceedings of the ACM on Programming Languages* (Вип. 1, Issue OOPSLA, c. 1–28). Association for Computing Machinery (ACM). <https://doi.org/10.1145/3133933> ;
19. García-Pérez, Á., Gotsman, A., Meshman, Y., & Sergey, I. (2018). Paxos Consensus, Deconstructed and Abstracted (Extended Version) (Версія 1). arXiv. <https://doi.org/10.48550/ARXIV.1802.05969> ;
20. Ongaro, D., & Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)* (pp. 305-319);

21. Howard, H., & Mortier, R. (2020). Paxos vs Raft. B Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data (c. 1–9). EuroSys '20: Fifteenth EuroSys Conference 2020. ACM. <https://doi.org/10.1145/3380787.3393681> ;
22. Huang, D., Ma, X., & Zhang, S. (2020). Performance Analysis of the Raft Consensus Algorithm for Private Blockchains. B IEEE Transactions on Systems, Man, and Cybernetics: Systems (Вип. 50, Issue 1, c. 172–181). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/tsmc.2019.2895471>;
23. Batty, C. J. K. (1990). Tauberian theorems for the Laplace-Stieltjes transform. B Transactions of the American Mathematical Society (Вип. 322, Issue 2, c. 783–804). American Mathematical Society (AMS). <https://doi.org/10.1090/s0002-9947-1990-1013326-6>;
24. Zhou, L., Chen, Y., Li, T., & Yu, Y. (2012). The Semi-join Query Optimization In Distributed Database System. B Proceedings of 2012 National Conference on Information Technology and Computer Science. 2012 National Conference on Information Technology and Computer Science. Atlantis Press. <https://doi.org/10.2991/cits.2012.232> ;
25. Fan, Y., & Zhang, Y. (2024). Improved SDD-1 query optimization algorithm. B H. Wu & H. Li (Ред.), International Conference on Computer Graphics, Artificial Intelligence, and Data Processing (ICCAID 2023) (c. 28). International Conference on Computer Graphics, Artificial Intelligence, and Data Processing (ICCAID 2023). SPIE. <https://doi.org/10.1117/12.3026345>;
26. Luo, C., & Carey, M. J. (2022). DynaHash: Efficient Data Rebalancing in Apache AsterixDB. B 2022 IEEE 38th International Conference on Data Engineering (ICDE) (c. 485–497). 2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE. <https://doi.org/10.1109/icde53745.2022.00041>;
27. Krishnan, P., Lalitha, V., & Natarajan, L. (2020). Coded Data Rebalancing: Fundamental Limits and Constructions (Версія 4). arXiv. <https://doi.org/10.48550/ARXIV.2001.04939>;
28. Singh, A., Kahlon, K. S., & Virk, R. S. (2014). Nonreplicated Static Data Allocation in Distributed Databases Using Biogeography-Based Optimization. B Chinese

- Journal of Engineering (Вип. 2014, с. 1–9). Hindawi Limited.
<https://doi.org/10.1155/2014/785321>;
29. Aponso, G. C. a. L., Tennakon, T. M. T. I., Arampath, A. M. C. B., Kandeepan, S., & Amaratunga, H. P. K. K. S. (2017). Database optimization using genetic algorithms for distributed databases. *International Journal of Computer (IJC)*, 24(1), 23–27. <http://ijcjournal.org/index.php/InternationalJournalOfComputer/article/download/803/413>;
 30. Ahmad, I., Karlapalem, K., Kwok, Y.-K., & So, S.-K. (2002). B Distributed and Parallel Databases (Вип. 11, Issue 1, с. 5–32). Springer Science and Business Media LLC. <https://doi.org/10.1023/a:1013324605452>;
 31. Spears, W. M., & Anand, V. (1991). A study of crossover operators in genetic programming. B *Lecture Notes in Computer Science* (с. 409–418). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-54563-8_104;
 32. Poon, P. W., & Carter, J. N. (1995). Genetic algorithm crossover operators for ordering applications. *Computers & Operations Research*, 22(1), 135-147;
 33. Syswerda, G. (1989, June). Uniform crossover in genetic algorithms. In *ICGA* (Vol. 3, No. 2–9);
 34. Sevinc, E., & Cosar, A. (2010). An Evolutionary Genetic Algorithm for Optimization of Distributed Database Queries. B *The Computer Journal* (Вип. 54, Issue 5, с. 717–725). Oxford University Press (OUP). <https://doi.org/10.1093/comjnl/bxp130>;
 35. Cedeño, W., & Vemuri, V. R. (1997). Database design with genetic algorithms. In *Evolutionary algorithms in engineering applications* (pp. 189-206). Berlin, Heidelberg: Springer Berlin Heidelberg;
 36. Palmes, P. P., Hayasaka, T., & Usui, S. (2005). Mutation-Based Genetic Neural Network. B *IEEE Transactions on Neural Networks* (Вип. 16, Issue 3, с. 587–600). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/tnn.2005.844858>;
 37. Maaranen, H., Miettinen, K., & Penttinen, A. (2007). On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, 37, 405-436;

38. Bottaci, L. (2001, May). A genetic algorithm fitness function for mutation testing. In *Proceedings of the SEMINALL-workshop at the 23rd international conference on software engineering, Toronto, Canada*;
39. Lambora, A., Gupta, K., & Chopra, K. (2019). Genetic Algorithm- A Literature Review. B 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon) (с. 380–384). 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). IEEE. <https://doi.org/10.1109/comitcon.2019.8862255>;
40. Lin, C.-H. (2013). A rough penalty genetic algorithm for constrained optimization. B Information Sciences (Вип. 241, с. 119–137). Elsevier BV. <https://doi.org/10.1016/j.ins.2013.04.001>
41. Reis, D., Piedade, B., Correia, F. F., Dias, J. P., & Aguiar, A. (2022). Developing Docker and Docker-Compose Specifications: A Developers' Survey. B IEEE Access (Вип. 10, с. 2318–2329). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/access.2021.3137671>;

ДОДАТОК А. АКТ ВПРОВАДЖЕННЯ В НАВЧАЛЬНИЙ ПРОЦЕС

«ЗАТВЕРДЖУЮ»

Декан факультету інформатики
та обчислювальної техніки

КПІ ім. Ігоря Сікорського

Ярослав КОРНАГА

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 Белоуса Романа Володимировича «Методи і програмні засоби підвищення ефективності виконання запитів у високонавантажених 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 С.В., вчений секретар кафедри, к.ф.-м.н., доц. Гавриленко О.В. розглянувши матеріали навчально-методичного забезпечення курсів: «Розроблення веб-застосувань» який підготовлений та викладається асистентом Белоусом Р. В., встановили, що в навчальному процесі на лабораторних заняттях, а також в процесі підготовки бакалаврських атестаційних робіт використовуються такі наукові результати, отримані дисертантом особисто:

- метод мінімізації обсягу мережевого трафіку у розподілених базах даних із використанням алгоритму консенсусу Raft шляхом попереднього обміну метаданими між вузлами та подальшої матеріалізації результатів на стороні вузла-лідера;
- модифікований метод узгодженості даних на основі Левенштейна з ваговими коефіцієнтами для часткових змін, що дозволяє скоротити обсяг переданих даних при синхронізації вузлів розподілених систем;
- технологія динамічного ребалансування даних у розподілених базах даних із застосуванням генетичних алгоритмів з елітарністю та адаптивним схрещенням для оптимального розподілу фрагментів бази даних.

Впровадження результатів дисертаційної роботи Белоус Р.В. 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.

Олександр РОЛІК

Доцент кафедри
інформаційних систем та технологій,
к.т.н., доц.

Євген КРИЛОВ

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

Олена ГАВРИЛЕНКО

ДОДАТОК Б. ФАЙЛ DOCKER-COMPOSE.YAML ДЛЯ РОЗГОРТАННЯ

docker-compose.yaml.

services:

backend-db:

image: mysql:8.0.36

container_name: backend-db

environment:

- "MYSQL_DATABASE=\${DB_DATABASE}"
- "MYSQL_USER=\${DB_USERNAME}"
- "MYSQL_PASSWORD=\${DB_PASSWORD}"
- "MYSQL_ROOT_PASSWORD=\${DB_PASSWORD}"

ports:

- "\${DOCKER_DB_PORT}:\${DB_PORT}"

volumes:

- db_data:/var/lib/mysql

backend-php:

build:

context: .

dockerfile: ./docker/php/Dockerfile

args:

- APP_ENV=\${APP_ENV:-local}
- BACK_ARTIFACTS_DIR=/tmp/composer/vendor

container_name: backend-php

working_dir: /app

command: /bin/sh -c "/app/ini-local.sh"

volumes:

- ./app:cached

ports:

- "9000:9000"

depends_on:

- backend-db

- backend-redis

web:

image: nginx:1.25.4-alpine3.18

container_name: backend-web

working_dir: /app

volumes:

- ./public:/app/public/

- ./docker/nginx/nginx_local.conf:/etc/nginx/conf.d/default.conf

ports:

- "\${DOCKER_WEB_PORT}:80"

depends_on:

- backend-php

backend-redis:

image: redis:latest

container_name: backend-redis

volumes:

- redis_data:/data:cached

ports:

- "\${DOCKER_REDIS_PORT}:\${REDIS_PORT}"

volumes:

db_data: {}

redis_data: {}

ms_data: {}