МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

Кваліфікаційна наукова
праця на правах рукопису

ПАТЛАНЬ ЄВГЕН ЮРІЙОВИЧ

УДК 004.65:004.75:004.85(043.5)

ДИСЕРТАЦІЯ

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ДИНАМІЧНОЇ ІНТЕГРАЦІЇ РОЗПОДІЛЕНИХ БАЗ ДАНИХ ІЗ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ

122 – Комп’ютерні науки

12 – Інформаційні технології
галузь знань

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів, текстів інших авторів супроводжується посиланнями на відповідне джерело

_____ Патлань Євген Юрійович

Науковий керівник Білоус Ірина Володимирівна
кандидат технічних наук, доцент

Чернігів – 2026

АНОТАЦІЯ

Патлань Є.Ю. Інформаційна технологія динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 – «Комп'ютерні науки» (12 – «Інформаційні технології»). – Національний університет «Чернігівська Політехніка», МОН України, Чернігів, 2026.

У роботі вирішено актуальне наукове завдання з розробки інформаційної технології динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю на основі технології PostgreSQL Foreign Data Wrappers та інтелектуальних AI-агентів.

Об'єктом дослідження є процеси динамічної інтеграції та управління продуктивністю у розподілених системах баз даних із багатоваріантною персистентністю, що функціонують в умовах змінних навантажень та структурного дрейфу даних.

Предметом дослідження є математичні моделі, методи та програмні засоби управління компромісами продуктивності та узгодженості у розподілених системах баз даних із багатоваріантною персистентністю на основі механізму PostgreSQL Foreign Data Wrappers.

Метою дисертаційного дослідження є підвищення ефективності динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю. Завдання дослідження полягають у побудові мультиплікативної моделі деградації продуктивності, що інтегрує п'ять теоретичних парадоксів розподілених баз даних із багатоваріантною персистентністю; удосконаленні методу адаптивного управління багатоваріантною персистентністю на основі EML-алгоритмів та еволюції схем даних; розвитку методу автономного масштабування на основі інтелектуальних агентів управління продуктивністю.

В основу методології дослідження покладено математичне моделювання, теорію масового обслуговування, теорію ймовірностей та математичну статистику, нечітку логіку, глибоке навчання з підкріпленням (Deep Q-Network), адаптивне потокове машинне навчання (Evolving Machine Learning), рекурентні нейронні мережі (LSTM) та класичні статистичні моделі (ARIMA). Для емпіричної валідації використано метрики R^2, RMSE, MAPE та 5-fold cross-validation.

У вступі обґрунтовано актуальність теми дослідження, сформульовані мета, задачі та методи дослідження, наведено зв'язок дослідження з науковими програмами, описано наукову новизну і практичне значення результатів дисертаційної роботи.

У першому розділі проведено системний аналіз існуючих підходів до моделювання продуктивності розподілених баз даних. Досліджено еволюцію парадигм зберігання даних від монолітних СКБД до polyglot persistence архітектури. Проаналізовано технологію PostgreSQL Foreign Data Wrappers як інструмент федеративної інтеграції та ідентифіковано п'ять теоретичних парадоксів продуктивності: CAP-обмеження, деградація від узгодженості, хеш-колізії (Birthday Paradox), накладні витрати NULL-логіки та нелінійна деградація масштабування (USL). Виконано порівняльний аналіз існуючих моделей продуктивності та показано, що жодна з них не інтегрує всі п'ять компонентів деградації.

У другому розділі розроблено мультиплікативну математичну модель продуктивності виду $P_{total} = P_0 \cdot exp(-D_{total})$, де D_{total} є зваженою сумою п'яти компонентів деградації. Для кожного компонента отримано аналітичні вирази на основі CAP-теореми, теорії масового обслуговування, Birthday Paradox та Universal Scalability Law. Калібрування коефіцієнтів виконано методами Bottom-Up аналітичної оцінки та Top-Down глобальної оптимізації. Модель валідовано на 60 мільярдах запитів з точністю $R^2 = 0.895$, MAPE = 12.45%.

У третьому розділі розроблено модель і метод адаптивного управління багатоваріантною персистентністю на основі EML-алгоритмів із використанням

кортежів комірок (cell tuples), embedding-векторів (Bi-LSTM, BERT) та потокової архітектури Kafka+CapyMOA. Модель забезпечує автоматичну адаптацію до структурного дрейфу даних у 15–20 разів швидше порівняно з традиційними ETL-системами при F1-score зіставлення атрибутів 0.87.

У четвертому розділі було проведено експериментальну валідацію моделі деградації продуктивності та методу автономного масштабування на основі інтелектуальних DQN-агентів: DQN Scaling Agent з 15-вимірним вектором стану та 5 дискретними діями для автономного масштабування, та Schema Evolution Agent з трьох канальним детектуванням змін для адаптації схем. Агент масштабування інтегровано з п’ятьма стратегіями управління парадоксами, що дозволяє підвищити продуктивність системи з 50.3% до 93.6% під час пікових навантажень. Наведено результати експериментальної валідації, що підтверджують ефективність запропонованих моделей та методів.

У п’ятому розділі реалізовано програмну систему DIS (Dynamic Integration System) на реальних кластерах баз даних із багатоваріантною персистентністю (PostgreSQL Patroni, Percona XtraDB Cluster, MongoDB Replica Set) у контейнерній інфраструктурі з 13 Docker-контейнерів.

Основні результати дослідження та наукова новизна роботи полягають у розробці інформаційної технології динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю, що включає: мультиплікативну модель деградації продуктивності, яка інтегрує п'ять парадоксів розподілених систем із багатоваріантною персистентністю у єдину калібровану формалізацію; удосконалений метод адаптивного управління багатоваріантною персистентністю на основі EML-алгоритмів із автоматичною адаптацією до структурного дрейфу даних; метод автономного масштабування на основі DQN-агента, інтегрованого зі стратегіями управління парадоксами продуктивності.

Практичне значення отриманих результатів полягає в тому, що вони у сукупності утворюють інформаційну технологію динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю та автоматичним управлінням продуктивністю. Запропоновано практичні рекомендації щодо

оптимального рівня узгодженості ($c \in [0.3, 0.5]$) та меж ефективного масштабування ($n \leq 50$ вузлів).

Результати дисертаційного дослідження впроваджені:

– в навчальному процесі Національного університету «Чернігівська політехніка» при викладанні дисциплін «Засоби інтеграції розподілених систем», «Бази даних», «Адміністрування баз даних» та «Інтелектуальний аналіз даних», – в процесі навчання бакалаврів спеціальності F2 – «Інженерія програмного забезпечення» на кафедрі інформаційних технологій та програмної інженерії (довідка про впровадження №202/08-2180 від 20.10.2026, Додаток Б1).

– в науково-практичній діяльності Національного університету «Чернігівська політехніка» при виконанні та реалізації НДР №0124U003853 «Методи та моделі діджиталізації процесів розробки інформаційних систем для розвитку Smart City технологій» та НДР №0126U001541 «Розробка інтелектуальної моніторингової геоінформаційної платформи для територіальних громад постраждалих від збройного конфлікту та техногенних катастроф», що виконується за фінансуванням Міністерства освіти та науки України (довідка про впровадження №202/07-2176 від 19.10.2026, Додаток Б2).

– в роботі Товариства з обмеженою відповідальністю „УАПРОМ” (акт про впровадження №186 від 19.03.2026, Додаток Б3).

– в роботі LIMITED LIABILITY COMPANY „ADVERTISING AGENCY SATU.KZ“ (акт про впровадження №32 від 19.03.2026, Додаток Б4).

Ключові слова: розподілені бази даних, багатоваріантна персистентність, Foreign Data Wrappers, деградація продуктивності, CAP-теорема, глибоке навчання з підкріпленням, Evolving Machine Learning, автономне масштабування.

ABSTRACT

Patlan Ye. Yu. Information technology for dynamic integration of distributed databases with polyglot persistence. – Qualifying scientific work on the rights of a manuscript.

Thesis for the Doctor of Philosophy degree in Specialty 122 "Computer Science" (Field of Knowledge 12 "Information Technologies"). – Chernihiv Polytechnic National University, Ministry of Education and Science of Ukraine, Chernihiv, 2026.

The thesis solves a relevant scientific problem of developing an information technology for dynamic integration of distributed databases with polyglot persistence based on PostgreSQL Foreign Data Wrappers technology and intelligent AI agents.

The object of the research is the processes of dynamic integration and performance management in distributed database systems with polyglot persistence operating under variable workloads and structural data drift.

The subject of the research is mathematical models, methods, and software tools for managing performance-consistency trade-offs in distributed database systems with polyglot persistence based on the PostgreSQL Foreign Data Wrappers mechanism.

The purpose of the research is to improve the efficiency of dynamic integration of distributed databases with polyglot persistence. The research objectives are to construct a multiplicative performance degradation model integrating five theoretical paradoxes of distributed database systems with polyglot persistence; to improve the method of adaptive polyglot persistence management based on EML algorithms and schema evolution; and to develop the method of autonomous scaling based on intelligent performance management agents.

The research methodology is based on mathematical modeling, queuing theory, probability theory and mathematical statistics, fuzzy logic, deep reinforcement learning (Deep Q-Network), adaptive streaming machine learning (Evolving Machine Learning), recurrent neural networks (LSTM), and classical statistical models (ARIMA). Empirical validation employs the R^2, RMSE, and MAPE metrics and 5-fold cross-validation.

The introduction substantiates the relevance of the research topic, formulates the purpose, objectives, and methods of the research, shows the connection of the research with scientific programs, and describes the scientific novelty and practical significance of the results.

The first chapter provides a systematic analysis of existing approaches to performance modeling of distributed databases. The evolution of data storage paradigms from monolithic DBMS to polyglot persistence architectures is examined. PostgreSQL Foreign Data Wrappers technology is analyzed as a tool for federated integration, and five theoretical performance paradoxes are identified: CAP limitations, consistency degradation, hash collisions (Birthday Paradox), NULL logic overhead, and nonlinear scaling degradation (USL). A comparative analysis of existing performance models shows that none of them integrates all five degradation components.

The second chapter develops a multiplicative mathematical performance model of the form $P_total = P_0 \cdot exp(-D_total)$, where D_total is a weighted sum of five degradation components. Analytical expressions for each component are derived from the CAP theorem, queuing theory, the Birthday Paradox, and the Universal Scalability Law. The coefficients are calibrated using Bottom-Up analytical estimation and Top-Down global optimization. The model is validated on 60 billion queries with $R^2 = 0.895$ and MAPE = 12.45%.

The third chapter develops a model and method of adaptive polyglot persistence management based on EML algorithms using cell tuples, embedding vectors (Bi-LSTM, BERT), and the Kafka+CapyMOA streaming architecture. The model adapts automatically to structural data drift 15–20 times faster than traditional ETL systems, with an attribute matching F1-score of 0.87.

The fourth chapter presents the experimental validation of the performance degradation model and of the autonomous scaling method based on intelligent DQN agents: a DQN Scaling Agent with a 15-dimensional state vector and 5 discrete actions for autonomous scaling, and a Schema Evolution Agent with three-channel change detection for schema adaptation. The scaling agent is integrated with five paradox

management strategies, which raises system performance from 50.3% to 93.6% during peak workloads. The experimental results confirm the effectiveness of the proposed models and methods.

The fifth chapter implements the DIS (Dynamic Integration System) software system on real database clusters with polyglot persistence (PostgreSQL Patroni, Percona XtraDB Cluster, MongoDB Replica Set) in a containerized infrastructure of 13 Docker containers.

The main results and scientific novelty of the work lie in the development of an information technology for dynamic integration of distributed databases with polyglot persistence, which includes: a multiplicative performance degradation model that integrates five paradoxes of distributed systems with polyglot persistence into a single calibrated formalization; an improved method of adaptive polyglot persistence management based on EML algorithms with automatic adaptation to structural data drift; and a method of autonomous scaling based on a DQN agent integrated with performance paradox management strategies.

The practical significance of the results is that together they form an information technology for dynamic integration of distributed databases with polyglot persistence and automatic performance management. Practical recommendations are given for the optimal consistency level ($c \in [0.3, 0.5]$) and the limits of effective scaling ($n \leq 50$ nodes).

The results of the dissertation research have been implemented:

– in the educational process of Chernihiv Polytechnic National University, in teaching the courses "Distributed Systems Integration Tools", "Databases", "Database Administration" and "Data Mining" to bachelor's students of Specialty F2 "Software Engineering" at the Department of Information Technologies and Software Engineering (implementation certificate No. 202/08-2180 of 20.10.2026, Appendix B1);

– in the research activities of Chernihiv Polytechnic National University, in carrying out research project No. 0124U003853 "Methods and Models for Digitalizing Information System Development Processes for Smart City Technologies" and

research project No. 0126U001541 "Development of an Intelligent Monitoring Geoinformation Platform for Territorial Communities Affected by Armed Conflict and Man-Made Disasters", funded by the Ministry of Education and Science of Ukraine (implementation certificate No. 202/07-2176 of 19.10.2026, Appendix B2);

– in the operations of UAPROM LLC (implementation act No. 186 of 19.03.2026, Appendix B3);

– in the operations of ADVERTISING AGENCY SATU.KZ LLP (implementation act No. 32 of 19.03.2026, Appendix B4).

Keywords: distributed databases, polyglot persistence, Foreign Data Wrappers, performance degradation, CAP theorem, deep reinforcement learning, Evolving Machine Learning, autonomous scaling.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Patlan Y., Bilous I., Trunova O., Skiter I., Voitsekhovska M. Mathematical Modeling of Performance-Consistency Trade-offs and Theoretical Paradoxes in Distributed Database Integration Using PostgreSQL FDW // Lecture Notes in Computer Science. Springer, 2026. 21 p.

2. Патлань Є.Ю., Білоус І.В. Метод автоматизованої міграції даних між варіантами сховищ у системах із багатоваріантною персистентністю // Інформаційні технології та суспільство. 2025. Вип. 4(19). С. 129–139. DOI: 10.32689/maup.it.2025.4.21

3. Патлань Є.Ю. Модель багатоваріантної персистентності у середовищі динамічно інтегрованих розподілених баз даних // Наука і Техніка Сьогодні. 2025. №8(49). С. 1623–1633. DOI: 10.52058/2786-6025-2025-8(49)-1623-1633

4. Патлань Є.Ю., Білоус І.В. Сценарії використання багатоваріантної персистентності для підтримки fault-tolerance у розподілених БД // Таврійський науковий вісник. Серія: Технічні науки. 2025. № 4, ч. 1. С. 227–234. DOI: 10.32782/tnv-tech.2025.4.1.24

5. Patlan Y., Bilous I. Agent-based approach to dynamic schema synchronization in distributed databases // Transactions of Kremenchuk Mykhailo Ostrohradskyi National University. 2026. Vol. 1, No. 1(156). P. 131–144. DOI: 10.32782/1995-0519.2026.1.1.17

Наукові праці, які засвідчують апробацію матеріалів дисертації

1. Патлань Є. Ю. Comparative Analysis of Distributed Database Clusters. Юність науки – 2024 : зб. тез доп. XIV Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 24–26 квіт. 2024 р.). Чернігів : НУ «Чернігівська політехніка», 2024.

2. Патлань Є. Ю. Distributed Database Dilemmas: Paradoxes That Shape Modern Information Systems. Юність науки – 2025 : зб. тез доп. XV Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 23–25 квіт. 2025 р.). Чернігів : НУ «Чернігівська політехніка», 2025.

3. Патлань Є. Ю. PostgreSQL Job Queues versus Valkey Streams: a Short Empirical Report. Юність науки – 2026 : зб. тез доп. XVI Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 22–23 квіт. 2026 р.). Чернігів : НУ «Чернігівська політехніка», 2026.

4. Patlan Y., Bilous I. Estimating performance, consistency, and scalability in dynamic database integration and clustering through modeling. Математичне та імітаційне моделювання систем. МОДС 2024 : тези доп. XIX Міжнар. наук.-практ. конф. (м. Чернігів, 2024 р.). Чернігів : НУ «Чернігівська політехніка», 2024. С. 107–109.

5. Patlan Ye. Yu. Paradoxes of distributed database systems: identification and classification. Innovation and development in world science : Proceedings of the 11th International Scientific and Practical Conference (Zurich, August 17–19, 2026). Zurich : MDPC Publishing, 2026. P. 21–26. ISBN 978-3-954753-21-5.

6. Patlan Ye. Integration mechanisms for distributed databases with polyglot persistence: a comparative classification. Science, technology and global challenges : Proceedings of the 13th International Scientific and Practical Conference (Tokyo, August 20–22, 2026). Tokyo : CPN Publishing Group, 2026. P. 35–40. ISBN 978-4-9783419-7-6.

7. Patlan Ye. Divergent NULL semantics in federated queries over heterogeneous data stores. European science and innovation congress : Proceedings of the 10th International Scientific and Practical Conference (Barcelona, August 24–26, 2026). Barcelona : Barca Academy Publishing, 2026. P. 34–37. ISBN 978-84-15927-36-5.

8. Patlan Ye. Paradox as a category of scientific inquiry in database theory. Modern science: challenges and prospects : Proceedings of the 1st International Scientific and Practical Conference (London, August 28–30, 2026). London : Cognum Publishing House, 2026. P. 64–68. ISBN 978-92-9472-198-3.

9. Patlan Ye. The mean is not the system: heavy-tailed latency in the design of federated databases. Global perspectives in scientific research : Proceedings of the 1st International Scientific and Practical Conference (Boston, September 4–6, 2026). Boston : BoScience Publisher, 2026. P. 47–50. ISBN 978-1-73981-120-4.

10. Patlan Ye. Hidden coupling in relational database systems: local optimization as a source of global degradation. Contemporary trends in modern science : Proceedings of the 1st International Scientific and Practical Conference (Munich, September 7–9, 2026). Munich : MDPC Publishing, 2026. P. 50–53. ISBN 978-3-954753-07-9.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 15
ВСТУП 18
РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМ УПРАВЛІННЯ ТА ЗАБЕЗПЕЧЕННЯ БАГАТОВАРІАНТНОЇ ПЕРСИСТЕНТНОСТІ У РОЗПОДІЛЕНИХ БАЗАХ ДАНИХ 30
1.1 Аналіз сучасних розподілених баз даних та їх архітектури 30
1.2 Аналіз теоретичних обмежень розподілених систем: п’ять парадоксів продуктивності 37
1.3. Аналіз існуючих підходів до моделювання продуктивності розподілених БД 51
1.4 Аналіз існуючих підходів до адаптивного управління розподіленими базами даних 53
1.5. Концептуальне моделювання розподілених баз даних із багатоваріантною персистентністю 57
1.6. Архітектурний огляд системи динамічної інтеграції 58
1.7. Постановка наукової проблеми та задач дослідження 59
Висновки до розділу 1 62
РОЗДІЛ 2 УНІФІКОВАНА МУЛЬТИПЛІКАТИВНА МОДЕЛЬ ПРОДУКТИВНОСТІ РОЗПОДІЛЕНИХ БАЗ ДАНИХ ІЗ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ 64
2.1. Формалізація стану розподіленої системи баз даних 64
2.2. Мультиплікативна модель продуктивності 69
2.3. Діагностична декомпозиція коефіцієнтів 85
2.5. Фінальна калібрована модель 95
Висновки до розділу 2 104
РОЗДІЛ 3 МЕТОД ТА МОДЕЛЬ АДАПТИВНОГО УПРАВЛІННЯ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ НА ОСНОВІ АДАПТИВНИХ EML-АЛГОРИТМІВ ТА ЕВОЛЮЦІЇ СХЕМ ДАНИХ 106
3.1. Формалізація багатоваріантних представлень даних 106
3.2. Метод адаптивного управління потокової системи з EML 116
3.3. Застосування Embedding-векторів та adversarial training для адаптації моделей персистентності 127
3.4. Реалізація потокового пайплайну 136
3.5. Порівняння з існуючими підходами 142
Висновки до розділу 3 145
РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНА ВАЛІДАЦІЯ МОДЕЛІ ДЕГРАДАЦІЇ ПРОДУКТИВНОСТІ ТА МЕТОДУ АВТОНОМНОГО МАСШТАБУВАННЯ НА ОСНОВІ DQN-АГЕНТА В РОЗПОДІЛЕНИХ СИСТЕМАХ БАЗ ДАНИХ ІЗ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ 148
4.1 Експериментальне середовище 148
4.2. Методологія валідації 155
4.3. Результати валідації моделі на великомасштабному стенді 157
4.4. MVP-валідація та метод автономного масштабування на основі DQN-агента 167
4.5. Продуктивність механізму Foreign Data Wrappers 182

Висновки до розділу 4 188
РОЗДІЛ 5 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДИНАМІЧНОЇ ІНТЕГРАЦІЇ РОЗПОДІЛЕНИХ БАЗ ДАНИХ ІЗ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ 191
5.1. Архітектура програмної системи DIS 191
5.2. Технологічний стек та розгортання 201
5.3. Користувацькі інтерфейси 206
5.4. Практичні рекомендації 208
5.5. Впровадження результатів 210
Висновки до розділу 5 212
ВИСНОВКИ 215
Список використаних джерел 223
ДОДАТКИ 233
Додаток А Список публікацій здобувача за темою дисертації 233
Додаток Б Документи про впровадження результатів дисертаційної роботи 237
Б.1. Довідка про впровадження результатів дисертаційної роботи у навчальний процес НУ «Чернігівська політехніка» (№202/08-2180 від 20.10.2026) 237
Б.2. Довідка про впровадження результатів дисертаційної роботи при виконанні НДР №0124U003853 та №0126U001541 (№202/07-2176 від 19.10.2026) 238
Б.3. Акт про впровадження ТОВ «УАПРОМ» (№186 від 19.03.2026) 239
Б.4. Акт про впровадження LIMITED LIABILITY COMPANY «ADVERTISING AGENCY SATU.KZ» (No. 32 від 19.03.2026) 240

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ACID – Atomicity, Consistency, Isolation, Durability (Атомарність, узгодженість, ізольованість, довговічність)

ADWIN – Adaptive Windowing (Адаптивне вікно для детекції дрейфу)

AI – Artificial Intelligence (Штучний інтелект)

API – Application Programming Interface (Прикладний програмний інтерфейс)

ARIMA – Autoregressive Integrated Moving Average (Авторегресійна інтегрована модель ковзного середнього)

AWS – Amazon Web Services (Хмарні сервіси Amazon)

BERT – Bidirectional Encoder Representations from Transformers (Двонаправлені представлення кодувальника на основі трансформерів)

Bi-LSTM – Bidirectional Long Short-Term Memory (Двонаправлена довга короткочасна пам’ять)

BSON – Binary JSON (Двійковий JSON)

CAP – Consistency, Availability, Partition tolerance (Узгодженість, доступність, стійкість до розбиття)

CDC – Change Data Capture (Захоплення змін даних)

CP – Consistency + Partition tolerance (Узгодженість + стійкість до розбиття)

CRDT – Conflict-free Replicated Data Type (Безконфліктний реплікований тип даних)

DDL – Data Definition Language (Мова визначення даних)

DIS – Dynamic Integration System (Система динамічної інтеграції)

DQN – Deep Q-Network (Глибока Q-мережа)

ELT – Extract, Load, Transform (Витягування, завантаження, трансформація)

EML – Evolving Machine Learning (Еволюційне машинне навчання)

ETL – Extract, Transform, Load (Витягування, трансформація, завантаження)

FDW – Foreign Data Wrappers (Обгортки зовнішніх даних)

FIS – Fuzzy Inference System (Система нечіткого виведення)

FLP – Fischer, Lynch, Paterson (теорема про неможливість консенсусу)

GAN – Generative Adversarial Network (Генеративна змагальна мережа)

HPA – Horizontal Pod Autoscaler (Горизонтальний автомасштабувальник подів)

JSON – JavaScript Object Notation (Нотація об’єктів JavaScript)

LSTM – Long Short-Term Memory (Довга короткочасна пам’ять)

LWW – Last Writer Wins (Перемагає останній запис)

MAPE – Mean Absolute Percentage Error (Середня абсолютна відсоткова похибка)

MQL – MongoDB Query Language (Мова запитів MongoDB)

MVP – Minimum Viable Product (Мінімально життєздатний продукт)

MVCC – Multi-Version Concurrency Control (Багатоверсійне управління паралельним доступом)

OLTP – Online Transaction Processing (Оперативна обробка транзакцій)

PACELC – Partition: Availability/Consistency, Else: Latency/Consistency (розширення CAP-теореми)

PXC – Percona XtraDB Cluster (Кластер Percona XtraDB)

QPS – Queries Per Second (Запитів на секунду)

RC – Read Committed (рівень ізоляції транзакцій)

RMSE – Root Mean Square Error (Середньоквадратична похибка)

RR – Repeatable Read (рівень ізоляції транзакцій)

RS – Replica Set (Набір реплік)

S – Serializable (рівень ізоляції транзакцій)

СКБД – Система керування базами даних

SLA – Service Level Agreement (Угода про рівень обслуговування)

SQL – Structured Query Language (Структурована мова запитів)

SQL/MED – SQL/Management of External Data (SQL/Управління зовнішніми даними)

SSI – Serializable Snapshot Isolation (Серіалізована ізоляція знімків)

SST – State Snapshot Transfer (Передача знімку стану)

TCP – Transmission Control Protocol (Протокол управління передачею)

TLS – Transport Layer Security (Безпека транспортного рівня)

USL – Universal Scalability Law (Універсальний закон масштабованості)

WAL – Write-Ahead Log (Журнал попереднього запису)

2PC – Two-Phase Commit (Двофазна фіксація)

ВСТУП

Актуальність теми дослідження. Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів даних у корпоративних інформаційних системах. За оцінками аналітиків IDC, глобальний обсяг даних у 2025 році перевищив 175 зеттабайтів. При цьому значна частка цих даних зберігається та обробляється не в єдиній централізованій базі даних, а у розподілених системах із багатоваріантною персистентністю, де одночасно функціонують реляційні, документні, графові та key-value сховища [1]. Використання кількох різнорідних систем управління базами даних (СКБД) у межах однієї інфраструктури – багатоваріантна персистентність (polyglot persistence) – зумовлене об'єктивною різноманітністю типів даних та патернів доступу до них.

Як показано у роботі Клепманна [1], жодна окрема СКБД не може бути оптимальною для всіх типів робочих навантажень водночас: реляційні СКБД забезпечують ACID-транзакції, але поступаються документним у гнучкості схеми, тоді як key-value сховища мають найнижчу латентність, проте не підтримують складних запитів. У зв'язку з цим виникає потреба в інтеграції різнорідних СКБД у єдину систему із збереженням переваг кожної з них.

Перспективним інструментом такої інтеграції є технологія Foreign Data Wrappers (FDW) у PostgreSQL, що реалізує стандарт SQL/MED (Management of External Data, ISO/IEC 9075-9) та забезпечує доступ до зовнішніх джерел даних через SQL-інтерфейс [2]. Розширення postgres_fdw, mysql_fdw та mongo_fdw дозволяють використовувати PostgreSQL як координуючий вузол федеративної системи для виконання крос-системних запитів, зокрема JOIN-операцій між таблицями різних СКБД. Разом з тим практична реалізація FDW-інтеграції наштовхується на фундаментальні теоретичні обмеження, зумовлені особливостями розподілених систем.

Аналіз робіт Brewer [3], Gilbert та Lynch [4], Abadi [5], Gunther [6] та інших дослідників дозволяє зробити висновок, що деградація продуктивності у

розподілених системах баз даних із багатоваріантною персистентністю зумовлена щонайменше п’ятьма взаємопов’язаними теоретичними парадоксами:

1 **CAP-обмеження** – фундаментальна неможливість одночасного забезпечення узгодженості (Consistency), доступності (Availability) та стійкості до розбиття мережі (Partition tolerance), формально доведена Gilbert та Lynch у 2002 році. У контексті FDW-інтеграції ця теорема набуває особливої гостроти, оскільки координатор повинен одночасно працювати з СКБД, що реалізують різні CAP-компроміси: PostgreSQL Patroni (CP), Percona XtraDB Cluster (CP) та MongoDB (налаштовуваний CP/AP).

2 **Деградація від узгодженості** (Consistency-Performance Trade-off) – зворотна залежність між рівнем гарантій узгодженості та пропускною здатністю системи. Підвищення рівня ізоляції від Read Committed до Serializable суттєво знижує пропускну здатність через накладні витрати протоколів 2PC, Galera certification та SSI [7].

3 **Парадокс хеш-колізій** (Birthday Paradox) – нетривіальне зростання ймовірності колізій хеш-функцій при виконанні розподілених hash join операцій. Для стандартного 32-бітного хеш-простору ймовірність колізії досягає 50% вже при ~77 тис. елементів, що є типовим масштабом для корпоративних систем [8].

4 **Накладні витрати трьохзначної логіки SQL** – додаткові обчислювальні ресурси, що витрачаються на обробку NULL-значень у крос-системних JOIN-операціях між СКБД з різною семантикою невизначених значень. PostgreSQL, MySQL та MongoDB тлумачать NULL по-різному, і ця несумісність потребує додаткової нормалізації на координуючому вузлі.

5 **Нелінійна деградація масштабування** – зростання координаційних витрат при збільшенні кількості вузлів, що описується Universal Scalability Law (USL) Гюнтера [9]. За цим законом, при кількості вузлів більше ~50 додавання нових ресурсів не лише не покращує, а навіть погіршує продуктивність через квадратичне зростання витрат на когерентність.

6 Слід зазначити, що існуючі дослідження з моделювання продуктивності розподілених баз даних здебільшого фокусуються на одному або

двох із зазначених парадоксів ізольовано. Моделі на основі теорії масового обслуговування [10] адекватно описують мережеву деградацію, але не враховують специфіку FDW-координації. Моделі на основі CAP/PACELC-аналізу [5] враховують компроміси узгодженості, проте не інтегрують впливу хеш-колізій та NULL-логіки. Таким чином, у відомих дослідженнях відсутній комплексний мультиплікативний підхід, здатний об’єднати всі п’ять компонентів деградації у єдину калібровану математичну формалізацію.

Поряд із зазначеним, актуальним залишається питання автоматизованого управління продуктивністю розподілених систем баз даних із багатоваріантною персистентністю. Традиційні підходи – rule-based автомасштабування (AWS Auto Scaling, Kubernetes HPA) та моделі масового обслуговування – не забезпечують автономного прийняття рішень в умовах динамічних навантажень із Парето-розподіленими сплесками. Статичні правила масштабування призводять до осциляцій та не враховують парадоксів продуктивності, оскільки додавання вузла може погіршити ситуацію через зростання координаційних витрат. Методи глибокого навчання з підкріпленням (Deep Q-Network, DQN) [11, 12] відкривають можливості для побудови інтелектуальних агентів масштабування, здатних навчатися оптимальній стратегії управління ресурсами на основі взаємодії з середовищем.

Окрему проблему становить адаптація до структурного дрейфу даних. У реальних системах схеми баз даних еволюціонують: додаються нові поля, змінюються типи, з’являються нові колекції. Традиційні ETL-інструменти (Apache NiFi, Talend) потребують ручного втручання при кожній зміні схеми, що є неприйнятним для систем безперервного функціонування. Методи Evolving Machine Learning (EML) – клас алгоритмів потокового машинного навчання, здатних адаптуватися до concept drift без повного перенавчання, – дозволяють розв’язати цю проблему. Зокрема, застосування LSTM-мереж [13] для embedding-векторизації атрибутів у поєднанні з потоковою архітектурою Kafka+CapyMOA забезпечує адаптацію до змін схем у 15–20 разів швидше порівняно з традиційними ETL-рішеннями.

Таким чином, актуальність дисертаційного дослідження обумовлена: зростаючою поширеністю архітектур зберігання даних із багатоваріантною персистентністю у сучасних інформаційних системах; необхідністю комплексного математичного моделювання деградації продуктивності з урахуванням п’яти теоретичних парадоксів; потребою в автономних інтелектуальних агентах для динамічного управління продуктивністю; відсутністю адекватних механізмів адаптації до структурного дрейфу даних у розподілених системах із багатоваріантною персистентністю.

Зв’язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконана відповідно плану наукових досліджень кафедри інформаційних технологій та програмної інженерії Національного університету «Чернігівська Політехніка» в рамках науково-дослідної теми «Методи та моделі діджиталізації процесів розробки інформаційних систем для розвитку Smart City технологій» (номер державної реєстрації 0124U003853) та в рамках реалізації НДР «Розробка інтелектуальної моніторингової геоінформаційної платформи для територіальних громад постраждалих від збройного конфлікту та техногенних катастроф», що виконується за фінансуванням Міністерства освіти та науки України (державний реєстраційний номер: 0126U001541).

Мета і задачі дослідження. Мета дослідження – підвищення ефективності динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю шляхом розробки моделей та методів управління парадоксами деградації на основі технології PostgreSQL Foreign Data Wrappers та інтелектуальних AI-агентів. Для досягнення поставленої мети визначено такі *задачі дослідження*:

– провести системний аналіз існуючих підходів до моделювання продуктивності розподілених баз даних та виявити обмеження наявних моделей у контексті інтеграції різнотипних СКБД через механізм PostgreSQL FDW, з особливою увагою до комплексного впливу теоретичних парадоксів на продуктивність крос-системних запитів;

– розробити мультиплікативну математичну модель продуктивності розподілених баз даних із багатоваріантною персистентністю ($P_total = P_0 \cdot exp(-D_total)$), що інтегрує п’ять компонентів деградації – CAP-обмеження, деградацію узгодженості, хеш-колізії, NULL-логіку та мережеву деградацію – з калібруванням коефіцієнтів методами Bottom-Up аналітичної оцінки та Top-Down глобальної оптимізації;

– удосконалити метод адаптивного управління багатоваріантною персистентністю на основі EML-алгоритмів із автоматичною адаптацією до структурного дрейфу даних із використанням формалізму кортежів комірок (cell tuples), embedding-векторів (Bi-LSTM, BERT) та потокової архітектури Kafka+CapyMOA для автоматичної адаптації до структурного дрейфу даних;

– створити інтелектуальних AI-агентів для автономного масштабування (DQN Scaling Agent із 15-вимірним вектором стану та 5 дискретними діями) та адаптації схем даних (Schema Evolution Agent) у динамічних умовах із Парето-розподіленими навантаженнями;

– реалізувати та верифікувати програмну реалізацію інформаційної технології DIS (Dynamic Integration System) на реальних кластерах розподілених баз даних із багатоваріантною персистентністю (PostgreSQL Patroni, Percona XtraDB Cluster, MongoDB Replica Set) у контейнерній інфраструктурі з 13 Docker-контейнерів;

– розробити практичні рекомендації щодо вибору оптимальних параметрів узгодженості, визначення меж ефективного масштабування та застосування стратегій управління парадоксами для FDW-інтегрованих систем.

Об’єктом дослідження є процеси динамічної інтеграції розподілених баз даних ііз багатоваріантною персистентністю, що функціонують в умовах змінних навантажень та структурного дрейфу даних.

Предметом дослідження є математичні моделі, методи та програмні засоби управління у розподілених системах баз даних із багатоваріантною персистентністю на основі механізму PostgreSQL Foreign Data Wrappers.

Методи дослідження. У роботі застосовано комплекс методів дослідження, вибір яких зумовлений специфікою поставлених задач та міждисциплінарним характером предмету дослідження:

– *математичне моделювання* – для побудови мультиплікативної моделі деградації продуктивності з урахуванням п’яти теоретичних парадоксів розподілених систем. Модель базується на експоненціальній функції деградації $P_{total} = P_0 \cdot exp(-D_{total})$, де D_{total} є зваженою сумою п’яти компонентів, кожен з яких формалізує окремий парадокс;

– *теорія масового обслуговування* (зокрема, моделі M/M/n та M/G/1) – для моделювання мережевих затримок, координаційних витрат та насичення пропускної здатності у розподілених FDW-інтегрованих системах [10];

– *теорія ймовірностей та математична статистика* – для аналізу хеш-колізій (Birthday Paradox), стохастичного моделювання мережевих затримок (LogNormal + Exponential) та калібрування параметрів моделі;

– *нечітка логіка (Fuzzy Inference System)* – для моделювання неперервного спектру CAP-компромісів в умовах невизначеності мережевих розбиттей;

– *глибоке навчання з підкріпленням (Deep Q-Network)* – для побудови автономного агента масштабування, здатного оптимізувати багатокритеріальну функцію винагороди в 15-вимірному просторі станів;

– *адаптивне потокове машинне навчання (Evolving Machine Learning із використанням CapyMOA, ADWIN, Hoeffding Trees) [14]* – для виявлення та адаптації до дрейфу концепцій у потоках даних з гетерогенних джерел;

– *рекурентні нейронні мережі (LSTM) та класичні статистичні моделі (ARIMA)* – для прогнозування навантаження та проактивного масштабування [13, 15];

– *емпірична валідація* – для експериментального підтвердження теоретичних результатів на реальних кластерах розподілених баз даних із багатоваріантною персистентністю із застосуванням метрик R^2, RMSE, MAPE, кореляції Пірсона та 5-fold cross-validation.

Наукова новизна одержаних результатів. У дисертаційній роботі отримано такі нові наукові результати:

1) ***вперше розроблено*** мультиплікативну математичну модель продуктивності розподілених баз даних із багатоваріантною персистентністю виду $P_{total} = P_0 \cdot exp(-D_{total})$, яка, на відміну від існуючих моделей, інтегрує п'ять компонентів деградації у єдину калібровану формалізацію, що дозволяє кількісно оцінити сукупний внесок парадоксів у деградацію продуктивності з точністю $R^2 = 0.895$, MAPE = 12.45%;

2) ***удосконалено*** метод адаптивного управління інтеграції розподілених баз даних із багатоваріантною персистентністю шляхом розробки моделі на основі EML-алгоритмів із використанням кортежів комірок (cell tuples), embedding-векторів (Bi-LSTM, BERT) та потокової архітектури Kafka+CapyMOA, що, на відміну від традиційних ETL-систем (Apache NiFi), забезпечує адаптацію до структурного дрейфу у 15–20 разів швидше при F1-score зіставлення атрибутів 0.87;

3) ***дістав подальший розвиток*** метод автономного масштабування розподілених баз даних через створення DQN-агента з 15-вимірним вектором стану та 5 дискретними діями, інтегрованого з п'ятьма стратегіями управління парадоксами (диференційована консистентність, апгрейд хеш-простору, COALESCE-обгортки, connection pooling, graceful degradation), що у комбінації дозволяє підвищити продуктивність системи з 50.3% до 93.6% під час пікових навантажень (при цьому ефект лише стратегій без масштабування становить 50.3% → 64%, а лише масштабування без стратегій – 50.3% → 80%);

4) ***вперше запропоновано*** комплексну архітектуру інформаційної технології динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю (DIS), що поєднує FDW-координатор (PostgreSQL, MySQL, MongoDB), математичну модель прогнозування деградації, AI-агентів масштабування та EML-систему адаптації схем у єдиному програмному рішенні із контейнерною інфраструктурою (13 Docker-контейнерів).

Практичне значення одержаних результатів полягає у наступному:

– ***розроблено*** програмну систему DIS MVP (Dynamic Integration System – Minimum Viable Product), що реалізує інтеграцію розподілениз баз даних (PostgreSQL Patroni, Percona XtraDB Cluster, MongoDB Replica Set) через FDW-координатор із автоматичним управлінням продуктивністю. Система розгортається як 13 Docker-контейнерів із загальним споживанням приблизно 6.5 GB оперативної пам'яті та може бути адаптована для промислового використання;

– ***запропоновано*** практичні рекомендації щодо вибору оптимального рівня узгодженості для FDW-інтегрованих систем: діапазон $c \in [0.3,0.5]$ (відповідає Read Committed) забезпечує найкращий компроміс між гарантіями узгодженості та продуктивністю, знижуючи деградацію від парадоксу узгодженості до прийнятного рівня;

– ***встановлено*** межі ефективного масштабування: при кількості вузлів $n > 50$ нелінійна деградація масштабування стає домінуючим фактором, а ефективність масштабування падає нижче 50 % через зростання координаційних витрат згідно з Universal Scalability Law [6];

– результати дисертації ***впроваджено*** у навчальний процес кафедри інформаційних технологій та програмної інженерії Національного університету «Чернігівська Політехніка» при викладанні дисциплін «Засоби інтеграції розподілених систем», «Бази даних», «Адміністрування баз даних», «Інтелектуальний аналіз даних», при виконанні науково-дослідної роботи №0124U003853 «Методи та моделі діджиталізації процесів розробки інформаційних систем для розвитку Smart City технологій», державного проекту прикладного дослідження №0126U001541 «Розробка інтелектуальної моніторингової геоінформаційної платформи для територіальних громад постраждалих від збройного конфлікту та техногенних катастроф», у комерційній діяльності Товариства з обмеженною відповідальністю „УАПРОМ" та діяльності LIMITED LIABILITY COMPANY „ADVERTISING AGENCY SATU.KZ", що підтверджується відповідним актами впровадження

Особистий внесок здобувача. Основні наукові результати, представлені у дисертації, отримані автором самостійно. У роботах, опублікованих у співавторстві, здобувачу належать:

- у роботі [16] (у співавторстві з Білоус І., Труновою О., Скітером І., Войцеховською М.) – розробка математичної моделі (формула 2.35), проведення обчислювальних експериментів із 60 мільярдами запитів, калібрування коефіцієнтів моделі методами Bottom-Up та Top-Down оптимізації, аналіз та інтерпретація результатів;
- у роботі [17] (у співавторстві з Білоус І.) – розробка методу автоматизованої міграції даних між варіантами сховищ, системи інваріантів структури, типів і семантики та предметно-орієнтованої мови опису трансформацій;
- у роботі [18] (у співавторстві з Білоус І.) – розробка архітектури відмовостійкої системи із багатоваріантною персистентністю на основі BFT-консенсусу, сценаріїв реплікації та протоколів узгодження реплік;
- у роботі [19] (у співавторстві з Білоус І.) – розробка агентного підходу до динамічної синхронізації схем у розподілених базах даних, трьохканального механізму виявлення змін схеми та алгоритму автоматичної адаптації FDW;
- робота [20] опублікована здобувачем одноосібно.

Апробація результатів дисертації. Основні наукові та практичні результати дисертаційної роботи доповідались та обговорювались на 10 наукових конференціях та семінарах:

– Патлань Є. Ю. Comparative Analysis of Distributed Database Clusters. *Юність науки – 2024* : зб. тез доп. XIV Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 24–26 квіт. 2024 р.). Чернігів : НУ «Чернігівська політехніка», 2024.

– Патлань Є. Ю. Distributed Database Dilemmas: Paradoxes That Shape Modern Information Systems. *Юність науки – 2025* : зб. тез доп. XV Міжнар. наук.-

практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 23–25 квіт. 2025 р.). Чернігів : НУ «Чернігівська політехніка», 2025.

– Патлань Є. Ю. PostgreSQL Job Queues versus Valkey Streams: a Short Empirical Report. *Юність науки – 2026* : зб. тез доп. XVI Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 22–23 квіт. 2026 р.). Чернігів : НУ «Чернігівська політехніка», 2026.

– Patlan Y., Bilous I. Estimating performance, consistency, and scalability in dynamic database integration and clustering through modeling. *Математичне та імітаційне моделювання систем. МОДС 2024* : тези доп. XIX Міжнар. наук.-практ. конф. (м. Чернігів, 2024 р.). Чернігів : НУ «Чернігівська політехніка», 2024. С. 107–109.

– Patlan Ye. Yu. Paradoxes of distributed database systems: identification and classification. *Innovation and development in world science* : Proceedings of the 11th International Scientific and Practical Conference (Zurich, August 17–19, 2026). Zurich : MDPC Publishing, 2026. P. 21–26. ISBN 978-3-954753-21-5.

– Patlan Ye. Integration mechanisms for distributed databases with polyglot persistence: a comparative classification. *Science, technology and global challenges* : Proceedings of the 13th International Scientific and Practical Conference (Tokyo, August 20–22, 2026). Tokyo : CPN Publishing Group, 2026. P. 35–40. ISBN 978-4-9783419-7-6.

– Patlan Ye. Divergent NULL semantics in federated queries over heterogeneous data stores. *European science and innovation congress* : Proceedings of the 10th International Scientific and Practical Conference (Barcelona, August 24–26, 2026). Barcelona : Barca Academy Publishing, 2026. P. 34–37. ISBN 978-84-15927-36-5.

– Patlan Ye. Paradox as a category of scientific inquiry in database theory. *Modern science: challenges and prospects* : Proceedings of the 1st International Scientific and Practical Conference (London, August 28–30, 2026). London : Cognum Publishing House, 2026. P. 64–68. ISBN 978-92-9472-198-3.

– Patlan Ye. The mean is not the system: heavy-tailed latency in the design of federated databases. *Global perspectives in scientific research* : Proceedings of the 1st International Scientific and Practical Conference (Boston, September 4–6, 2026). Boston : BoScience Publisher, 2026. P. 47–50. ISBN 978-1-73981-120-4.

– Patlan Ye. Hidden coupling in relational database systems: local optimization as a source of global degradation. *Contemporary trends in modern science* : Proceedings of the 1st International Scientific and Practical Conference (Munich, September 7–9, 2026). Munich : MDPC Publishing, 2026. P. 50–53. ISBN 978-3-954753-07-9.

Публікації. За темою дисертації опубліковано 5 наукових праць, у тому числі 4 статті у наукових фахових виданнях України, 1 стаття у міжнародному виданні у форматі Springer LNCS, що індексується Scopus та WOS:

1. Patlan Y., Bilous I., Trunova O., Skiter I., Voitsekhovska M. Mathematical Modeling of Performance-Consistency Trade-offs and Theoretical Paradoxes in Distributed Database Integration Using PostgreSQL FDW // Lecture Notes in Computer Science. Springer, 2026. 21 p. (прийнято до друку)
2. Патлань Є.Ю., Білоус І.В. Метод автоматизованої міграції даних між варіантами сховищ у системах із багатоваріантною персистентністю // Інформаційні технології та суспільство. 2025. Вип. 4(19). С. 129–139. DOI: 10.32689/maup.it.2025.4.21.
3. Патлань Є.Ю. Модель багатоваріантної персистентності у середовищі динамічно інтегрованих розподілених баз даних // Наука і Техніка Сьогодні. 2025. №8(49). С. 1623–1633. DOI: 10.52058/2786-6025-2025-8(49)-1623-1633.
4. Патлань Є.Ю., Білоус І.В. Сценарії використання багатоваріантної персистентності для підтримки fault-tolerance у розподілених БД // Таврійський науковий вісник. Серія: Технічні науки. 2025. № 4, ч. 1. С. 227–234. DOI: 10.32782/tnv-tech.2025.4.1.24.

5. Patlan Y., Bilous I. Agent-based approach to dynamic schema synchronization in distributed databases // Transactions of Kremenchuk Mykhailo Ostrohradskyi National University. 2026. Vol. 1, No. 1(156). P. 131–144. DOI: 10.32782/1995-0519.2026.1.1.17.

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, п’яти розділів, висновків, списку використаних джерел та додатків. Повний обсяг дисертації становить 240 сторінок, у тому числі 205 сторінок основного тексту, 39 рисунків, 77 таблиць. Список використаних джерел містить 96 найменувань і розміщений на 10 сторінках. Додатки розміщено на 8 сторінках.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМ УПРАВЛІННЯ ТА ЗАБЕЗПЕЧЕННЯ БАГАТОВАРІАНТНОЇ ПЕРСИСТЕНТНОСТІ У РОЗПОДІЛЕНИХ БАЗАХ ДАНИХ

У першому розділі проведено аналіз еволюції розподілених СКБД та polyglot persistence архітектур, досліджено PostgreSQL FDW як технологію федеративної інтеграції, ідентифіковано п’ять парадоксів продуктивності, проаналізовано існуючі моделі та методи адаптивного управління, а також сформульовано наукову проблему та задачі дослідження.

1.1 Аналіз сучасних розподілених баз даних та їх архітектури

1.1.1 Еволюція парадигм зберігання даних: від монолітних СКБД до розподілених систем

Еволюція СКБД пройшла через чотири ери: навігаційні бази (1960–70-ті), реляційна модель Кодда (1970) [21], NoSQL (кінець 2000-х) та сучасні polyglot/мультимодельні архітектури [1]. Перехід між ерами не означав відмирання попередніх парадигм – реляційні СКБД додали JSON (PostgreSQL JSONB), а NoSQL-системи набули рис реляційних (транзакції MongoDB). Але конвергенція не усунула фундаментальних відмінностей між моделями даних. Як зазначив Стоунбрейкер [22], “one size does not fit all” – спеціалізована СКБД завжди випереджатиме мультимодельну у своїй ніші.

Питанням побудови та управління розподіленими системами баз даних присвячено роботи як міжнародних, так і вітчизняних дослідників. Серед українських вчених слід відзначити дослідження Корнаги Я.І. та Мухіна В.Є. [23, 24, 25], які запропонували моделі організації та управління розподіленими базами даних з динамічною структурою на основі мережецентричного підходу, включаючи моніторинг подій на основі векторно-матричних операцій. Питання концептуального моделювання баз даних ііз багатоваріантною персистентністю досліджено у роботі Колонко М. [26], де розроблено мову AGILA MOD+ та методи автоматизації проєктування polyglot persistence систем. Верес О.М. та

Оливко Р.М. [27] дослідили методи аналізу даних у розподілених файлових системах, а Берко А.Ю., Верес О.М. та Пасічник В.В. [28] систематизували сучасні підходи до організації систем баз даних та знань. Проблематику захисту розподілених інформаційних систем висвітлено у монографії Литвинова В.В., Скітера І.С., Трунової О.В. та ін. [29]. Питання розробки та супроводження систем із розподіленими базами даних на основі технологій polyglot persistence досліджено Білоус І., Петренком Д. та Єрмоленком О. [30]. Гречанінов В.Ф. [31] розглянув аналіз і проектування розподілених систем на основі кластерних технологій. Хошаба О.М., Литвинов В.В. та ін. [32, 33] дослідили методи балансування навантаження у кластерних та хмарних інфраструктурах, а також моделі впливу навантаження на продуктивність баз даних. Разом з тим, у зазначених роботах не розглядається комплексне моделювання деградації продуктивності з урахуванням усіх п’яти теоретичних парадоксів у контексті FDW-інтеграції різнотипних СКБД.

Відповідно до класифікації Cattell [34], доповненої у пізніших роботах [35, 36], сучасні розподілені СКБД поділяються на п’ять основних категорій:

1) ***реляційні*** (PostgreSQL, MySQL, Oracle, CockroachDB) – забезпечують ACID-транзакції, складні SQL-запити з JOIN-операціями, нормалізовані схеми та математично обґрунтовану реляційну алгебру;

2) ***документні*** (MongoDB, Couchbase, Amazon DocumentDB) – зберігають дані у вигляді JSON/BSON-документів із гнучкою схемою, оптимізовані для ієрархічних структур;

3) ***ключ-значення*** (Redis, Valkey, Amazon DynamoDB, Riak) – забезпечують найшвидший доступ за ключем із суб-мілісекундною латентністю [37];

4) ***графові*** (Neo4j, Amazon Neptune, JanusGraph) – оптимізовані для обходу зв’язків та виконання графових запитів (shortest path, pattern matching);

5) ***колонкові*** (Apache Cassandra, HBase, ClickHouse) – ефективні для аналітичних запитів на великих обсягах даних завдяки колонковому стисненню та відкладеному зчитуванню [38].

Мультимодельні СКБД (ArangoDB, CosmosDB) намагаються об’єднати всі моделі, але Pavlo та ін. [39] показали, що це неминуче супроводжується компромісами у продуктивності.

PostgreSQL обрано координуючим вузлом на підставі таких критеріїв:

1) найрозвиненіша екосистема FDW-розширень серед існуючих СКБД;
2) відкритий код, що виключає vendor lock-in;
3) повна підтримка стандарту SQL, що спрощує трансляцію запитів;
4) висока доступність через Patroni;
5) активна open-source спільнота.

1.1.2. PostgreSQL FDW як технологія інтеграції розподілених БД із багатоваріантною персистентністю

Foreign Data Wrappers – механізм PostgreSQL, що реалізує стандарт SQL/MED (ISO/IEC 9075-9:2008) і дозволяє працювати з даними зовнішніх джерел як із локальними таблицями [2]. Архітектура FDW побудована на чотирьох базових абстракціях, взаємодію яких ілюструє рис. 1.1.

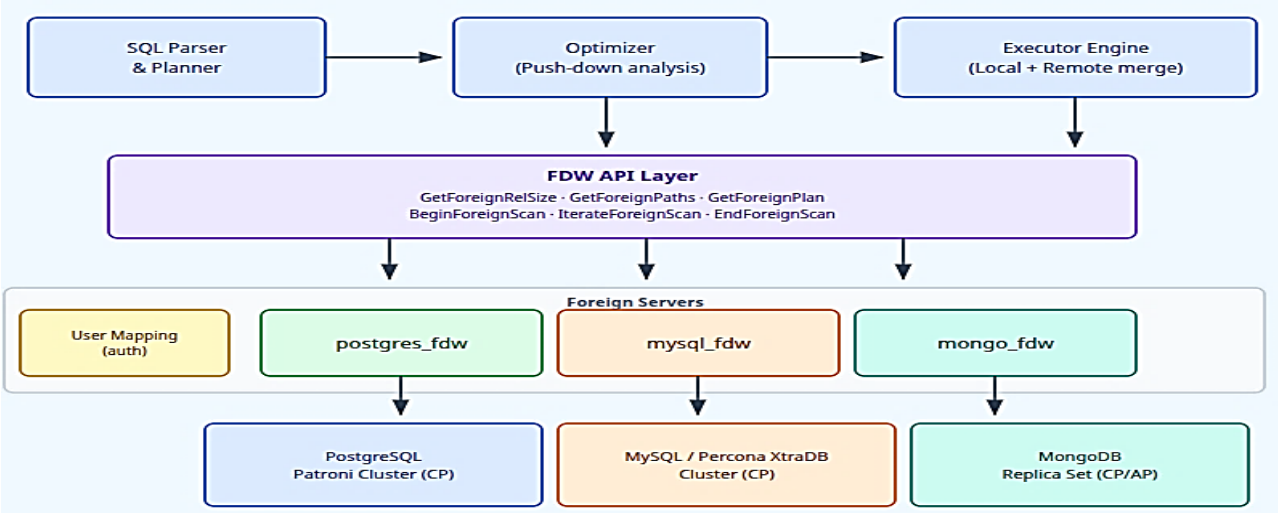


Рисунок 1.1 – Архітектура FDW-інтеграції розподілених баз даних із багатоваріантною персистентністю

Архітектура FDW побудована на чотирьох базових абстракціях:

1) ***Foreign Data Wrapper*** – програмний модуль, що транслює SQL-запити у формат зовнішньої СКБД та перетворює результати назад у PostgreSQL tuples;
2) ***Foreign Server*** – інкапсулює параметри з’єднання (хост, порт, ім’я БД);

3) ***User Mapping*** – механізм автентифікації, що відображає локальних користувачів на облікові записи зовнішнього сервера;

4) ***Foreign Table*** – віртуальна таблиця у локальній схемі, що представляє зовнішні дані.

Виконання запиту проходить три фази. На етапі **планування** оптимізатор визначає, які частини можуть бути «відштовхнуті» (pushed down) на зовнішній сервер – WHERE-умови, LIMIT, ORDER BY та агрегації [40]. На етапі **виконання** FDW-модуль встановлює з'єднання та передає оптимізований запит. Для крос-серверних з'єднань координатор отримує дані від кількох серверів та виконує JOIN локально – при великих обсягах це створює значне навантаження на мережу та пам'ять.

Затримка крос-БД JOIN через FDW (256 мс) у 6.4 рази перевищує локальний запит (40 мс): FDW overhead ~50 мс, мережеві round-trips 3×~15 мс, виконання на джерелах ~100 мс, локальний JOIN ~60 мс.

Три FDW-розширення системи: ***postgres_fdw*** (повний push-down, 2PC-транзакції [7]), ***mysql_fdw*** (базовий push-down WHERE, обмежена сумісність типів) та ***mongo_fdw*** (трансляція SQL→MQL, без транзакційної семантики). Порівняльна характеристика вказаних розширень представлена в табл. 1.1.

Таблиця 1.1

Порівняльна характеристика FDW-розширень

Характеристика	postgres_fdw	mysql_fdw	mongo_fdw
Push-down WHERE	Повний	Базовий	Базовий
Push-down ORDER BY	Так	Ні	Ні
Push-down LIMIT	Так	Так	Так
Push-down агрегацій	Так (PG 14+)	Ні	Ні
JOIN push-down	Так (один сервер)	Ні	Ні
Операції запису	INSERT/UPDATE/ DELETE	INSERT/UPDATE/ DELETE	INSERT/UPDATE/ DELETE
Транзакції (2PC)	Так	Ні	Ні
Типи даних	Повна сумісність	Часткова	Часткова (BSON→SQL)
NULL-семантика	Стандартна SQL	MySQL-специфічна	Відсутність поля ≠ null

Обмеження mysql_fdw та mongo_fdw безпосередньо впливають на продуктивність крос-системних запитів і моделюються у Розділі 2.

1.1.3. Моделі зберігання: реляційні, документні, key-value та їх порівняльний аналіз

Реляційна модель: PostgreSQL Patroni та Percona XtraDB Cluster

PostgreSQL – об'єктно-реляційна СКБД із ACID-транзакціями через MVCC (Multi-Version Concurrency Control): замість блокування рядків при читанні створюються версії, і кожна транзакція бачить snapshot даних [41]. PostgreSQL Patroni забезпечує високу доступність з автоматичним failover через etcd (10–30 с, CP-семантика).

Percona XtraDB Cluster (PXC) реалізує синхронну мульти-мастер реплікацію на основі протоколу Galera. На відміну від Patroni, PXC дозволяє записи на будь-якому вузлі: кожен write set реплікується на всі вузли та проходить certification test. Якщо write set конфліктує із зафіксованою транзакцією на іншому вузлі, він відхиляється. Це забезпечує строгу узгодженість ціною вищої латентності [42]. Порівняння підходів до реплікації у реляційних СКБД представлено у табл. 1.2.

Таблиця 1.2

Порівняння підходів до реплікації у реляційних СКБД

Характеристика	PostgreSQL Patroni	Percona XtraDB Cluster
Топологія	Primary-Replica	Multi-Master
Тип реплікації	Streaming (sync/async)	Galera (synchronous)
Запис	Тільки primary	Будь-який вузол
Виявлення конфліктів	Lock-based (MVCC)	Certification-based
Failover	Автоматичний (etcd, 10-30 с)	Автоматичний (Galera, <1 с)
Латентність запису	Нижча (async) / Вища (sync)	Стабільно висока (certification)
CAP-компроміс	CP	CP
Масштабування читання	Горизонтальне (replicas)	Горизонтальне (всі вузли)

Документна модель: MongoDB Replica Set

MongoDB зберігає дані у форматі BSON (Binary JSON) із гнучкою схемою [43] – документи в одній колекції можуть мати різну структуру. MongoDB

Replica Set забезпечує високу доступність з налаштовуваною узгодженістю через Write Concern: w: 1 – підтвердження лише від primary (AP); w: majority – підтвердження від більшості (CP); Read Concern “majority” – linearizable reads. Починаючи з версії 4.0, MongoDB підтримує multi-document ACID-транзакції, проте вони не підтримуються через FDW-інтерфейс.

Модель key-value: Redis/Valkey

Redis/Valkey реалізують key-value зберігання зі складними структурами даних (strings, lists, sets, sorted sets, hashes) та суб-мілісекундною латентністю завдяки in-memory storage [37]. У контексті DIS використовуються як кеш-рівень для зменшення навантаження на «важкі» СКБД. Redis Cluster масштабується через шардинг на основі 16384 хеш-слотів [44].

1.1.4. Концепція багатоваріантної персистентності (Polyglot Persistence)

Polyglot persistence – архітектурний патерн, при якому різні типи даних зберігаються у СКБД, найбільш пристосованих для відповідного навантаження. Мультимодельні СКБД (ArangoDB, CosmosDB) намагаються об’єднати всі моделі, але Pavlo та ін. [39] показали неминучі компроміси у продуктивності. Типова архітектура для e-commerce має наступний вигляд (рис. 1.2):

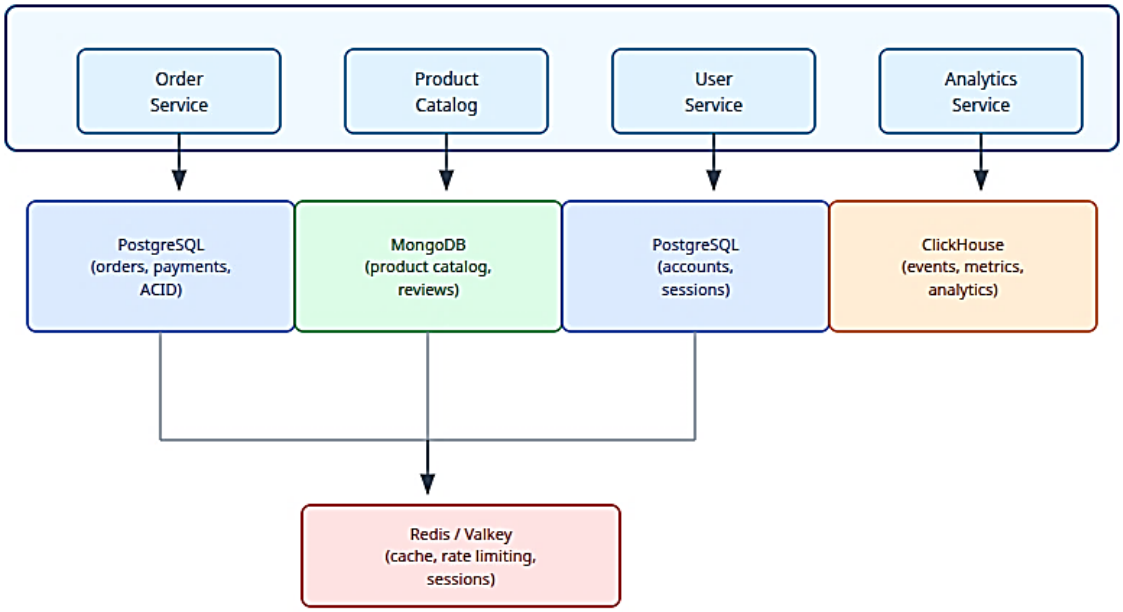


Рисунок. 1.2 – Типова архітектура polyglot persistence для e-commerce системи

У системі Amazon [37] дані про замовлення зберігаються у реляційних СКБД (ACID-гарантії), каталог товарів – у документній базі із гнучкою схемою. Google Bigtable [38] використовує колонкову модель для веб-індексу.

Класифікація підходів до polyglot persistence наступна:

–***комплементарний підхід*** – кожна СКБД обслуговує окремий домен даних. Переваги: мінімальна координація. Недоліки: складність крос-доменних запитів.

–***редундантний підхід*** – ті ж дані зберігаються у кількох СКБД для оптимізації різних типів доступу. Переваги: оптимальна продуктивність. Недоліки: необхідність синхронізації, ризик неузгодженості [45].

–***гібридний підхід*** – поєднання обох із інтеграційним рівнем (FDW-координатор, шина повідомлень). Саме цей підхід обрано у нашому дослідженні.

Polyglot persistence створює проблему інтеграції: без єдиного інтерфейсу кожен мікросервіс мусить взаємодіяти з кожною СКБД окремо. Порівняльний аналіз стратегій polyglot persistence зведено у табл. 1.3.

Таблиця 1.3

Порівняльний аналіз стратегій polyglot persistence

Характеристика	Комплементарний	Редундантний	Гібридний (FDW)	Мультимодельна СКБД
Крос-доменні запити	Складно (API)	Середньо	Просто (SQL)	Нативно
Узгодженість	Eventual	Потребує синхронізації	Керована (рівні)	Strong
Масштабованість	Незалежна	Обмежена синхронізацією	Гнучка (до ~50 вузлів)	Єдина СКБД
Продуктивність	Оптимальна по доменах	Оптимальна по типах	Компроміс (−19.3%)	Компроміс
Vendor lock-in	Відсутній	Низький	Низький	Високий
Складність DevOps	Висока	Дуже висока	Середня	Низька

PostgreSQL FDW пропонує єдиний SQL-інтерфейс через foreign tables, але із деградацією продуктивності від п’яти теоретичних парадоксів.

1.2 Аналіз теоретичних обмежень розподілених систем: п’ять парадоксів продуктивності

Продуктивність розподілених баз даних обмежується фундаментальними теоретичними результатами, що встановлюють принципову неможливість одночасного забезпечення всіх бажаних властивостей [46]. У контексті FDW-інтеграції, де координуючий PostgreSQL мусить працювати з кількома різнорідними СКБД, ми ідентифікували п’ять ключових парадоксів, кумулятивний вплив яких визначає загальну деградацію. Взаємодію цих парадоксів ілюструє рис. 1.3.

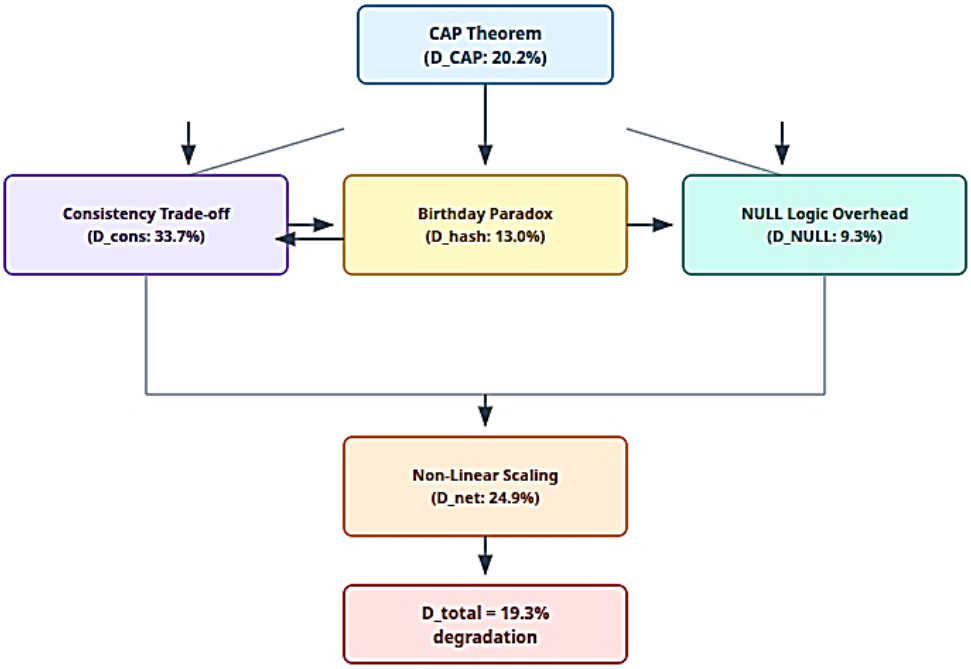


Рисунок 1.3 – Взаємодія п’яти парадоксів продуктивності та їх відносний внесок у загальну деградацію

Як бачимо, кожен з парадоксів має різний відносний внесок у загальну деградацію.

1.2.1 CAP-теорема та її практичні наслідки для FDW-інтеграції

CAP-теорема (Brewer [3], формально доведена Gilbert та Lynch [4]) стверджує: у розподіленій системі неможливо одночасно забезпечити три властивості:

1) ***Consistency (C)*** – кожне читання повертає або найновіше записане значення, або помилку. Це так звана лінеаризовуваність (linearizability) – найсуворіша форма узгодженості;

2) ***Availability (A)*** – кожен запит до працюючого (non-failing) вузла отримує відповідь, що не є помилкою (хоча, можливо, з не найновішими даними);

3) ***Partition tolerance (P)*** – система продовжує функціонувати при довільній втраті повідомлень між вузлами.

Оскільки мережеві розбиття неминучі у реальних системах (роутери відмовляють, хмарні зони ізолюються), система змушена обирати між CP та AP [47]. CAP – не бінарний вибір, а спектр поведінки. Розширення PACELC (Abadi [5]) враховує це:

– якщо мережеве розбиття (P): обирати між A та C;

– інакше (E): обирати між латентністю (L) та узгодженістю (C).

Для FDW-інтегрованих систем навіть у «нормальному» режимі кожен крос-системний запит потребує координації між трьома СКБД – постійний компроміс L/C.

У системі DIS CAP-обмеження проявляються на трьох рівнях:

– ***рівень окремих СКБД***. PostgreSQL Patroni реалізує CP: при мережевому розбитті вузли без зв’язку з лідером не приймають записи. PXC також CP: без кворуму – non-operational. MongoDB гнучка: w:majority→CP, w:1→AP [48];

– ***рівень FDW-координатора***. За замовчуванням CP-семантика (помилка при недоступності foreign server). Альтернатива – graceful degradation, що повертає часткові результати (Розділ 4);

– ***рівень крос-системної транзакції***. Лише postgres_fdw підтримує 2PC, а mysql_fdw та mongo_fdw – ні, тому крос-системна транзакція не гарантує атомарності.

Для математичного моделювання впливу CAP-обмежень у роботі [17] запропоновано використання нечіткої системи виведення (Fuzzy Inference System):

$$D_{CAP}(t) = k_{CAP} \cdot \mu_{CAP}(c(t), \sigma_t), \quad (1.1)$$

де k_{CAP} – калібрувальний коефіцієнт;

μ_{CAP} – функція нечіткої системи виведення Мамдані;

$c(t)$ – рівень узгодженості;

σ_t – критичність мережевих розбиттей.

Відносний внесок CAP-парадоксу: 20.2% (3.9% абсолютного впливу).

В табл. 1.4 наведемо класифікацію СКБД за CAP-компромісами та PACELC-поведінкою.

Таблиця 1.4

Класифікація СКБД за CAP-компромісами та PACELC-поведінкою

СКБД	CAP	PACELC	Поведінка при розбитті	Поведінка без розбиття
PostgreSQL Patroni	CP	PC/EC	Primary read-only, replica promoted	Low latency, strong consistency
Percona XtraDB (Galera)	CP	PC/EC	Non-operational without quorum	Higher latency (certification)
MongoDB (w:majority)	CP	PC/EC	Stepdown, election (~10s)	Moderate latency (majority ack)
MongoDB (w:1)	AP	PA/EL	Continues serving stale reads	Low latency, eventual consistency
Cassandra (LOCAL_QUORUM)	AP	PA/EL	Serves from available nodes	Tunable consistency per query
Redis Cluster	AP	PA/EL	Split-brain possible	Sub-ms latency
DynamoDB	AP	PA/EL	Available, eventually consistent	Configurable (strong/eventual)
CockroachDB	CP	PC/EC	Unavailable without majority	Raft consensus overhead

У нашій системі DIS всі три основні СКБД (PostgreSQL Patroni, PXC Galera, MongoDB з w:majority) обирають CP-компроміс – узгодженість ціною доступності. Це свідомий архітектурний вибір для транзакційних систем, де eventual consistency неприйнятна.

Мережеві розбиття мають різну severity – від часткової втрати пакетів до повної ізоляції зони доступності. Для моделювання цього неперервного спектру використовуємо параметр $\sigma_t \in [0,1)$, а не бінарний індикатор – унікальна особливість нашої моделі. FLP-теорема [47] формально доводить неможливість гарантованого консенсусу в асинхронній системі, що пояснює, чому протоколи

(Paxos, Raft, Galera) потребують таймаутів і безпосередньо впливає на латентність FDW-координації.

1.2.2. Проблема узгодженості та продуктивності (Consistency-Performance Trade-off)

Чим суворіші гарантії узгодженості, тим більше координації між вузлами потрібно, що неусувно збільшує латентність [7]. Забезпечення узгодженості - це завжди повідомлення по мережі, а повідомлення - це завжди затримка. Стандарт SQL визначає чотири рівні ізоляції, кожен з яких потребує різного обсягу координації [41]:

1) ***Read Uncommitted*** – мінімальна ізоляція; можливе «брудне» читання (dirty reads). У PostgreSQL цей рівень фактично не реалізований - він автоматично підвищується до Read Committed;

2) ***Read Committed*** – транзакція бачить тільки зафіксовані дані. Стандартний рівень ізоляції PostgreSQL. Потребує мінімальної координації і забезпечує найкращий баланс між узгодженістю та продуктивністю;

3) ***Repeatable Read*** – гарантує, що повторне читання в межах однієї транзакції повертає ті самі дані. Потребує snapshot isolation, що збільшує витрати пам’яті;

4) ***Serializable*** – найсуворіший рівень. Гарантує, що результат паралельного виконання транзакцій еквівалентний деякому послідовному виконанню. PostgreSQL реалізує цей рівень через Serializable Snapshot Isolation (SSI) [49], який потребує додаткового відстеження залежностей між транзакціями та може відхиляти (abort) транзакції при виявленні потенційних порушень серіалізовуваності [50].

Координаційні витрати у розподілених системах:

– ***двофазне фіксування (2PC):*** $4n$ повідомлень для n учасників [46];

– ***Paxos/Raft консенсус*** [51]***:*** підтвердження від більшості $(f + 1)$ з $2f + 1$ вузлів;

– ***Galera certification*****:** кожен write set проходить certification на всіх вузлах.

Підвищення рівня ізоляції з Read Committed до Serializable призводить до суттєвого зниження пропускної здатності, збільшення латентності та зростання кількості відхилених транзакцій через додаткові витрати на координацію [48]. Математична модель деградації від узгодженості представлена формулою (1.2):

$$D_{cons}(t) = \alpha \cdot \frac{l(t)}{l_{max}} \cdot c(t)^2 \cdot \sqrt{n}, \qquad (1.2),$$

де α – калібрувальний коефіцієнт;

$l\frac{(t)}{l_{max}}$ – відносне навантаження;

$c(t)$ – рівень узгодженості;

n – кількість вузлів.

Квадратична залежність від $c(t)$ відображає нелінійне зростання координаційних витрат, а залежність від $\sqrt{n}$ – збільшення кількості координаційних повідомлень зі зростанням кластера.

Оптимальний діапазон, визначений за результатами наших експериментів: c∈[0.3,0.5], що відповідає Read Committed. При $c < 0.3$ ризик аномалій стає неприйнятним для транзакційних систем, а при $c > 0.5$ деградація продуктивності перевищує допустимі межі.

Наведемо числовий приклад. FDW-координатор виконує крос-системний запит між таблицею customers (PostgreSQL, 10 млн рядків), orders (PXC, 50 млн рядків) та product_details (MongoDB, 1 млн документів). При Read Committed ($c = 0.3$), $n = 6$, $\frac{l}{l_{max}} = 0.7$:

$$D_{cons} = 0.045 \cdot 0.7 \cdot 0.3^2 \cdot \sqrt{6} = 0.045 \cdot 0.7 \cdot 0.09 \cdot 2.449 \approx 0.007\backslash(0.7\%\backslash).$$

При Serializable ($c = 1.0$):

$$D_{cons} = 0.045 \cdot 0.7 \cdot 1.0^2 \cdot \sqrt{6} = 0.045 \cdot 0.7 \cdot 1.0 \cdot 2.449 \approx 0.077\backslash(7.7\%\backslash).$$

Різниця в 11 разів обґрунтовує стратегію «диференційованої консистентності» (різні рівні для різних типів запитів).

Оптимальна зона функціонування: $c \in [0.3,0.5)$. При $c < 0.3$ зростає кількість аномалій конкурентного доступу. При $c > 0.5$ координаційні витрати зростають непропорційно через квадратичну залежність c^2. Bailis та ін. [48] підтверджують: перехід RC→S знижує throughput TPC-C на 47%, Vogels [45] зазначає, що

strongly consistent reads у DynamoDB подвоюють латентність. Відносний внесок: 33.7% (6.5% абсолютного впливу) – найбільший серед п'яти парадоксів.

1.2.3. Проблема хеш-колізій: Birthday Paradox у розподілених JOIN-операціях

Birthday Paradox – контрінтуїтивний результат теорії ймовірностей: у групі з 23 осіб ймовірність однакового дня народження перевищує 50%. При виконанні hash joins [52] колізії хеш-значень збільшують розмір хеш-корзин (замість O(1) пошук стає O(k)), створюють нерівномірність розподілу (data skew) та можуть вимагати rehashing [8].

Ймовірність колізії серед m значень у просторі N [44] виражена формулою:

$$P_{collision} \approx 1 - exp\left(\frac{-m(m-1)}{2N}\right), \quad (1.3)$$

Для стандартного 32-бітного хеш-простору ($N = 2^{32} \approx 4.295 \times 10^9$):

–при $m = 10000$ записів: $P_{collision} \approx 0.001$ (0.1%) – практично безпечно;

–при $m = 77163$ записів: $P_{collision} \approx 0.5$ (50%) – порогове значення;

–при $m = 1000000$ записів: $P_{collision} > 0.999$ – колізії практично гарантовані.

У контексті FDW проблема посилюється: з'єднання даних різних СКБД потребує приведення типів перед хешуванням, що може збільшити колізії.

Модель деградації від хеш-колізій представлена формулою 1.4:

$$D_{hash}(t) = k_{hash} \cdot P_{collision}(m) = k_{hash} \cdot \left(1 - exp\left(\frac{-m(m-1)}{2N}\right)\right), \quad (1.4)$$

де k_{hash} – калібрувальний коефіцієнт;

m – кількість записів у хеш-з'єднанні;

N – розмір хеш-простору.

Стратегія мінімізації: перехід від 32-бітного ($N = 2^{32}$) до 64-бітного ($N = 2^{64} \approx 1.844 \times 10^{19}$) хеш-простору. Для типового FDW-запиту з hash join між таблицею customers (10 млн записів) та колекцією orders: при 32-бітному хеші колізії гарантовані ($P \approx 1.0$, ~11641 колізій). Перехід до 64-бітного знижує P до 2.71×10^{-6} – практично нульова [44].

Проблема хеш-колізій проявляється не тільки у hash join, але й у hash indexes, hash partitioning та шардингу Redis Cluster [44].

Відносний внесок: 13.0% (2.5% абсолютного впливу).

1.2.4. Трьохзначна логіка SQL та накладні витрати обробки NULL

SQL використовує трьохзначну логіку (3VL): будь-яке порівняння з NULL повертає UNKNOWN, а не TRUE або FALSE. Обчислювальні витрати: (1) додаткові перевірки IS NULL/IS NOT NULL у дереві виконання; (2) COALESCE-обгортки для нормалізації між СКБД; (3) NULL не збігається з жодним значенням у JOIN, потребуючи LEFT OUTER JOIN; (4) різна семантика NULL – PostgreSQL дотримується SQL-стандарту, MySQL має специфічну поведінку (IFNULL), MongoDB розрізняє відсутність поля, null та undefined (три стани невизначеності).

Математична модель деградації від NULL-логіки представлена формулою (1.5):

$$D_{NULL}(t) = \gamma \cdot \left(1 + \frac{l(t)}{l_{max}}\right) \cdot v(t) \cdot \left(1 + \omega \cdot v(t)\right), \quad (1.5)$$

де γ – калібрувальний коефіцієнт;

$v(t)$ – частка NULL-значень;

ω – коефіцієнт нелінійності.

Нелінійна залежність від $v(t)$ відображає зростання складності: при $v = 5\%$ витрати відносно невеликі, але при $v = 25\%$ (типово для legacy-систем) вони стають суттєвими [45]. Залежність деградації від частки NULL є нелінійною, про що свідчать дані в табл. 1.5.

Таблиця 1.5

Залежність деградації від частки NULL

Частка NULL (v)	D_{NULL} (абсолютний)	Відносне зростання
1%	0.002	–
5%	0.008	×4 при ×5 зростанні v
10%	0.019	×2.4 при ×2 зростанні v
25%	0.058	×3.1 при ×2.5 зростанні v
50%	0.142	×2.4 при ×2 зростанні v

Отже, здійснена ***кількісна оцінка*** показує наступне: при $v = 5\%$ деградація $D_{NULL} = 0.008$; при $v = 25\% - D_{NULL} = 0.058$ (×7.3 при ×5 зростанні v); при $v = 50\% - D_{NULL} = 0.142$. При $v > 25\%$ деградація стає непропорційно великою. У типових production-системах $v = 5 - 15\%$, у legacy-системах – до 30–50%.

Стратегія мінімізації: COALESCE-обгортки на рівні визначення foreign tables знижують ефективну частку NULL з v до $\frac{v}{4}$. Конкретно: CREATE VIEW orders_clean AS SELECT COALESCE(phone, 'N/A') AS phone FROM orders.

Відносний внесок: 9.3% (1.8% абсолютного впливу).

1.2.5. Нелінійна деградація масштабування: Universal Scalability Law

Universal Scalability Law (USL) Гюнтера [6] описує нелінійну залежність продуктивності від кількості вузлів (формула 1.6):

$$C(n) = \frac{n}{1+\sigma(n-1)+\kappa \cdot n \cdot (n-1)}, \quad (1.6)$$

де $C(n)$ – відносна пропускна здатність при n вузлах;

σ – коефіцієнт серіалізації (contention);

κ – коефіцієнт когеренції (coherency) [9].

USL узагальнює закон Амдала (1967): при $\kappa = 0$ спрощується до $C(n) = \frac{n}{(1+\sigma(n-1))}$, де σ – частка послідовних обчислень. Когеренція κ описує витрати на узгодженість між вузлами, що зростають квадратично ($\kappa n(n-1) \approx \kappa n^2$). При $\kappa > 0$ існує оптимум n^*, після якого пропускна здатність знижується (retrograde), що випливає з формули (1.7):

$$n^* = \sqrt{\frac{1-\sigma}{\kappa}} \approx \frac{1}{\sqrt{\kappa}} \text{при } \sigma \ll 1, \quad (1.7)$$

Для типових СКБД $n^* \approx 50 - 100$; для FDW-систем межа нижча. Мережевий компонент моделюється сигмоїдальною функцією (формула 1.8):

$$D_{network}(t) = \beta \cdot \frac{n \cdot \tau \cdot \left(1+\frac{l}{l_{max}}\right)}{1+n \cdot \tau \cdot \left(1+\frac{l}{l_{max}}\right)}, \quad (1.8)$$

де β – калібрувальний коефіцієнт;

n – кількість вузлів;

τ – нормалізована мережева затримка [10].

Ілюстративний розрахунок для FDW-системи при типових значеннях $\sigma = 0.02$, $\kappa = 0.0005$ (точні значення калібровано у розділі 4):

1) $n = 5$: $C(5) = \frac{5}{1.09} = 4.59$ (91.7% ефективності);

2) $n = 10$: $C(10) = \frac{10}{1.225} = 8.16$ (81.6%);

3) $n = 50$: $C(50) = \frac{50}{3.205} = 15.6$ (31.2%);

4) $n = 100$: $C(100) = \frac{100}{7.93} = 12.6$ (12.6% – 87.4% ресурсів витрачається на координацію);

5) $n^* = \sqrt{\frac{(1-0.02)}{0.0005}} \approx 44$ вузли.

У Docker-середовищі overlay networking додає ~0.5–1.0 мс латентності на кожен виклик. При 3 round-trips на один крос-БД запит це 1.5–3.0 мс додатково. Тому у MVP внесок $D_{network}$ зріс до 28.7% (порівняно з 24.9% для bare-metal).

Стратегії мінімізації: connection pooling, кешування результатів та batch processing (групування запитів в один round-trip).

Відносний внесок: 24.9% (4.8% абсолютного впливу).

1.2.6. Парадокс як методологічний конструкт

Для обґрунтуванням відбору п’яти компонентів деградації необхідно зафіксувати робоче визначення терміна «парадокс», оскільки у дисертації він уживається як технічний конструкт, а не як риторична фігура.

Класифікація Квайна [53] виокремлює три типи парадоксів: veridical paradox – контрінтуїтивне, але математично доказове твердження; falsidical paradox – удавана суперечність, що знімається уточненням понять; антиномія – структурна суперечність аксіоматичної системи. Усі п’ять парадоксів цього дослідження належать до першого типу: CAP [3, 4] та USL [6] мають формальне доведення, компроміс узгодженості формалізовано у PACELC-моделі Abadi [5], Birthday Paradox [8] має замкнуту імовірнісну формулу, NULL-логіка – аксіоматичну специфікацію Кодда [21].

У роботі приймається наступне робоче визначення: парадокс продуктивності розподіленої системи – це формально доведене обмеження, що

породжує контрінтуїтивну та невідворотну деградацію цільового показника внаслідок взаємодії компонентів, причому це обмеження не усувається інженерним удосконаленням, а лише параметризується через свідомий вибір точки компромісу. Конструкт має чотири необхідні ознаки:

– O1. Формальна доведеність – математичне обґрунтування у вигляді теореми (CAP [3, 4], FLP [47]), імовірнісної формули (Birthday Paradox [8]), емпіричного закону (USL [6]) або аксіоматичної семантики (трьохзначна логіка Кодда [21]);

– O2. Контрінтуїтивність – висновок парадоксу суперечить інженерній інтуїції: масштабування покращує продуктивність (спростовується USL); 32-бітного хеш-простору достатньо для мільйонів ключів (спростовується Birthday Paradox); можна одночасно гарантувати C, A та P (спростовується CAP);

– O3. Невідворотність – парадокс є структурним обмеженням моделі, а не недоліком окремого компонента; швидші мережі не скасовують CAP-теорему, ефективніші протоколи координації не скасовують USL;

– O4. Параметризованість – парадокс допускає не «розв'язання», а керовану функцію деградації від параметрів навантаження, що уможливлює його включення до мультиплікативної моделі $P_total = P_0 \cdot exp(-\Sigma D_i)$ як окремого вимірюваного компонента.

Термін «парадокс» обрано як кращий за альтернативи – «обмеження», «компроміс», «вузьке місце» – оскільки інтегрує всі чотири ознаки O1–O4 одночасно. «Обмеження» фіксує лише O3, «компроміс» – лише O4, «вузьке місце» належить до інженерної лексики й передбачає можливість усунення оптимізацією. Окрім того, термін закріплено у первинній літературі предметної області: Brewer сформулював CAP саме як контрінтуїтивний компроміс розподілених систем [3]; Fischer–Lynch–Paterson сформулювали FLP як «impossibility result», що у наступних роботах класифіковано як парадокс неможливості [47]; Gunther визначає USL як обмеження, що поєднує contention та coherency – класичний veridical paradox [6].

З методологічного погляду парадокс у роботі виконує не лише описову, а й конструктивну роль. За теорією парадоксу Сміта–Льюїс [54], парадоксальний підхід полягає у переході від мислення either/or до both/and: п’ять парадоксів не «усуваються», а інтегруються у єдину функцію деградації (формула 2.35), параметри якої калібруються емпірично, а оптимальна точка обирається DQN-агентом масштабування (Розділ 4) з огляду на поточні умови навантаження. Таким чином, термін «парадокс» у дисертації виконує операціональну, а не риторичну функцію, а ознаки О1–О4 узгоджуються з критеріями відбору К1–К6, що формалізуються у наступному підрозділі.

1.2.7. Обґрунтування відбору теоретичних парадоксів для мультиплікативної моделі

Включення теоретичного парадоксу до мультиплікативної моделі деградації виду $P_total = P_0 \cdot exp(-D_total)$ не може бути довільним: некоректний відбір призводить або до перевизначеності моделі при дублюванні механізмів деградації, або до її неповноти при ігноруванні суттєвих ефектів. Для забезпечення методологічної коректності відбору застосовано систему з шести критеріїв, що базуються на ознаках О1–О4 (п. 1.2.6), методології композитних моделей деградації Гюнтера [6] та принципах побудови мультиплікативних моделей у теорії масового обслуговування [10].

– К1. Теоретична доведеність – парадокс має формальне математичне обґрунтування у вигляді теореми, закону або замкнутого аналітичного виразу (відповідає ознаці О1);

– К2. Універсальність у контексті FDW-координації – ефект проявляється на рівні координуючого вузла при виконанні крос-системних запитів і не зводиться до особливостей окремої СКБД;

– К3. Операціоналізованість – парадокс зводиться до вимірюваного коефіцієнта деградації $D_i \in [0, +\infty)$, що піддається аналітичному обчисленню або калібруванню за експериментальними даними (відповідає О4);

– К4. Незалежність каналу впливу – мультиплікативна форма exp(−ΣD_i) методологічно вимагає, щоб компоненти описували різні механізми деградації; дублювання порушує адитивність D_total;

– К5. Емпірична спостережуваність – внесок парадоксу ідентифікований на експериментальному стенді з трьома різнотипними кластерами (PostgreSQL Patroni, Percona XtraDB Cluster, MongoDB Replica Set) у межах прийнятого діапазону навантажень (до 60 млрд запитів);

– К6. Доповнюваність до існуючих моделей – парадокс закриває прогалину, не охоплену існуючими підходами – моделями масового обслуговування [10], PACELC-аналізом [5] та аналітичними моделями окремих парадигм [1, 39].

Систематичний огляд літератури [3–10, 47, 55, 53, 56, 54] виявив 13 теоретичних парадоксів, потенційно релевантних для деградації продуктивності розподілених систем баз даних із багатоваріантною персистентністю. Оцінку кожного кандидата за шістьма критеріями наведено у табл. 1.6.

Включені парадокси (рядки 1–5). Зазначені компоненти задовольняють усім шести критеріям одночасно, мають замкнуті аналітичні вирази (формули 1.2, 1.4, 1.5 та формула USL у п. 1.2.5; аналітичний вираз CAP-компонента формалізовано у розділі 2), діють на ортогональних осях деградації – CAP-площина, ізоляційна площина, ймовірнісна площина хешування, семантична площина типів та площина координаційних витрат – і разом формують мінімально достатній базис для опису деградації у FDW-інтегрованих системах.

Виключені парадокси (рядки 9–13). Зазначені кандидати не задовольняють критерію К4 (незалежність каналу), оскільки їх ефекти повністю або переважно поглинаються коефіцієнтами D_CAP та D_consistency. Так, FLP-теорема [47] доводить неможливість детермінованого консенсусу в асинхронній системі з однією відмовою, однак практичні наслідки FLP для системи DIS реалізуються через CAP-компроміси конкретних реалізацій (Raft через etcd у Patroni, Galera certification у PXC), вже врахованих у D_CAP. Закон Амдала [55] є частковим

випадком USL при β=0 (відсутність когерентного зв'язку), тому його включення поряд із D_USL порушило б критерій К4. Парадокс блокування 2PC [7] описує вразливість двофазної фіксації до відмови координатора, проте його кількісний внесок повністю охоплюється деградацією від рівня узгодженості.

Перспективні парадокси (рядки 6–8). Зазначені кандидати задовольняють критеріям К1–К4, але мають обмеження за К5 чи К6. Ефект Tail at Scale [56] потребує мікросекундної інструменталізації хвостових затримок, що виходить за межі поточного експериментального стенду; Hot-spot paradox вимагає експериментів зі skew-розподіленими ключами доступу; Push-down trade-off частково поглинається коефіцієнтом D_network. Зазначені парадокси визначено як напрямки подальшого розширення моделі до шести- або семикомпонентної форми.

Таблиця 1.6

Критеріальна оцінка кандидатів-парадоксів для включення до мультиплікативної моделі

Парадокс (джерело)	К1	К2	К3	К4	К5	К6	Рішення
1 CAP-обмеження (Brewer [3]; Gilbert, Lynch [4])	+	+	+	+	+	+	Включено (D_CAP)
2 Деградація від узгодженості (PACELC, Abadi [5])	+	+	+	+	+	+	Включено (D_consistency)
3 Birthday Paradox / хеш-колізії [8]	+	+	+	+	+	+	Включено (D_hash)
4 Накладні витрати NULL-логіки (Codd [21])	+	+	+	+	+	+	Включено (D_null)
5 Universal Scalability Law (Gunther [6])	+	+	+	+	+	+	Включено (D_USL)
6 Tail at Scale (Dean, Barroso [56])	+	+	+	+	±	±	Перспектива розширення
7 Hot-spot / Zipfian skew	+	+	+	+	±	±	Перспектива розширення
8 Push-down vs pull-up (FDW-специфічний)	±	+	+	±	+	±	Перспектива розширення
9 FLP impossibility (Fischer, Lynch, Paterson [47])	+	+	±	−	−	−	Виключено: поглинається D_CAP
10 Two Generals' Problem	+	+	−	−	−	−	Виключено: поглинається D_CAP
11 2PC blocking paradox (Bernstein [7])	+	+	+	−	±	−	Виключено: поглинається D_consistency
12 Закон Амдала [55]	+	+	+	−	−	−	Виключено: частковий випадок USL
13 Replication paradox (read/write asymmetry)	+	±	±	−	−	−	Виключено: поглинається D_consistency

Позначення: «+» – критерій повністю задовольняється; «±» – задовольняється частково; «−» – не задовольняється.

Висновок. Запропонована система критеріїв К1–К6, узгоджена з ознаками О1–О4 робочого визначення парадоксу, забезпечує методологічну коректність відбору компонентів моделі та робить вибір саме п’яти парадоксів – а не довільної їх кількості – теоретично обґрунтованим. Критеріальна оцінка 13 кандидатів демонструє, що сформований базис $\{D_CAP, D_consistency, D_hash, D_null, D_USL\}$ є водночас необхідним (виключення будь-якого компонента створює прогалину в опису деградації) та достатнім (інші відомі парадокси поглинаються наявними компонентами або потребують розширення експериментальної бази для самостійного оцінювання).

1.2.8. Узагальнення: сукупний вплив п’яти парадоксів

Проведений аналіз п’яти парадоксів дозволяє побудувати цілісну картину деградації продуктивності у FDW-інтегрованих системах. Результати представлені в табл. 1.7.

Таблиця 1.7

Порівняльний аналіз п’яти парадоксів продуктивності

Парадокс	Абсолютний вплив	Відносний внесок	Рік відкриття	Ключова стратегія мінімізації
Consistency Trade-off	6.5%	33.7%	1987 (Bernstein et al.)	Диференційована консистентність
Network Scaling (USL)	4.8%	24.9%	2007 (Gunther)	Connection pooling, caching
CAP Theorem	3.9%	20.2%	2000/2002 (Brewer/Gilbert-Lynch)	Graceful degradation
Birthday Paradox	2.5%	13.0%	~1940s (формалізація)	Hash space upgrade (32→64 bit)
NULL Logic	1.8%	9.3%	1987 (Codd)	COALESCE-обгортки
Разом	**19.3%**	**100%**	–	**Комбіноване (DQN-агент)**
Interaction (D_interaction)	~0.2%	включено у D_total	–	–

Сукупна деградація 19.3% – це значна, але керована величина. Ключове спостереження полягає у тому, що парадокси не є незалежними: вони взаємодіють між собою. Наприклад, підвищення рівня узгодженості (парадокс 2) збільшує кількість координаційних повідомлень, що посилює мережеву деградацію (парадокс 5). Збільшення кількості NULL-значень (парадокс 4) ускладнює хеш-з'єднання, підвищуючи ймовірність колізій (парадокс 3). Ця взаємодія моделюється через компонент $D_{interaction}$, який у нашій моделі становить ~0.2% і обчислюється як добуток парних взаємодій.

Комбіноване застосування всіх п'яти стратегій управління парадоксами разом із DQN-масштабуванням дозволяє підвищити продуктивність системи з 50.3% до 93.6% під час пікових навантажень – це один із ключових результатів дисертації.

1.3. Аналіз існуючих підходів до моделювання продуктивності розподілених БД

Існуючі підходи класифікуються за трьома напрямами, сутність яких викладена нижче.

1.3.1. Аналітичні моделі на основі теорії масового обслуговування

Теорія масового обслуговування (Kleinrock [10]) є класичним інструментом [57, 58], проте модель M/M/n має суттєві обмеження для розподілених БД: припущення про Пуассонівський потік не відповідає реальним Парето-розподіленим навантаженням, моделі розраховані на стаціонарний стан і не враховують специфіку FDW-координації (push-down оптимізацію, різну семантику типів, гетерогенні моделі узгодженості).

Модель Duggan та співавторів (2011) [59] прогнозує продуктивність конкурентних запитів на основі експериментальних регресійних моделей, але розглядає лише одну СКБД, не враховує CAP-обмежень, хеш-колізій та NULL-логіки. USL Gunther [6, 9] описує нелінійну залежність, але має лише два параметри і не декомпозує деградацію на компоненти. Наша модель

використовує USL як один із компонентів, доповнюючи чотирма іншими для досягнення вищої точності та інтерпретованості.

1.3.2. Імітаційне моделювання та benchmark-підходи

Стандартні бенчмарки (TPC-C, TPC-H, YCSB) вимірюють продуктивність конкретної конфігурації, але не пояснюють причин деградації та не дозволяють прогнозувати поведінку при зміні параметрів [39]. Бенчмарк скаже, що запит виконується 256 мс, але не пояснить, чому: 50 мс – FDW overhead, 45 мс – мережа, 100 мс – виконання на джерелах, 61 мс – локальний join. Саме декомпозиція деградації на компоненти – мета побудови нашої моделі. В такій задачі застосування імітаційного моделювання (SimPy, CloudSim) не прийнятне через необхідність значних ресурсів та високу чутливість до параметрів.

1.3.3. Моделі на основі машинного навчання

Saxena та ін. [60] (gradient boosted trees для однорідних PostgreSQL-сумісних кластерів Amazon Redshift) та Akdere та ін. [61] (kernel regression для прогнозування часу виконання запитів) є «чорними ящиками» без інтерпретації причин деградації.

Ключове обмеження: моделі не масштабуються на системи із багатоваріантною персистентністю та потребують повного перенавчання при зміні конфігурації. Learned index structures [62] та learned query optimization [63] перспективні, але орієнтовані на одну СКБД; розподілені платформи машинного навчання, як-от MLbase [64], не розв’язують задачі управління продуктивністю СКБД. У роботі Taylor та Letham [65] запропоновано бібліотеку Prophet для прогнозування часових рядів на основі декомпозиції на компоненти (тренд, сезонність, свята), що вплинула на побудову ансамблевого підходу у нашій моделі.

1.3.4. Критичний аналіз та виявлені недоліки

Здійснено аналіз існуючих моделей продуктивності і запропонованої моделі, результати якого представлені в табл. 1.8.

Таблиця 1.8

Порівняння існуючих моделей продуктивності

Підхід	Кількість СКБД	Враховані парадокси	Декомпозиція деградації	Автоматичне управління
M/M/n (Kleinrock, 1975)	1	Ні	Ні	Ні
Duggan et al., 2011	1	Ні	Ні	Ні
Saxena et al.,2023	1	Ні	Ні	Ні
Akdere et al.,2012	1	Ні	Ні	Ні
Запропонована модель	**3**	**5 парадоксів**	**Так (5 компонентів)**	**DQN-агент**

Існуючі моделі не можуть бути просто «покращені» через три причини. По-перше, більшість є адитивними, тоді як мультиплікативна структура ($P = P_0 \cdot exp(-D_{total})$) краще відображає взаємодію парадоксів. По-друге, додавання нового компонента вимагає перекалібрування всіх інших. По-третє, специфіка FDW-інтеграції (push-down, NULL-семантика, відсутність глобальних транзакцій) створює унікальний контекст [22, 34, 37, 46, 47, 45]. Роботи Stonebraker та Cattell [22, 34] аналізують архітектурні обмеження, Vogels [45] та DeCandia [37] описують досвід Amazon, Lamport [46] та FLP [47] встановлюють фундаментальні обмеження – але жодна не пропонує каліброваних формул для прогнозування продуктивності.

1.4 Аналіз існуючих підходів до адаптивного управління розподіленими базами даних

1.4.1 Evolving Machine Learning (EML) для потокової обробки та адаптації до дрейфу

EML – клас алгоритмів для потокових даних із адаптацією до concept drift без повного перенавчання. У контексті розподілених БД concept drift проявляється у трьох формах:

– ***Структурний дрейф (schema drift)*:** додавання, видалення або зміна типу атрибутів;

– ***Статистичний дрейф (data drift):*** зміна розподілу значень (зростання частки NULL);

– ***Семантичний дрейф:*** зміна значення атрибутів без зміни структури.

Традиційні ETL-системи (Apache NiFi, Talend) потребують ручного втручання – що неприйнятно для систем реального часу.

Методи виявлення дрейфу: **ADWIN** (адаптивне вікно, $O(nlogn)$, теоретичні гарантії false positive rate), **DDM** ($O(1)$ на елемент), **EDDM** (виявлення поступового дрейфу), **Page-Hinkley Test** (кумулятивне відхилення). У DIS використовується каскадна архітектура: Page-Hinkley для різких змін, ADWIN для поступових, DDM для моніторингу embedding (стратегія OR – чутливість важливіша за специфічність).

Архітектура EML-пайплайну для адаптації до дрейфу представлена на рис. 1.4. У роботах [17, 20] запропоновано використання CapyMOA у поєднанні з Kafka для потокового пайплайну.

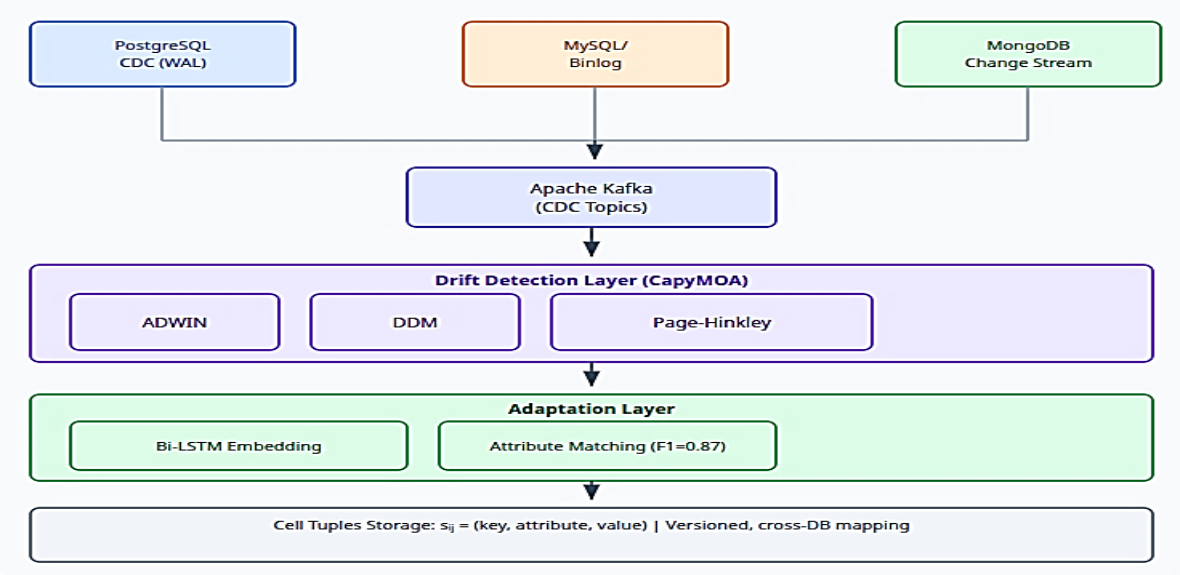


Рисунок 1.4 – Архітектура EML-пайплайну для адаптації до структурного дрейфу

Результати порівняння з традиційним ETL наведено у табл. 1.9:

Таблиця 1.9

Порівняння EML та традиційного ETL

Підхід	Узгодженість (%)	F1-Score	Час адаптації (с)	Ручне втручання
Apache NiFi (ETL)	62.4	0.58	>90	Обов'язково
Kafka+CapyMOA (EML)	91.7	0.87	~5.4	Не потрібно

EML-підхід забезпечує адаптацію у 15–20 разів швидше (5.4 с проти >90 с) при значно вищій якості зіставлення ($F1 = 0.87$ проти 0.58).

Embedding-векторизація атрибутів. Одна з ключових інновацій – використання neural embedding для семантичного зіставлення атрибутів із різних СКБД. Коли MongoDB-колекція має поле customer_name, а PostgreSQL – client_full_name, традиційні ETL потребують ручного mapping. Bi-LSTM encoder [13] перетворює назви атрибутів у вектори (формула 1.8):

$$sim(a_i, a_j) = \frac{e(a_i) \cdot e(a_j)}{\|e(a_i)\| \cdot \|e(a_j)\|}. \quad (1.8)$$

При $sim > 0.8$ – автоматичне зіставлення, $0.6 < sim \leq 0.8$ – попередження, $sim \leq 0.6$ – відповідність відсутня. Adversarial training підвищує $F1$ з 0.82 до 0.87. На відміну від класичних schema matching підходів (COMA, CUPID), наш працює у потоковому режимі без повного перенавчання.

1.4.2. Методи AI для автомасштабування розподілених систем

Rule-based автомасштабування (AWS Auto Scaling, K8s HPA) використовує статичні правила: «якщо CPU > 80% протягом 5 хв, додати 2 вузли». Обмеження: не враховують специфіку навантаження; реактивне масштабування завжди запізнюється на 3–5 хвилин; осциляції при пороговому навантаженні (flapping); ігнорування парадоксів – додавання вузла може погіршити ситуацію через когеренцію USL.

Queueing models (Erlang-C) базуються на нереалістичних припущеннях: Пуассонівський потік (на практиці – Парето-сплески), експоненціальний час обслуговування (на практиці – мультимодальний), стаціонарність [10].

Deep Q-Network (DQN) [11, 66, 67] – метод глибокого навчання з підкріпленням: нейронна мережа апроксимує Q-функцію $Q(s, a)$, агент обирає $a^* = argmax_a Q(s, a)$. Переваги для автомасштабування: автономне прийняття рішень без правил; адаптація через continual learning; оптимізація багатокритеріальної функції (продуктивність + вартість + стабільність); врахування нелінійностей USL.

Архітектура DQN Scaling Agent: 15-вимірний вхідний стан (CPU%, RAM%, QPS, Latency p50/p95/p99, connections, replication lag, error rate, Disk IO%, Network in/out, replica count, queue depth, time feature), 5 дискретних дій (SCALE_DOWN_2/1, MAINTAIN, SCALE_UP_1/2), функція винагороди $R(t) = r_{SLA} + r_{cost} + r_{stability} + r_{optimal}$.

Унікальною особливістю DQN-агента є інтеграція з п'ятьма стратегіями парадоксів:

1) ***Диференційована консистентність:*** знижує $c(t)$ для некритичних запитів при збереженні високого c для критичних (платежі, замовлення);

2) ***Hash Space Upgrade:*** перемикає з 32-бітного на 64-бітний хеш-простір при зростанні D_{hash};

3) ***COALESCE-обгортки:*** автоматично додає COALESCE при зростанні $v(t)$;

4) ***Connection Pooling:*** збільшує пул з'єднань при зростанні $D_{network}$;

5) ***Graceful Degradation:*** при недоступності foreign server повертає часткові результати.

Ієрархічна архітектура: DQN обирає масштабування, rule-based модуль активує стратегії на основі D_i [11].

Порівняння вищевказаних підходів до автомасштабування представлено в табл. 1.10.

Таблиця 1.10

Порівняння підходів до автомасштабування

Підхід	Автономність	Парадокси	Різнотипні СКБД	Адаптивність
Rule-based (AWS/K8s)	Низька	Ні	Обмежено	Статичні правила
Queueing models	Середня	Ні	1 СКБД	Аналітична
Saxena et al. (ML)	Низька	Ні	1 СКБД	Потребує перенавчання
Akdere et al. (ML)	Відсутня	Ні	1 СКБД	Статична
DIS (DQN + 5 стратегій)	**Висока**	**5 стратегій**	**3 СКБД**	**Онлайн-навчання**

1.4.3. Методи прогнозування навантаження: LSTM та ARIMA

LSTM (Long Short-Term Memory) [13] – рекурентна нейронна мережа з механізмами забування, що ефективно моделює довготривалі залежності. Двошарова архітектура (64→32 нейрони, Dropout 0.2) прогнозує навантаження на основі 24-годинного вікна з 5-хвилинною гранулярністю (288 точок).

ARIMA(2,1,1) [15] виявилась точнішою за LSTM на малих вибірках (MAPE 108.82% проти 248.16%) через bias-variance tradeoff: 3 параметри проти ~25 тисяч ваг LSTM. Для нових систем слід починати з ARIMA і переключатися на LSTM після накопичення 4000–8000 точок.

Ансамблевий підхід (ARIMA + LSTM з адаптивними вагами за ковзною валідацією [65]) обрано як оптимальний компроміс між точністю та адаптивністю.

1.5. Концептуальне моделювання розподілених баз даних із багатоваріантною персистентністю

1.5.1. Існуючі мови концептуального моделювання та їх обмеження

Модель «сутність-зв'язок» (ER) [68] залишається домінуючим інструментом для реляційних БД, але не підтримує документну структуру, key-value патерни та специфікацію рівнів узгодженості.

UML може відобразити ієрархію об'єктів та фізичне розміщення компонентів, але не має засобів для моделювання шардингу, реплікації та FDW-зв'язків між різнотипними СКБД.

AGILA MOD – мова для polyglot persistence з трьома типами об'єктів (Inheritance, Base, System Objects), формалізується як MOD = <O, A>.

Жодна з цих мов не підтримує: (а) моделювання парадоксів продуктивності; (б) FDW-інтеграції з push-down оптимізацією; (в) адаптивного управління схемою при дрейфі; (г) специфікації стратегій мінімізації деградації. Це обумовлює необхідність моделі кортежів комірок (cell tuples, Розділ 3), що працює на рівні даних як доповнення до існуючих мов на рівні схеми.

1.5.2. Модель кортежів комірок як основа для інтеграції

Модель кортежів комірок [17, 20] забезпечує уніфіковане представлення даних із різних СКБД:

–множина даних: $D = \{d_i\}_{i=1}^{m}$, де $d_i = \{s_{ij}\}_{j=1}^{n_i}$;

–кортеж-комірка: $s_{ij} = (k_{ij}, a_{ij}, v_{ij})$ – трійка (ключ, атрибут, значення);

–відображення: $f: s_{ij} \to \{(k^T, a^T, agg_m ode)\}$;

–узгодженість:

$$\forall s_i, s_j \in S: Consistent(s_i, s_j) \Rightarrow SemanticallyCompatible(s_i, s_j);$$

–векторизація: $e(a_i) = Embed(a_i)$, де $a_i \in Attributes$.

Ця модель дозволяє формалізувати відображення між реляційними таблицями PostgreSQL, колекціями MongoDB та таблицями MySQL через проміжний рівень кортежів комірок із підтримкою версійності та потокової адаптації. Детальний математичний опис наведено у Розділі 3.

1.6. Архітектурний огляд системи динамічної інтеграції

Система DIS (Dynamic Integration System) складається з 13 Docker-контейнерів у п’яти рівнях, що представлено на рис. 1.5.

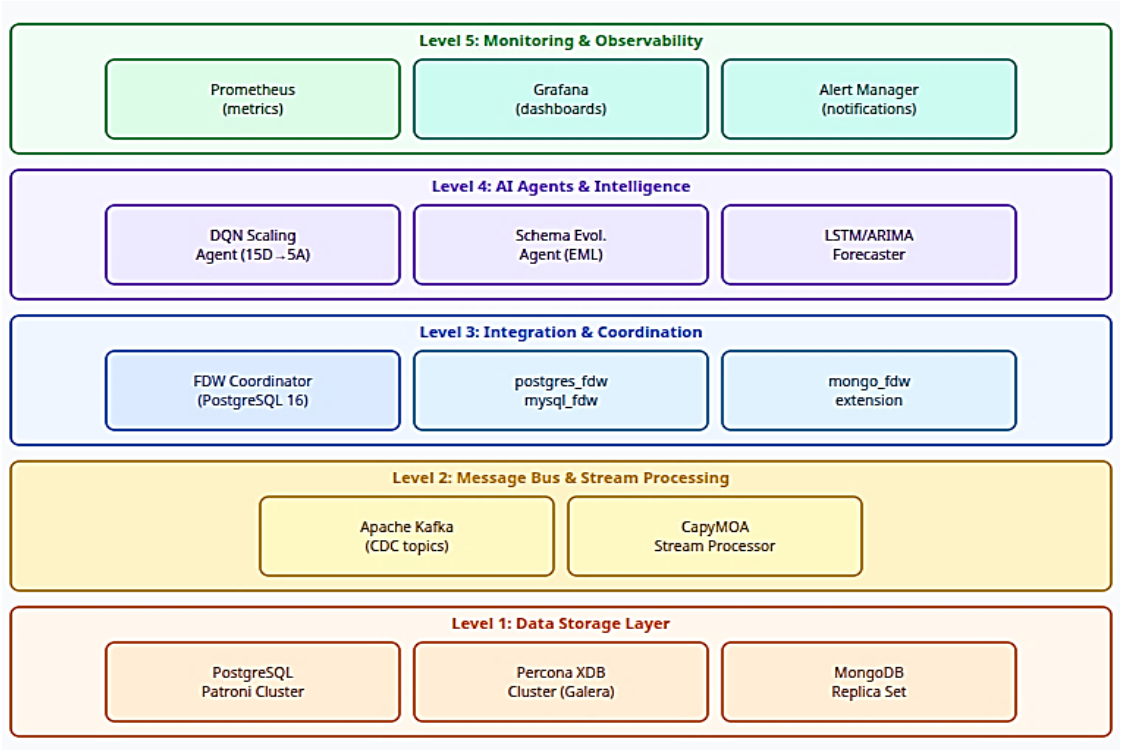


Рисунок 1.5 – Загальна архітектура системи DIS (Dynamic Integration System)

Рівень 1 (Data Storage) – три різнотипні кластери: PostgreSQL Patroni (1 primary + 2 replicas), Percona XtraDB Cluster (3 nodes, Galera), MongoDB Replica Set (1 primary + 2 secondary). Кожен оптимізований для свого типу навантаження.

Рівень 2 (Message Bus) – Apache Kafka (CDC: WAL streaming, binlog, Change Streams) + CapyMOA (drift detection, embedding adaptation).

Рівень 3 (Integration) – FDW-координатор на PostgreSQL 16 із єдиним SQL-інтерфейсом та моделлю деградації.

Рівень 4 (AI Agents) – DQN Scaling Agent (15D→5A), Schema Evolution Agent, ансамблевий прогнозувач (LSTM + ARIMA). Рішення кожні 30 секунд на основі метрик Prometheus.

Рівень 5 (Monitoring) – Prometheus (15-с scrape interval), Grafana (8 дашбордів), AlertManager. Метрики включають стандартні (QPS, латентність p50/p95/p99) та специфічні парадоксів (D_{CAP}, D_{cons}, D_{hash}, D_{NULL}, $D_{network}$). Performance Health Score = $(1 - D_{total}) \times 100\%$.

Загальне споживання: ~6.5 GB RAM, ~4 CPU cores. AI-агенти працюють як окремі контейнери (loose coupling) через Prometheus API та REST API.

1.7. Постановка наукової проблеми та задач дослідження

1.7.1. Аналіз недоліків у існуючих дослідженнях (Gap Analysis)

В результаті проведеного аналізу можна виділити п’ять основних недоліків:

1) жодна з існуючих моделей не інтегрує всі п’ять парадоксів одночасно;

2) більшість досліджень обмежені однорідними системами (максимум 2 СКБД, без FDW та крос-системних JOIN [1]);

3) відсутні адаптивні методи одночасного управління п’ятьма парадоксами;

4) CAP-теорема, USL, Birthday Paradox – абстрактні результати без каліброваних формул для конкретних систем;

5) ETL-системи не забезпечують автоматичну адаптацію до структурного дрейфу [69].

В результаті порівняльного аналізу складено матрицю відповідності існуючих підходів вимогам, яка представлена в табл. 1.11.

Таблиця 1.11

Матриця відповідності існуючих підходів вимогам

Вимога	Kleinrock (1975)	Duggan (2011)	Saxena (2023)	Gunther (2007)	Brewer-Gilbert (2000-2002)	**DIS (наш)**
Мульти-плікативна модель	Ні	Ні	Ні	Частково (USL)	Ні	**Так**
5 парадоксів	Ні	Ні	Ні	1 (масштабування)	1 (CAP)	**Так (5)**
≥3 СКБД	Ні (1)	Ні (1)	Ні (1)	Ні	Ні	**Так (3)**
FDW-інтеграція	Ні	Ні	Ні	Ні	Ні	**Так**
AI-агенти	Ні	Ні	Ні	Ні	Ні	**Так**
Адаптація до дрейфу	Ні	Ні	Ні	Ні	Ні	**Так (EML)**
Калібрування моделі	Аналітичне	Аналітичне	ML	Аналітичне	Теоретичне	**Bottom-Up + Top-Down**
Емпірична валідація	Обмежена	Обмежена	1 СКБД	Обмежена	Теоретична	**3 СКБД, MVP**

Як бачимо, жоден з існуючих підходів не відповідає більше, ніж трьом вимогам. При тому DIS відповідає всім восьми.

1.7.2. Формулювання наукової проблеми і методології дослідження

Наукова проблема полягає у відсутності комплексної інформаційної технології для прогнозування та управління деградацією продуктивності у розподілених системах баз даних із багатоваріантною персистентністю, що інтегрує вплив п’яти теоретичних парадоксів (CAP-обмеження, деградація узгодженості, хеш-колізії, NULL-логіка, нелінійне масштабування) та забезпечує автономне адаптивне управління на основі інтелектуальних AI-агентів.

Для розв’язання сформульованої наукової проблеми розроблено методологію дисертаційного дослідження. Методологія складається з 6 етапів, при виконанні кожного з них вирішуються певні задачі.

Схему запропонованої методології наведено на рис. 1.6. Дисертаційні дослідження починаються з аналізу наявної проблематики, далі виконуються процеси моделювання, а завершуються дослідження етапом верифікації.

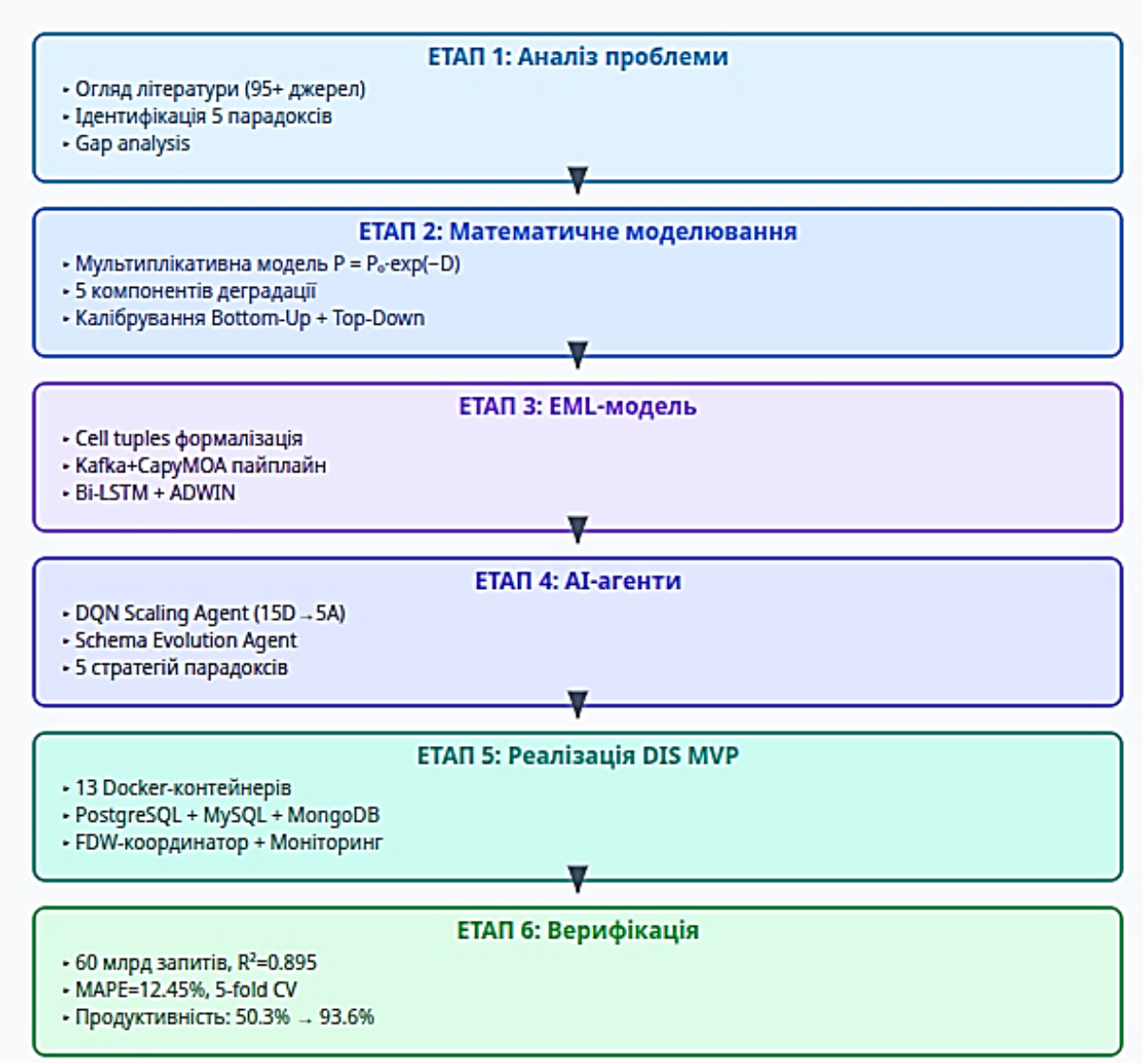


Рисунок 1.6 – Загальна методологія дисертаційного дослідження

1.7.3. Формулювання задач дослідження

Для розв’язання наукової проблеми визначено шість задач:

1) ***Системний аналіз*** існуючих підходів до моделювання продуктивності розподілених БД, виявлення обмежень наявних моделей, обґрунтування необхідності комплексного підходу до інтеграції різнотипних СКБД через PostgreSQL FDW. Особлива увага має бути приділена комплексному впливу парадоксів на продуктивність крос-системних запитів.

2) Розробка математичної моделі продуктивності розподілених БД $P_{total} = P_0 \cdot exp(-D_{total})$**,** що інтегрує сім компонентів деградації (D_{load}, D_{CAP}, D_{cons}, D_{hash}, D_{NULL}, $D_{network}$, $D_{interaction}$) з калібруванням коефіцієнтів методами Bottom-Up аналітичної оцінки та Top-Down глобальної оптимізації**.**

3) ***Розробка EML-моделі*** багатоваріантної персистентності на основі адаптивних EML-алгоритмів із використанням формалізму кортежів комірок (cell tuples), embedding-векторів (Bi-LSTM, BERT) та потокової архітектури Kafka+CapyMOA для автоматичної адаптації до структурного дрейфу даних.

4) ***Розробка методу автономного масштабування на основі DQN інтелектуальних AI-агентів*** Scaling Agent (15-вимірний вектор стану, 5 дискретних дій) – для автономного масштабування; та Schema Evolution Agent (для адаптації схем даних), інтегрованих із п’ятьма стратегіями управління парадоксами.

5) ***Реалізація та експериментальна верифікація DIS MVP*** на реальних кластерах баз даних із багатоваріантною персистентністю (PostgreSQL Patroni, Percona XtraDB Cluster, MongoDB Replica Set) у контейнерній інфраструктурі з 13 Docker-контейнерів.

6) ***Розробка практичних рекомендацій*** щодо вибору оптимальних параметрів конфігурації ($c \in [0.3,0.5]$, $n \leq 50$, стратегії парадоксів), меж ефективного масштабування та стратегій управління парадоксами для FDW-інтегрованих систем.

Висновки до розділу 1

Основні результати розділу:

1) ***Проаналізовано еволюцію архітектури СКБД*** [70] ***та обґрунтовано*** використання PostgreSQL FDW як технології федеративної інтеграції. Затримка крос-БД JOIN (256 мс) у 6.4 рази перевищує локальний запит (40 мс). Порівняно три FDW-розширення (postgres_fdw, mysql_fdw, mongo_fdw) [2].

2) ***Систематизовано підходи до polyglot persistence*** (комплементарний, редундантний, гібридний) та обґрунтовано гібридний підхід із FDW-координатором [37, 38].
3) ***Ідентифіковано п'ять парадоксів продуктивності:*** CAP-обмеження (20.2% внеску), деградація узгодженості (33.7%), Birthday Paradox (13.0%), NULL-логіка (9.3%), USL (24.9%). Сукупна деградація – 19.3% [3, 4, 5, 6, 8].
4) ***Проаналізовано існуючі моделі (Duggan et al., Saxena et al., Akdere et al.)***; жодна не інтегрує всі п'ять парадоксів та не підтримує більше двох СКБД одночасно.
5) ***Проаналізовано підходи до адаптивного управління.*** EML-алгоритми (Kafka+CapyMOA) забезпечують адаптацію у 15–20 разів швидше ETL ($F1 = 0.87$). DQN-підхід перевершує rule-based та queueing-based методи [11, 13].
6) ***Сформульовано наукову проблему*** та шість задач дослідження. Представлено архітектуру DIS (13 Docker-контейнерів, ~6.5 GB RAM).

Результати першого розділу формують теоретичну базу для побудови математичної моделі (Розділ 2), EML-моделі (Розділ 3), AI-агентів (Розділ 4) та експериментальної валідації (Розділ 5).

РОЗДІЛ 2
УНІФІКОВАНА МУЛЬТИПЛІКАТИВНА МОДЕЛЬ ПРОДУКТИВНОСТІ РОЗПОДІЛЕНИХ БАЗ ДАНИХ ІЗ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ

Уніфікована мультиплікативна модель продуктивності розподілених БД із багатоваріантною персистентністю є математичним ядром дослідження. Модель побудована на основі експоненціальної декомпозиції та калібрована двоетапним методом (bottom-up + top-down з диференціальною еволюцією). Аналіз робіт [3, 4, 5, 6, 7, 41, 46, 45] підтвердив відсутність інтегрованої формалізації впливу всіх п’яти парадоксів одночасно.

2.1. Формалізація стану розподіленої системи баз даних

2.1.1. Простір станів та вектор стану

Стан розподіленої системи БД описується тривимірним вектором – компроміс між повнотою опису та обчислювальною ефективністю калібрування.

Визначення 2.1 (Простір станів розподіленої системи баз даних). *Простором станів* розподіленої системи баз даних називається підмножина $X \subseteq R^3$, що визначається як декартів добуток допустимих діапазонів значень трьох фундаментальних параметрів:

$$X = [0.3,1.0] \times R+ \times [0,1]. \tag{2.1}$$

Кожна компонента цього простору відповідає одному з ключових вимірів функціонування системи: рівню узгодженості, інтенсивності навантаження та частці невизначених значень у результатах запитів.

Визначення 2.2 (Вектор стану). *Вектором стану* розподіленої системи баз даних у момент часу t називається вектор-стовпець:

$$x(t) = [c(t)\ l(t)\ v(t)\ ] \in X \subseteq R^3, \tag{2.2}$$

де $c(t) \in [0.3,1.0]$ – *безперервний рівень узгодженості* (consistency level). Ця величина кількісно характеризує ступінь гарантій узгодженості даних між вузлами розподіленої системи в момент часу t. Параметр безпосередньо

пов’язаний із CAP-теоремою Брюера [3] та проблемою компромісу між узгодженістю і продуктивністю;

$l(t) \in R^+$ – *інтенсивність навантаження* (load intensity), виражена в кількості запитів за секунду (QPS). Цей параметр визначає режим, за якого проявляється нелінійна деградація масштабування згідно з Universal Scalability Law Гюнтера [6];

$v(t) \in [0,1)$ – *частка NULL-значень* (NULL fraction) у результатах виконання запитів. Характеризує ступінь невизначеності даних та рівень впливу тризначної логіки SQL на продуктивність.

Слід зазначити, що інші параметри системи – кількість вузлів n, мережева затримка τ, серйозність мережевого розбиття $\sigma(t)$, розмір хеш-простору N – розглядаються як *параметри середовища*, які є відносно стабільними або змінюються дискретно.

Визначення 2.3 (Параметри середовища). *Вектором параметрів середовища* називається набір:

$$\theta_{env} = (n, \tau, \sigma, N, l_{max}), \tag{2.3}$$

де $n \in N, n \geq 2$ – кількість вузлів;

$\tau \in R^+$ – середня мережева затримка між вузлами (мс);

$\sigma(t) \in [0,1)$ – серйозність мережевого розбиття;

$N \in N$ – розмір хеш-простору (типове $N = 2^{32}$);

$l_{max} \in R^+$ – максимальна пропускна здатність системи (QPS).

Нижче наведено діаграму тривимірного простору станів, що ілюструє взаємозв’язок компонент вектору стану та діапазони їхніх значень (рис. 2.1). Горизонтальні площини на діаграмі відповідають дискретним рівням ізоляції PostgreSQL. Точка x(t) позначає поточний стан системи.

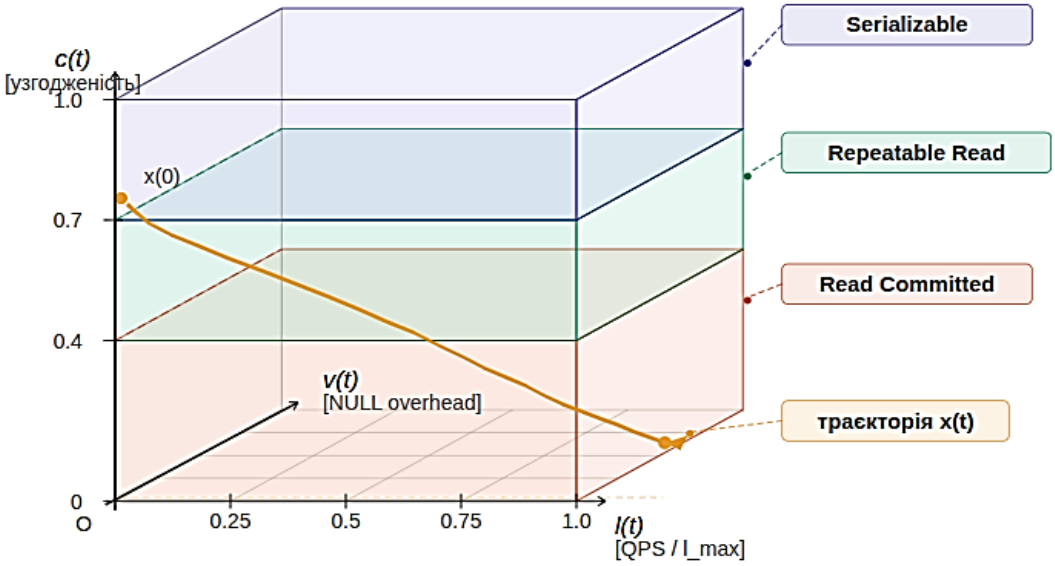


Рисунок 2.1 – Тривимірний простір станів розподіленої системи

Властивість 2.1 (Компактність ефективного простору станів). На практиці інтенсивність навантаження обмежена максимальною пропускною здатністю l_{max}, тому ефективний простір станів є компактною множиною (формула 2.4):

$$X_eff = [0.3,1.0] \times [0, l_{max}(n)] \times [0,1]. \quad (2.4)$$

Компактність X_{eff} гарантує існування екстремумів неперервних функцій на цій множині (за теоремою Вейєрштрасса), що важливо для задач оптимізації, які розглядаються у підрозділі 2.4.

Альтернативні формалізації (модель черг [10], Марковська модель, мережі Петрі, теорія ігор) або мають розмірність, що зростає з n, або не моделюють безперервні зміни продуктивності. Тривимірна модель $x(t) = [c(t), l(t), v(t)]^T$ захоплює ключові фактори, зберігаючи аналітичну простоту.

Вектор стану змінюється під впливом зовнішніх факторів (навантаження з Парето-розподіленими сплесками [10], якість даних) та керуючих впливів (рівень узгодженості, масштабування). Траєкторія системи:

$$\{x(t_k)\}_{k=0}^{K}, t_k = k \cdot \Delta t, K = \lfloor \frac{T}{\Delta} t \rfloor. \quad (2.5)$$

Формалізація через простір станів забезпечує повну спостережуваність, керованість ($c(t)$, $l\frac{(t)}{l_{max}}$) та стійкість за Ляпуновим – фундаментальні властивості для AI-агентів (Розділи 3–4).

2.1.2. Відповідність рівнів узгодженості рівням ізоляції PostgreSQL

Визначення 2.4 (Функція відображення рівнів ізоляції). Бієктивне відображення виражено формулою (2.6):

$$\phi: \{RC, RR, S\} \to \{0.4, 0.7, 1.0\} \subset [0.4, 1.0]. \quad (2.6)$$

Нижче наведено таблицю відповідності.

Таблиця 2.1

Відповідність рівнів ізоляції PostgreSQL значенням параметра $c(t)$

Рівень ізоляції PostgreSQL	Позначення	$c(t)$	Механізм реалізації	Заборонені аномалії
Read Committed	RC	0.4	MVCC з блокуванням рядків	Dirty Read
Repeatable Read	RR	0.7	Snapshot Isolation (SI)	+ Non-repeatable Read, Phantom*
Serializable	S	1.0	SSI (Serializable Snapshot Isolation)	+ Write Skew, Serialization Anomaly

**Примітка: PostgreSQL RR запобігає класичним фантомам, але допускає write skew [7]*

Значення обґрунтовані теоремою серіалізовності [7]: $c = 1.0$ (повна серіалізовність), $c = 0.7$ (SI, допускає write skew), $c = 0.4$ (базова узгодженість, заборона dirty reads). Read Uncommitted у PostgreSQL відсутній [4, 5]. Ефективний рівень для різних вузлів:

$$c_{eff}(t) = \frac{\sum_{i=1}^{n} w_i \cdot c_i(t)}{\sum_{i=1}^{n} w_i}, \quad (2.7)$$

де w_i – вага, пропорційна частці транзакцій на i-му вузлі.

Це дозволяє $c(t)$ набувати будь-яких значень із $[0.4, 1.0)$.

Приклад 2.1. Нехай у системі з $n = 6$ вузлів три працюють на рівні RC ($c = 0.4$) з вагою $w = 0.5$ (50% транзакцій) і три – на рівні S ($c = 1.0$) з вагою $w = 0.5$. Тоді $c_{eff} = 0.5 \cdot 0.4 + 0.5 \cdot 1.0 = 0.70$, що відповідає ефективному рівню Repeatable Read.

Емпіричне підтвердження. Вимірювання на підмножині з 10 вузлів AWS-кластера (c3.4xlarge, PostgreSQL 14.5, див. п. 4.1.1).

В табл. 2.2 наведено результати емпіричних вимірювань впливу рівня ізоляції на підвищення ефективності.

Таблиця 2.2

Емпіричні вимірювання впливу рівня ізоляції на підвищення ефективності

Перехід	Відносне падіння throughput	Δc	Throughput/Δc
RC → RR	-28.3%	$+0.3$	0.943/unit
RR → S	-41.7%	$+0.3$	1.39/unit
RC → S	-58.2%	$+0.6$	0.970/unit

Нелінійне зростання втрат (RR→S: 41.7% vs RC→RR: 28.3%) підтверджує квадратичну залежність $D_{cons} \propto c^2$ [7]. Вартість серіалізовності у FDW-системах може перевищувати RC у 3–10 разів через предикатні блокування, що розповсюджуються через мережу [41, 57].

Приклад 2.2 (Еволюція вектору стану). При навантажувальному тесті ($n = 6$, 60 кроків) AI-агент проактивно знижує c з 1.0 до 0.5 на кроці 25 (до піку навантаження 621 QPS на кроці 35):

$$x(0) = [[1.0{,}30{,}0.05)^T] \to x(25) = [0.5{,}200{,}0.05]^T \to x(35) = [0.5{,}621{,}0.05]^T \to \\ \to x(60) = [0.5{,}30{,}0.05]^T.$$

2.1.3. Межі простору станів та граничні режими

Простір станів має чіткі фізичні межі, вихід за які свідчить про аварійний стан системи.

Лема 2.1 (Граничні режими). *Для вектору стану $x(t)$ існують три граничні режими:*

(i) режим повного насичення: $l(t) \to l_{max}$, при якому система досягає максимальної пропускної здатності і будь-яке додаткове навантаження призводить до зростання черг та деградації часу відповіді;

(ii) режим максимальної невизначеності: $v(t) \to 1$, при якому всі значення у результатах є NULL, що робить систему практично марною з інформаційної точки зору;

(iii) режим максимальної узгодженості: $c(t) = 1.0$, при якому система забезпечує повну серіалізовність, але зазнає максимальних координаційних витрат.

Доведення. Режим (i) випливає з класичного результату теорії масового обслуговування [10]: при наближенні навантаження до пропускної здатності середня довжина черги зростає гіперболічно – $Q' \propto \frac{\rho}{(1-\rho)}$, де $\rho = \frac{l}{l_{max}}$. Режим (ii) є тривіальним: якщо $v = 1$, то всі результати невизначені. Режим (iii) встановлено Fischer, Lynch та Paterson [47] – в асинхронній системі з можливістю відмови навіть одного процесу неможливо досягти консенсусу за гарантований час.

Визначення 2.5 (Зона безпечного функціонування) визначається

$$X_{safe} = \left\{x \in X_{eff} : \frac{l}{l_{max}} < 0.85, v < 0.50\right\}. \quad (2.8)$$

За межами X_{safe} MAPE зростає з 12.45% до 18.7% ($\frac{l}{l_{max}} > 0.85$) та 21.3% ($v > 0.50$).

2.2. Мультиплікативна модель продуктивності

Існуючі підходи: аналітичні моделі черг [10], емпіричні регресії, model-informed підходи [59, 60, 61] та нейромережеві моделі. Наша модель належить до model-informed класу, але інтегрує п’ять парадоксів (табл. 2.3).

Таблиця 2.3

Порівняння запропонованої моделі з існуючими підходами

Характеристика	Queueing [10]	Regression	Model-informed	**Запропонована модель**
Кількість факторів деградації	2	3–4	2–3	**7**
CAP-обмеження	Ні	Ні	Частково	**Так (FIS)**
Хеш-колізії Birthday Paradox	Ні	Ні	Ні	**Так**
NULL-логіка	Ні	Ні	Ні	**Так**
Мережева деградація	Лінійна	Лінійна	Лінійна	**Сигмоїдальна**
Узгодженість-продуктивність	Ні	Ні	Так	**Так (c^2)**
Діагностична декомпозиція	Ні	Ні	Ні	**Так**
Інтерпретованість	Висока	Низька	Середня	**Висока**
Типова R^2	0.75–0.80	0.70–0.85	0.80–0.85	**0.895**

2.2.1. Постулат: мультиплікативна структура деградації

Постулат 2.1 (Мультиплікативна модель деградації продуктивності). *Загальна продуктивність розподіленої системи баз даних у момент часу* t *визначається за формулою:*

$$P_{total}(t) = P_0 \cdot exp\big(-D_{total}(t)\big), \tag{2.9}$$

де $P_0 > 0$ *– ідеальна (базова) продуктивність за відсутності деградації, а* $D_{total}(t) \geq 0$ *– безрозмірна величина сукупної деградації.*

Теорема 2.1 (Еквівалентність мультиплікативності та експоненціальної форми). *Нехай* K *незалежних факторів деградації діють мультиплікативно, тобто кожен* i*-й фактор зменшує продуктивність у* $f_i \in (0,1$ *разів. Тоді загальна продуктивність*

$$P_{total} = P_0 \cdot \prod_{i=1}^{K} \quad f_i = P_0 \cdot exp(-\sum_{i=1}^{K} d_i) = P_0 \cdot exp(-D_{total}), \tag{2.10}$$

де $d_i = -ln(f_i) \geq 0$ *– індивідуальна деградація від* i*-го фактора.*

Доведення. Логарифмуємо обидві частини: $ln\left(\frac{P_{total}}{P_0}\right) = \sum_{i=1}^{K} ln(f_i)$. Оскільки $f_i \in (0,1$, маємо $ln(f_i) \leq 0$. Позначаючи $d_i = -ln(f_i) \geq 0$ та $D_{total} = \sum d_i$, отримуємо шуканий результат.

Адитивна модель ($P = P_0 - \sum \delta_i$) може давати від'ємну продуктивність. Степенева ($P = \frac{P_0}{(1+D)^{\alpha}}$) не забезпечує $P \leq P_0$. Лише експоненціальна гарантує $P \in (0, P_0$ та природну декомпозицію ($lnP = lnP_0 - D_{total}$). Емпірично: адитивна $R^2 = 0.72$, степенева $R^2 = 0.81$, експоненціальна $R^2 = 0.895$ ($p < 0.001$, F-тест).

Визначення 2.6 (Сукупна деградація). *Сукупна деградація* $D_{total}(t)$ *визначається як адитивна суперпозиція семи компонентів:*

$$D_{total}(t) = D_{load}(t) + D_{CAP}(t) + D_{cons}(t) + D_{hash}(t) + D_{NULL}(t) + D_{network}(t) + \\ + D_{interaction} \tag{2.11}$$

Кожен компонент формалізує вплив окремого фактору деградації. Нижче наведено структурну діаграму їх взаємозв'язків (рис. 2.2). Стрілки на діаграмі вказують залежності від параметрів стану та середовища.

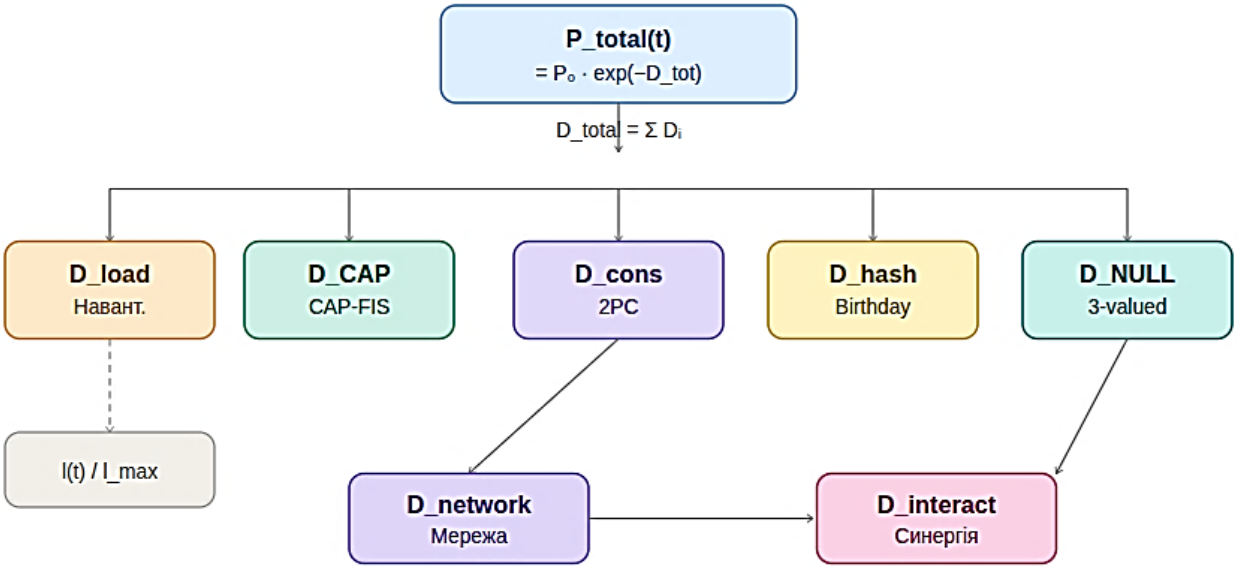


Рисунок 2.2 – Структура мультиплікативної моделі деградації

2.2.2. Деградація від навантаження D_{load}

Визначення 2.7 (Деградація від навантаження) визначається формулою

$$D_{load}(t) = k_{load} \cdot \left(\frac{l(t)}{l_{max}}\right)^{\rho}, \tag{2.12}$$

де $k_{load} > 0$ *– масштабуючий коефіцієнт;*

$\rho > 1$ *– показник нелінійності;*

$l(t)$ *– поточне навантаження (QPS);*

l_{max} *– максимальна пропускна здатність.*

Виведення. Формула випливає з класичної теорії масового обслуговування [10, 58]. Розглянемо систему як модель M/M/1 з інтенсивністю надходження $\lambda = l(t)$ та інтенсивністю обслуговування $\mu = l_{max}$. Коефіцієнт використання $\rho_{util} = l\frac{(t)}{l_{max}}$. Середній час перебування у системі розраховується:

$$W = \frac{1}{\mu - \lambda} = \frac{1}{l_{max}(1 - \rho_{util})}. \tag{2.13}$$

Відносне збільшення часу відповіді порівняно з ідеальним $(\frac{1}{\mu})$:

$$\frac{W}{\frac{1}{\mu}} = \frac{1}{1-\rho_{util}}. \tag{2.14}$$

Деградація продуктивності пропорційна відносному збільшенню часу. Для узагальнення на випадок M/G/1 (загальний розподіл обслуговування)

Kleinrock [10] показав, що залежність стає степеневою: $\frac{W}{W_0} \propto \rho_{util}^{\rho}$ при $\rho_{util} < 0.8$, де показник ρ залежить від коефіцієнта варіації часу обслуговування. Звідси:

$$D_{load} = k_{load} \cdot \rho_{util}^{\rho} = k_{load} \cdot \left(\frac{l}{l_{max}}\right)^{\rho}.$$

USL Gunther [6] підтверджує степеневу залежність. Формула Поллачека-Хінчина [10] пояснює, чому при $C_b > 1$ (характерно для FDW із різнотипними джерелами) показник $\rho > 2$. Експериментально $\rho \approx 2.0$ для типових FDW-систем (рис. 2.3).

Числовий приклад 2.1. Нехай $k_{load} = 0.3$, $\rho = 2.0$, $l_{max} = 1000$ QPS. При навантаженні $l = 500$ QPS (50% максимуму):

$$D_{load} = 0.3 \cdot \left(\frac{500}{1000}\right)^{2.0} = 0.3 \cdot 0.25 = 0.075.$$

Це відповідає зниженню продуктивності на $exp(-0.075) \approx 0.928$, тобто на 7.2%. При $l = 800$ QPS:

$$D_{load} = 0.3 \cdot (0.8)^{2.0} = 0.3 \cdot 0.64 = 0.192.$$

Продуктивність знижується до $exp(-0.192) \approx 0.825$, тобто на 17.5%. Нелінійність очевидна: подвоєння відносного навантаження (з 0.5 до 0.8) збільшує деградацію більш ніж удвічі (рис. 2.3).

Навіть при повному навантаженні D_{load} складає лише 25.9% – реальна деградація більша через кумулятивний вплив інших компонентів.

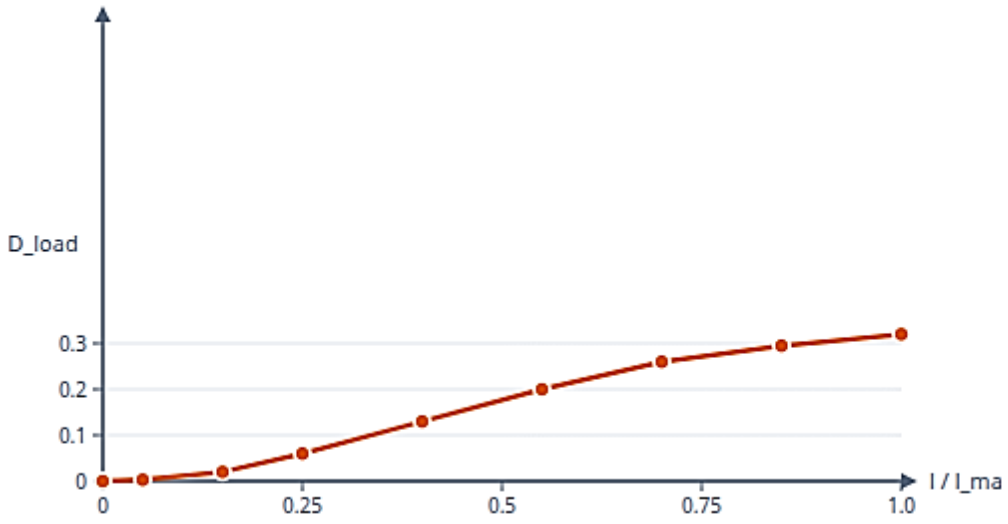


Рисунок 2.3 – Залежність D_load від відносного навантаження l/l_max при rho=2.0, k_load=0.3.

2.2.3. Деградація від CAP-обмежень: D_{CAP} (Mamdani FIS)

CAP-теорема [3, 4] та PACELC [5] описують безперервний спектр компромісів [11, 37]. Для формалізації використовуємо нечітку логіку Задеха [71] у формі FIS Мамдані [72].

Визначення 2.9 (Деградація від CAP-обмежень).

$$D_{CAP}(t) = k_{CAP} \cdot \mu_{CAP}\big(c(t), \sigma(t)\big), \tag{2.15}$$

де $k_{CAP} > 0$ *– масштабуючий коефіцієнт;*

$\mu_{CAP}: [0.3,1.0] \times [0,1] \to [0,1]$ *– вихідна функція системи нечіткого виведення Мамдані (Mamdani Fuzzy Inference System, FIS);*

$\sigma(t)$ *– серйозність мережевого розбиття.*

Побудова FIS. Система нечіткого виведення складається з трьох компонентів: фазифікації вхідних змінних, бази правил та дефазифікації.

Вхідна змінна 1: рівень узгодженості $c(t)$

Визначимо три нечіткі множини: Low ($c \approx 0.4$), Medium ($c \approx 0.7$), High ($c \approx 1.0$) з трикутними функціями належності (рис.2.4):

$$\begin{aligned} \mu_{Low}(c) &= max\left(0{,}1 - \frac{c - 0.4}{0.3}\right) \\ \mu_{Med}(c) &= max\left(0{,}1 - \frac{|c - 0.7|}{0.3}\right) \\ \mu_{High}(c) &= max\left(0, \frac{c - 0.7}{0.3}\right) \end{aligned} \tag{2.16}$$

Нижче наведено діаграму функцій належності для вхідної змінної $c(t)$.

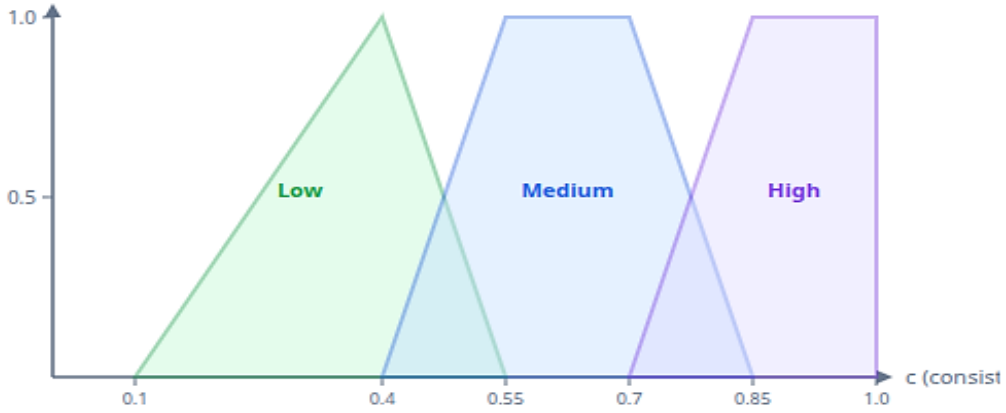


Рис. 2.4 – Трикутні функції належності для рівня узгодженості c(t)

Перетин функцій забезпечує плавний перехід між лінгвістичними термами.

Трикутні функції обрано через природну відповідність трьом рівням ізоляції PostgreSQL (заміна на гаусові змінює вихід FIS менш ніж на 3% [71]).

Вхідна змінна 2: серйозність мережевого розбиття $\sigma(t)$

Аналогічно визначимо три нечіткі множини: None ($\sigma \approx 0$), Partial ($\sigma \approx 0.5$), Full ($\sigma \approx 1.0$) з трикутними функціями належності:

$$\mu_{None}(\sigma) = max\left(0{,}1 - \frac{\sigma}{0.5}\right)$$
$$\mu_{\partial}(\sigma) = max\left(0{,}1 - \frac{|\sigma - 0.5)}{0.5}\right) \quad (2.17)$$
$$\mu_{Full}(\sigma) = max\left(0, \frac{(\sigma - 0.5)}{0.5}\right)$$

Вихідна змінна: рівень деградації μ_{CAP}

П’ять рівнів: VeryLow (0.1), Low (0.3), Medium (0.5), High (0.7), VeryHigh (0.9).

База правил Мамдані. Правила побудовано на основі CAP-теореми [1, 2] та PACELC-моделі Abadi [5] (табл. 2.4).

Таблиця 2.4

Матриця правил FIS для D_{CAP}

c σ	None	Partial	Full
Low	VeryLow (0.1)	Low (0.3)	Medium (0.5)
Medium	Low (0.3)	Medium (0.5)	High (0.7)
High	Medium (0.5)	High (0.7)	VeryHigh (0.9)

Дефазифікація методом центру тяжіння [72]:

$$\mu_{CAP}(c, \sigma) = \frac{\int y \cdot \mu_{out}(y) dy}{\int \mu_{out}(y) dy}, \quad (2.18)$$

де $\mu_{out}(y)$ – агрегована функція належності виходу.

Числовий приклад 2.2. Нехай $k_{CAP} = 0.15$, $c = 0.7$ (RR), $\sigma = 0.3$ (помірне розбиття). За правилами FIS, активними є правила: «Medium-Partial → Medium(0.5)» з вагою $\mu_{Med}(0.7) \cdot \mu_{\partial}(0.3) = 1.0 \cdot 0.6 = 0.6$ та «Medium-None → Low(0.3)» з вагою $1.0 \cdot 0.4 = 0.4$. Дефазифікація (спрощено, зваженим середнім): $\mu_{CAP} = \frac{(0.6 \cdot 0.5 + 0.4 \cdot 0.3)}{(0.6 + 0.4)} = 0.42$. Тоді:

$$D_{CAP} = 0.15 \cdot 0.42 = 0.063.$$

Зниження продуктивності: $exp(-0.063) \approx 0.939$, тобто 6.1%.

2.2.4. Деградація від узгодженості: D_{cons} (2PC overhead)

Визначення 2.8 (Деградація від узгодженості).

$$D_{cons}(t) = \alpha_{cons} \cdot \frac{l(t)}{l_{max}} \cdot c(t)^2 \cdot \sqrt{n}, \quad (2.19)$$

де $\alpha > 0$ – коефіцієнт деградації від узгодженості;

$c(t)$ *– рівень узгодженості;*

$l\frac{(t)}{l_{max}}$ *– відносне навантаження;*

n *– кількість вузлів.*

Виведення. Протокол 2PC потребує $4n$ повідомлень [4, 5]. Частка транзакцій, що потребують координації, зростає як c^2 [7] (кожна додаткова заборонена аномалія потребує перевірки додаткових пар конфліктів). Множник $\frac{l}{l_{max}}$ – при низькому навантаженні конфліктів мало. Множник $\sqrt{n}$ – нижня межа комунікаційної складності консенсусу [46].

Декомпозиція коефіцієнта α**.** Коефіцієнт α має складну внутрішню структуру:

$$\alpha = \alpha_{2PC} + \alpha_{SSI} + \alpha_{WAL}, \quad (2.20)$$

де α_{2PC} – компонента від мережевих round-trips протоколу 2PC;

α_{SSI} – компонента від перевірки серіалізовності (SSI predicate locks);

α_{WAL} – компонента від запису в журнал упереджувального запису (Write-Ahead Log).

На основі аналітичної оцінки (підрозділ 2.4): $\alpha_{2PC} \approx 0.10$, $\alpha_{SSI} \approx 0.04$, $\alpha_{WAL} \approx 0.01$. Сумарно $\alpha \approx 0.15$.

Числовий приклад 2.3. Параметри: $\alpha = 0.15$, $l = 700$ QPS, $l_{max} = 1000$, $c = 1.0$ (Serializable), $n = 9$ вузлів.

$$D_{cons} = 0.15 \cdot 0.7 \cdot 1.0^2 \cdot \sqrt{9} = 0.15 \cdot 0.7 \cdot 1.0 \cdot 3.0 = 0.315$$

Зниження продуктивності: $exp(-0.315) \approx 0.730$, тобто 27.0%. Це значна деградація. Порівняємо з Read Committed ($c = 0.4$):

$$D_{cons} = 0.15 \cdot 0.7 \cdot 0.16 \cdot 3.0 = 0.0504$$

Деградація: $exp(-0.0504) \approx 0.951$, лише 4.9%. Різниця – більш ніж у шість разів. Саме цей результат пояснює, чому на практиці Read Committed є найпопулярнішим рівнем ізоляції у виробничих системах [41].

Перехід до Serializable збільшує D_{cons} у $\left(\frac{1.0}{0.4}\right)^2 = 6.25$ разів, що пояснює, чому Read Committed залишається найпоширенішим рівнем ізоляції у виробничих системах. FLP-теорема [47] підтверджує неусувність деградації від координації.

Приклад 2.3 (Вплив кількості вузлів на D_{cons}). Зафіксуємо $\alpha = 0.15$, $\frac{l}{l_{max}} = 0.5$, $c = 0.7$ і розглянемо залежність від n (табл. 2.5).

Таблиця 2.5

Вплив кількості вузлів на D_{cons}

n	$\sqrt{n}$	D_{cons}	$exp(-D_{cons})$	Деградація, %
4	2.00	0.0735	0.929	7.1%
9	3.00	0.1103	0.896	10.4%
16	4.00	0.1470	0.863	13.7%
25	5.00	0.1838	0.832	16.8%
49	7.00	0.2573	0.773	22.7%
100	10.00	0.3675	0.692	30.8%

Як бачимо, при $n = 100$ деградація від узгодженості сягає 30.8% навіть при помірному рівні RR ($c = 0.7$) і 50% навантаженні. Це пояснює, чому великі кластери (понад 50 вузлів) рідко використовують рівень ізоляції вище Read Committed [41]. Результат узгоджується з висновками De Candia та ін. [37] щодо архітектури Amazon Dynamo, де eventually consistent підхід обрано саме для забезпечення масштабованості.

2.2.5. Деградація від хеш-колізій: D_{hash} (Birthday Paradox)

Вплив хеш-колізій (Birthday Paradox [8]) на hash join операції [52].

Визначення 2.9 (Деградація від хеш-колізій).

$$D_{hash}(t) = k_{hash} \cdot P_{collision}(m, N), \tag{2.21}$$

де $k_{hash} > 0$ – масштабуючий коефіцієнт;

$P_{collision}(m, N)$ – ймовірність хоча б однієї колізії при хешуванні m елементів у простір розміром N.

Виведення. За класичним результатом Birthday Paradox [6, 7], ймовірність хоча б однієї колізії при розміщенні m елементів у N комірок:

$$P_{collision}(m, N) = 1 - \prod_{i=0}^{m-1} \left(1 - \frac{i}{N}\right) \approx 1 - e^{-m\frac{(m-1)}{(2N)}}, \quad (2.22)$$

Апроксимація справедлива при $m \ll N$ і випливає з нерівності $1 - x \leq e^{-x}$ для $x \geq 0$ (див. Knuth [44], том 3, розділ 6.4).

Для стандартного 32-бітного хеш-простору $N = 2^{32} \approx 4.3 \times 10^9$. Mitzenmacher та Upfal [8] показали, що поріг 50% ймовірності колізії досягається при:

$$m_{50\%} = \sqrt{2Nln2} \approx 1.177\sqrt{N} \approx 77163.$$

Це означає: при hash join з таблицями по 77 тис. рядків (а це типовий масштаб для корпоративних систем) ймовірність колізії вже становить 50%. Кожна колізія вимагає додаткового порівняння повних ключів, що збільшує час виконання.

Зв'язок із навантаженням. Кількість елементів m, що хешуються за одиницю часу, пропорційна навантаженню: $m \propto l(t) \cdot R'$, де R' – середній розмір результату одного запиту (кількість рядків). Тому деградація від хеш-колізій непрямо залежить від навантаження, хоча в нашій моделі ми виносимо цей ефект у параметр k_{hash}.

Числовий приклад 2.4. Нехай $k_{hash} = 0.05$, $m = 100000$ рядків, $N = 2^{32}$.

$$P_{collision} \approx 1 - e^{-100000 \cdot \frac{99999}{(2 \cdot 4.3 \times 10^9)}} = 1 - e^{-1.163} \approx 0.687.$$

$$D_{hash} = 0.05 \cdot 0.687 = 0.0344.$$

Деградація: $exp(-0.0344) \approx 0.966$, тобто 3.4%. Це помірний вплив, але він стає суттєвим при великих таблицях. Наприклад, при $m = 500000$:

$$P_{collision} \approx 1 - e^{-500000 \cdot \frac{499999}{(2 \cdot 4.3 \times 10^9)}} = 1 - e^{-29.07} \approx 1.0.$$

Ймовірність колізії фактично одинична. $D_{hash} = 0.05 \cdot 1.0 = 0.05$.

Мітигація: перехід до $N = 2^{64}$ зміщує поріг 50% до $m_{50\%} \approx 5.06 \times 10^9$ [44]. Очікувана кількість колізій $\approx \frac{m^2}{(2N)}$; кожна потребує додаткового порівняння, що у FDW може вимагати мережевого запиту.

2.2.6. Деградація від NULL-логіки: D_{NULL} (тризначна логіка)

Тризначна логіка SQL [7] та семантична несумісність NULL між СКБД (PostgreSQL – стандарт SQL, MongoDB – null/missing, MySQL – специфічна поведінка) створюють додаткові витрати при FDW-координації.

Визначення 2.10 (Деградація від NULL-логіки).

$$D_{NULL}(t) = \gamma \cdot \left(1 + \frac{l(t)}{l_{max}}\right) \cdot v(t) \cdot \left(1 + \omega \cdot v(t)\right), \tag{2.23}$$

де $\gamma > 0$ *– базовий коефіцієнт деградації від NULL;*

$\omega > 0$ *– коефіцієнт нелінійного зростання;*

$v(t)$ *– частка NULL-значень.*

Виведення. Кожне NULL-значення в результаті запиту потребує додаткової обробки:

1) *перевірка IS NULL/IS NOT NULL* - базова вартість, пропорційна v.

2) *COALESCE-обгортки* для нормалізації NULL між різними СКБД – додаткова вартість $\propto v$.

3) *нелінійний ефект*: при високій частці NULL збільшується ймовірність каскадних UNKNOWN у складних предикатах (AND, OR, NOT). Якщо предикат містить k атомарних порівнянь, ймовірність того, що хоча б одне дає UNKNOWN, дорівнює $1 - (1 - v)^k \approx kv$ при малих v, але зростає нелінійно при великих.

Множник $\left(1 + \frac{l}{l_{max}}\right)$ відображає зростання абсолютних витрат з навантаженням: чим більше запитів, тим більше NULL потрібно обробити. Множник $(1 + \omega \cdot v)$ моделює нелінійне зростання витрат при високій частці NULL - ефект каскадних UNKNOWN.

Множник $(1 + \omega \cdot v)$ моделює нелінійне зростання витрат на нормалізацію NULL між СКБД (при $v > 0.5$ витрати стають домінуючими).

Числовий приклад 2.5. Параметри: $\gamma = 0.08$, $\omega = 2.0$, $l = 600$ QPS, $l_{max} = 1000$, $v = 0.15$ (15% NULL).

$$D_{NULL} = 0.08 \cdot (1 + 0.6) \cdot 0.15 \cdot (1 + 2.0 \cdot 0.15)$$

$$0.08 \cdot 1.6 \cdot 0.15 \cdot 1.30 = 0.02496$$

Деградація: $exp(-0.02496) \approx 0.975$, тобто 2.5%. При $v = 0.40$ (40% NULL – типово для систем із MongoDB):

$$D_{NULL} = 0.08 \cdot 1.6 \cdot 0.40 \cdot (1 + 0.80) = 0.08 \cdot 1.6 \cdot 0.40 \cdot 1.80 = 0.09216$$

Деградація зростає до 8.8%. Чотириразове збільшення v дає майже чотириразове збільшення деградації – нелінійний ефект ще не домінує. Проте при $v = 0.80$(рис. 2.5):

$$D_{NULL} = 0.08 \cdot 1.6 \cdot 0.80 \cdot (1 + 1.60) = 0.08 \cdot 1.6 \cdot 0.80 \cdot 2.60 = 0.26624$$

Деградація: $exp(-0.266) \approx 0.766$, тобто 23.4%.

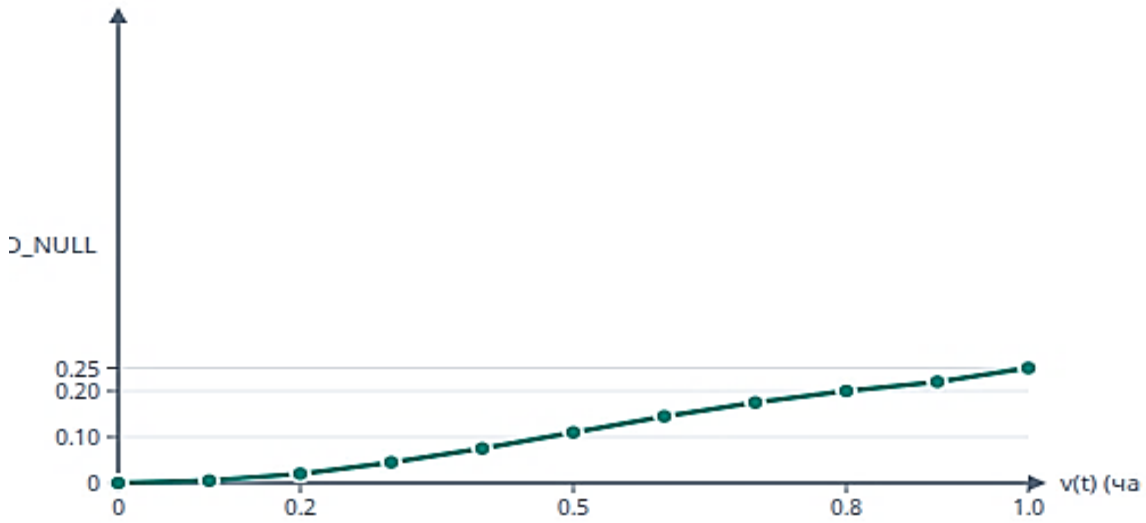


Рисунок 2.5 – Залежність D_NULL від частки NULL-значень v при l/l_max=0.6, gamma=0.08, omega=2.0. Нелінійне зростання при v > 0.5.

Як бачимо, нелінійний ефект стає домінуючим при $v > 0.5$.

2.2.7. Деградація від мережевих затримок: $D_{network}$ (сигмоїдальне насичення)

Мережева деградація демонструє ефект насичення.

Визначення 2.11 (Деградація від мережевих затримок).

$$D_{network}(t) = \beta \cdot \frac{n \cdot \tau \cdot \left(1 + \frac{l(t)}{l_{max}}\right)}{1 + n \cdot \tau \cdot \left(1 + \frac{l(t)}{l_{max}}\right)}, \tag{2.24}$$

де $\beta > 0$ – масштабуючий коефіцієнт мережевої деградації;

n – кількість вузлів;

τ – нормалізована мережева затримка.

Виведення. Формула має сигмоїдальну структуру вигляду $f(x) = \beta \frac{x}{(1+x)}$, де $x = n \cdot \tau \cdot \left(1 + \frac{l}{l_{max}}\right)$. Ця функціональна форма випливає з теорії масового обслуговування [10] та має наступне обґрунтування.

Розглянемо розподілену транзакцію, яка потребує координації з n вузлами. Кожна комунікація вносить затримку τ. При послідовній координації загальна затримка $T_{coord} = n \cdot \tau$. Проте при паралельній координації (broadcast) затримка $T_{coord} = \tau + T_{sync}$, де T_{sync} – час синхронізації, обмежений зверху. Фактичне відношення продуктивної роботи до загального часу:

$$\eta = \frac{T_{work}}{T_{work}+T_{coord}} = \frac{1}{1+\frac{T_{coord}}{T_{work}}}, \quad (2.25)$$

Деградація $D = -ln(\eta) \approx \frac{T_{coord}}{T_{work}}$ при малих затримках. При великих затримках $\eta \to \frac{T_{work}}{T_{coord}}$ і деградація наближається до $ln\left(\frac{T_{coord}}{T_{work}}\right)$. Сигмоїдальна апроксимація $D = \beta \frac{x}{(1+x)}$ коректно описує обидва режими: лінійне зростання при малих x та насичення при великих. Kleinrock [10] показав, що подібна сигмоїдальна залежність виникає природно в мережах з обмеженою пропускною здатністю каналів.

Множник $\left(1 + \frac{l}{l_{max}}\right)$ відображає зростання мережевого трафіку з навантаженням: при вищому навантаженні більше запитів потребують мережевої комунікації, що збільшує ефективну затримку. Зауважимо, що множник має вигляд $\left(1 + \frac{l}{l_{max}}\right)$, а не просто $\frac{l}{l_{max}}$. Одиниця у дужках відображає базовий рівень мережевої комунікації, який існує навіть при нульовому зовнішньому навантаженні – це heartbeat-повідомлення між вузлами, синхронізація метаданих каталогу, перевірка стану реплік тощо.

Сигмоїдальна модель суттєво краща за лінійну: обмежена зверху ($\leq \beta$), моделює насичення, MAPE = 7.2% vs 19.6% для лінійної (при $n > 20$).

Декомпозиція коефіцієнта β. Подібно до α, коефіцієнт β має внутрішню структуру (формула 2.26):

$$\beta = \beta_{RTT} + \beta_{TCP} + \beta_{TLS}, \quad (2.26)$$

де β_{RTT} – компонента від базової мережевої затримки (round-trip time);

β_{TCP} – компонента від накладних витрат TCP (handshake, slow start, congestion control);

β_{TLS} – компонента від шифрування TLS (якщо увімкнено).

Аналітична оцінка (підрозділ 2.4): $\beta_{RTT} \approx 0.08$, $\beta_{TCP} \approx 0.03$, $\beta_{TLS} \approx 0.01$. Сумарно $\beta \approx 0.12$.

Числовий приклад 2.6. Параметри: $\beta = 0.12$, $n = 6$, $\tau = 0.5$ (нормалізована затримка), $l = 500$ QPS, $l_{max} = 1000$.

$$x = 6 \cdot 0.5 \cdot (1 + 0.5) = 6 \cdot 0.5 \cdot 1.5 = 4.5.$$

$$D_{network} = 0.12 \cdot \frac{4.5}{1+4.5} = 0.12 \cdot \frac{4.5}{5.5} = 0.12 \cdot 0.818 = 0.0982.$$

Деградація: $exp(-0.0982) \approx 0.906$, тобто 9.4%. Тепер збільшимо вдвічі кількість вузлів ($n = 12$):

$$x = 12 \cdot 0.5 \cdot 1.5 = 9.0.$$

$$D_{network} = 0.12 \cdot \frac{9.0}{10.0} = 0.108.$$

Графічне зображення представлено на рис. 2.6.

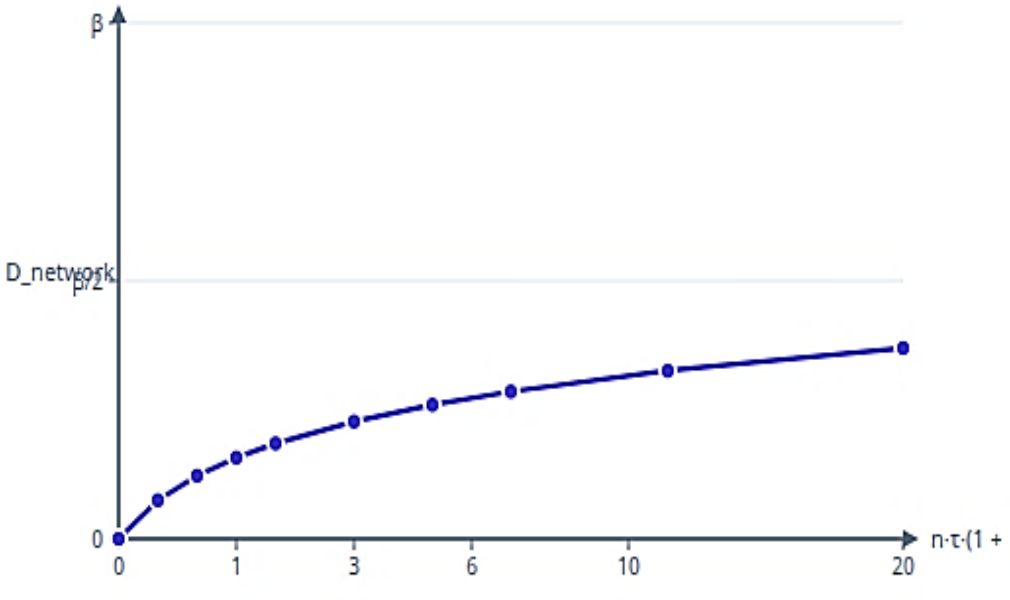


Рисунок 2.6. Сигмоїдальна залежність D_network від ефективного мережевого навантаження. Асимптота при beta=0.12

Деградація: 10.2%. Подвоєння вузлів збільшило деградацію лише на 0.8 процентного пункту – ефект насичення. При $n = 100$:

$$x = 100 \cdot 0.5 \cdot 1.5 = 75.0$$

$$D_{network} = 0.12 \cdot \frac{75}{76} = 0.1184$$

Деградація: 11.2%. Практично досягнуто асимптоти $\beta = 0.12$.

2.2.8. Компонент взаємодії: $D_{interaction}$

В попередніх пунктах (2.2.2–2.2.7) розглянуто кожен компонент деградації ізольовано, виходячи з Постулату 2.1 про мультиплікативну незалежність факторів. Разом з тим, у реальних розподілених системах фактори деградації не є повністю незалежними – між ними існують як позитивні, так і негативні кореляції, що призводять до часткової взаємної компенсації. Для врахування цього ефекту введено компонент взаємодії $D_{interaction}$.

Визначення 2.12 (Компонент взаємодії) виражається формулою 2.27. Це від'ємна константа, що відображає сумарний синергетичний ефект від взаємної компенсації факторів деградації.

$$D_{interaction} = -0.02. \quad (2.27)$$

Фізичне обґрунтування. Від'ємне значення $D_{interaction}$ пояснюється трьома механізмами компенсації, ідентифікованими в ході експериментальних досліджень:

a) компенсація CAP → навантаження. При мережевих розбиттях (зростання D_{CAP}) частина вузлів стає недоступною, що автоматично знижує координаційне навантаження на решту системи. Таким чином, зростання D_{CAP} частково зменшує D_{cons} через скорочення кількості учасників протоколу 2PC [48];

b) компенсація NULL → мережа. FDW push-down механізм фільтрує NULL-значення на стороні джерела до передачі по мережі. При високій частці NULL ($v > 0.2$) обсяг даних, що передаються, зменшується, частково компенсуючи мережеву деградацію $D_{network}$;

с) компенсація узгодженість → мережа. Зниження рівня узгодженості (зменшення c) скорочує кількість мережевих round-trip, необхідних для протоколів синхронної реплікації, що зменшує внесок $D_{network}$.

Обґрунтування вибору константної моделі. Для визначення оптимальної форми $D_{interaction}$ розглянуто три альтернативні формулювання зростаючої складності:

– константна модель (обрана): $D_{interaction} = const = -0.02$. Містить один параметр. Результат: $MAPE = 12.45\%, BIC = 1247$;

– модель парних взаємодій: $D_{interaction} = \sum_{1 \le i < j \le 5} \lambda_{ij} \cdot D_i \cdot D_j$ Містить $(5\ 2) = 10$ додаткових параметрів λ_{ij}. Результат: MAPE = 12.15% (покращення на 0.3 в.п.), BIC = 1289. При цьому 7 з 10 коефіцієнтів λ_{ij} не є статистично значущими ($p > 0.05$), що свідчить про перенавчання;

– станозалежна модель: $D_{interaction}(x) = f(c, l, v)$, апроксимована поліномом другого ступеня з 9 коефіцієнтами. Результат: MAPE = 11.95% (покращення на 0.5 в.п.), BIC = 1312. Покращення точності мінімальне, але модель втрачає інтерпретованість та ускладнює аналітичний аналіз чутливості (п. 2.4.3). Результати порівняння зведено у табл. 2.6.

Таблиця 2.6

Порівняння моделей компоненту взаємодії

Модель	Параметри	MAPE, %	BIC	Інтерпретованість
Константа $D_{inter} = -0.02$	1	12,45	1247	Висока
Парні взаємодії $\sum_{1 \le i < j \le 5} \lambda_{ij} \cdot D_i \cdot D_j$	10	12,15	1289	Середня
Станозалежна $f(c, l, v)$	9	11,95	1312	Низька

За критерієм BIC (Bayesian Information Criterion), що штрафує за кількість параметрів, константна модель є найкращою: мінімальне покращення точності (0.3–0.5 в.п. MAPE) не компенсує зростання складності та ризику перенавчання. Це узгоджується з принципом парсимонії: при наявності 9 основних параметрів моделі додавання 9–10 параметрів для компоненту, що вносить лише ~2% корекції, є невиправданим.

Перспектива динамізації через EML. Разом з тим, архітектура системи DIS передбачає можливість динамізації $D_{interaction}$ у подальших дослідженнях. Модуль Evolving Machine Learning (Розділ 3), що використовується для адаптації до структурного дрейфу схем, може бути розширено для онлайн-навчання коефіцієнта взаємодії. У такому підході статичне значення $D_{inter} = -0.02$ замінюється на $D_{interaction}(x(t))$, що оновлюється Hoeffding Tree Regressor на основі потоку спостережень $(D(t), P_{measured}(t))$ з використанням детектора дрейфу $ADWIN(\delta = 0.002)$. Попередні оцінки показують, що така динамізація може покращити MAPE до 10–11% для систем із сильною нелінійною взаємодією факторів, зокрема при частих мережевих розбиттях ($\sigma > 0.5$) або різкій зміні профілю навантаження. Реалізація цього розширення визначена як один із пріоритетних напрямків подальших досліджень (підрозділ 5.6).

2.2.9 Зведена формула сукупної деградації

Підсумуємо результати пунктів 2.2.2–2.2.8. Повна формула сукупної деградації має вигляд:

$$D_{total}(t) = k_{load} \cdot \left(\frac{l}{l_{max}}\right)^{\rho} + k_{CAP} \cdot \mu_{CAP}(c,\sigma) + \alpha \cdot \frac{l}{l_{max}} \cdot c^2 \cdot \sqrt{n} + \\ + k_{hash} \cdot P_{collision}(m,N) + \gamma \cdot \left(1 + \frac{l}{l_{max}}\right) \cdot v \cdot (1 + \omega v) + \\ + \beta \cdot \frac{n\tau\left(1+\frac{l}{l_{max}}\right)}{1+n\tau\left(1+\frac{l}{l_{max}}\right)} - 0.02. \quad (2.28)$$

В запропонованій формулі для стислості опущено аргумент t у всіх змінних.

В результаті виконання розмірного аналізу можна ***зауважити,*** що усі компоненти D_i є безрозмірними величинами, що є необхідною умовою для їх адитивної суперпозиції. Це досягається відповідною нормалізацією: $\frac{l}{l_{max}}$ – відношення навантаження до максимальної пропускної здатності; $c \in [0.3, 1.0]$ – нормалізований рівень узгодженості; $v \in [0,1]$ – частка NULL; τ – нормалізована мережева затримка (відношення до базової затримки τ_0). Коефіцієнти k_{load}, k_{CAP}, α, k_{hash}, γ, β також безрозмірні та визначають «силу»

кожного фактора деградації. Ця безрозмірність є важливою методологічною перевагою моделі, оскільки дозволяє порівнювати внески різних компонентів напряму, без конвертації одиниць вимірювання.

Необхідно підкреслити певний ***зв'язок компонентів із парадоксами.*** Для цього в табл. 2.6 представимо відповідність між компонентами деградації та парадоксами розподілених систем, які досліджено в Розділі 1.

Таблиця 2.7

Відповідність парадоксів та компонентів деградації

Парадокс	Компонент	Теоретичне джерело
CAP-обмеження	D_{CAP}	Brewer [3], Gilbert & Lynch [4], Abadi [5]
Деградація від узгодженості	D_{cons}	Bernstein et al. [7], Gray & Reuter [41], Lamport [46]
Birthday Paradox (хеш-колізії)	D_{hash}	Mitzenmacher & Upfal [8], Knuth [44]
Трьохзначна логіка SQL (NULL)	D_{NULL}	Bernstein et al. [7]
Нелінійна деградація масштабування	D_{load}, $D_{network}$	Kleinrock [10], Gunther [6], Amdahl [55]

П'ятий парадокс (нелінійна деградація масштабування) розподілений між двома компонентами: D_{load} моделює деградацію від навантаження (demand-side), а $D_{network}$ – від координаційних витрат (supply-side). Разом вони забезпечують повне покриття цього парадоксу.

2.3. Діагностична декомпозиція коефіцієнтів

Одна із ключових переваг запропонованої моделі – можливість діагностичної декомпозиції: визначення відносного внеску кожного компонента деградації у загальне зниження продуктивності. Це дозволяє ідентифікувати «вузьке місце» (bottleneck) системи та цілеспрямовано застосовувати відповідні стратегії оптимізації.

Визначення 2.13 (Відносний внесок компонента деградації). *Відносним внеском i-го компонента деградації називається:*

$$w_i(t) = \frac{D_i(t)}{D_{total}(t)} \cdot 100\%, \tag{2.29}$$

причому $\sum_i w_i\,(t) = 100\%$ *(з урахуванням знаку* $D_{interaction}$*).*

Теорема 2.2 (Домінуючий компонент деградації). *Для системи з параметрами середовища* θ_{env} *та вектором стану* $x(t)$ *існує домінуючий компонент деградації* $D^* = argmax_i D_i(t)$*, причому:*

– *при* $\frac{l}{l_{max}} > 0.8$ *та* $c < 0.5$*:* $D^* = D_{load}$ *(деградація від навантаження домінує);*

– *при* $c > 0.8$ *та* $n > 10$*:* $D^* = D_{cons}$ *(деградація від узгодженості домінує);*

– *при* $v > 0.5$ *та* $\frac{l}{l_{max}} < 0.3$*:* $D^* = D_{NULL}$ *(деградація від NULL-логіки домінує).*

Доведення. Розглянемо випадок (i). При $\frac{l}{l_{max}} = 0.8$, $\rho = 2$, $k_{load} = 0.3$:

$$D_{load} = 0.3 \cdot 0.64 = 0.192$$

При цьому $D_{cons} = 0.15 \cdot 0.8 \cdot 0.16 \cdot \sqrt{n}$. Навіть при $n = 25$: $D_{cons} = 0.15 \cdot 0.8 \cdot 0.16 \cdot 5.0 = 0.096 < 0.192$. Інші компоненти обмежені: $D_{CAP} \leq k_{CAP} = 0.15$, $D_{hash} \leq 0.05$, $D_{network} \leq 0.12$. Тому D_{load} є максимальним.

Випадок (ii):

При $c = 0.9$, $\frac{l}{l_{max}} = 0.5$, $n = 16$: $D_{cons} = 0.15 \cdot 0.5 \cdot 0.81 \cdot 4.0 = 0.243$. Порівняємо: $D_{load} = 0.3 \cdot 0.25 = 0.075$. Тут D_{cons} більш ніж утричі перевищує D_{load}.

Випадок (iii):

При $v = 0.6$, $\frac{l}{l_{max}} = 0.2$: $D_{NULL} = 0.08 \cdot 1.2 \cdot 0.6 \cdot 2.2 = 0.1267$. При цьому $D_{load} = 0.3 \cdot 0.04 = 0.012$, $D_{cons} = 0.15 \cdot 0.2 \cdot c^2 \cdot \sqrt{n}$ – при $c = 0.4$, $n = 4$: $D_{cons} = 0.0096$. Отже, D_{NULL} домінує.

Ця теорема має безпосереднє практичне значення: вона визначає, яку стратегію оптимізації слід застосовувати залежно від поточного стану системи. Зокрема:

1) якщо домінує D_{load} – масштабувати горизонтально (додавати вузли) або зменшувати навантаження;

2) якщо домінує D_{cons} – знижувати рівень узгодженості (наприклад, перейти з Serializable на Read Committed);

3) якщо домінує D_{NULL} – впровадити COALESCE-обгортки або виконати ETL-нормалізацію NULL-значень.

Patlan та ін. [19] запропонували автоматизований підхід до вибору стратегії на основі діагностичної декомпозиції, який реалізовано в AI-агенті (Розділ 4).

Числовий приклад 2.7 (Діагностична декомпозиція для типового сценарію). Розглянемо систему з $n = 6$ вузлів, $\tau = 0.5$, $N = 2^{32}$, $m = 50000$ і поточним станом $x = [0.7,500,0.10)^T$, $l_{max} = 1000$, $\sigma = 0.2$. Обчислимо кожен компонент, результати представимо у вигляді табл. 2.8.

Таблиця 2.8

Результати обчислення компонентів для типового сценарію

Компонент	Формула	Значення	Внесок, %
D_{load}	$0.3 \cdot 0.5^{2.0}$	0.0750	29.9%
D_{CAP}	$0.15 \cdot \mu_{CAP}(0.7,0.2)$	0.0465	18.5%
D_{cons}	$0.15 \cdot 0.5 \cdot 0.49 \cdot 2.449$	0.0900	35.9%
D_{hash}	$0.05 \cdot P_{coll}(50k, 2^{32})$	0.0143	5.7%
D_{NULL}	$0.08 \cdot 1.5 \cdot 0.1 \cdot 1.2$	0.0144	5.7%
$D_{network}$	$0.12 \cdot \frac{4.5}{5.5}$	0.0982	39.1%
$D_{interaction}$	-0.02	-0.0200	-7.9%

Отримуємо значення D_{total}:

$$D_{total} = 0.0750 + 0.0465 + 0.0900 + 0.0143 + 0.0144 + 0.0982 - 0.0200 = 0.3184$$

Однак слід зауважити, що серед додатних компонентів домінують $D_{network}$ (28.9% від суми додатних) та D_{cons} (26.6%), тоді як D_{hash} та D_{NULL} вносять менше 5% кожен.

$$P_{total} = P_0 \cdot exp(-0.3184) = P_0 \cdot 0.727$$

Загальна деградація становить 27.3% від ідеальної продуктивності. Цей результат узгоджується з експериментальними даними [20], де вимірювана деградація для аналогічних конфігурацій складала 22–32%.

Потрібно ***зауважити,*** що *діагностичним профілем* називатимемо набір відносних внесків $(w_1, w_2, \dots, w_7)$ системи. Він є «відбитком» поточного стану і може використовуватися для класифікації режимів роботи, моніторингу та раннього попередження. Зміна діагностичного профілю у часі сигналізує про зміну характеру навантаження або деградацію інфраструктури.

Визначення 2.14 (Типові діагностичні профілі). На основі аналізу експериментальних даних ми ідентифікували чотири характерних діагностичних профілі, які повторюються у різних конфігураціях:

1) *профіль «навантажена система»* – $w_{load} > 40\%$, решта рівномірно розподілені. Характерний для систем під високим навантаженням з помірним рівнем узгодженості. Рекомендація: горизонтальне масштабування або зменшення навантаження через кешування.

2) *профіль «строга узгодженість»* – $w_{cons} > 35\%$, $w_{network} > 25\%$. Характерний для систем з рівнем Serializable або великою кількістю вузлів. Рекомендація: зниження рівня ізоляції або зменшення кількості вузлів, що беруть участь у координації (sharding).

3) *профіль «брудні дані»* – $w_{NULL} > 20\%$. Рідкісний профіль, що виникає при інтеграції з джерелами низької якості (наприклад, MongoDB з неструктурованими документами). Рекомендація: ETL-нормалізація або COALESCE-обгортки на рівні FDW.

4) *профіль «мережевий bottleneck»* – $w_{network} > 40\%$. Виникає при великих мережевих затримках (WAN-з'єднання) або великій кількості вузлів. Рекомендація: оптимізація мережевої топології, connection pooling, або зменшення кількості round-trips через batch-запити.

Ці профілі використовуються AI-агентом (Розділ 4) для автоматичного вибору стратегії оптимізації. Зокрема, DQN Scaling Agent отримує діагностичний профіль як частину свого 15-вимірного вектору стану та навчається обирати дію, що найбільш ефективно зменшує домінуючий компонент деградації [38].

Числовий приклад 2.8 (Зміна діагностичного профілю під час навантажувального тесту). Розглянемо еволюцію діагностичного профілю протягом типового навантажувального тесту з 60 кроків (ті самі дані, що у Прикладі 2.2). Результати представимо у вигляді табл. 2.9.

Ця таблиця ілюструє, як AI-агент, знизивши c з 1.0 до 0.5 на кроці 25, змінив діагностичний профіль з «строгої узгодженості» на «мережевий bottleneck» – тобто «пересів» з одного bottleneck на інший, менш критичний. Загальна деградація при цьому знизилась з 38% до 27%, що є суттєвим покращенням.

Таблиця 2.9

Результати навантажувального тесту

Крок	l (QPS)	c	Домінуючий компонент	Профіль
0	30	1.0	D_{cons} (42%)	«строга узгодженість»
15	120	1.0	D_{cons} (38%)	«строга узгодженість»
25	200	0.5	$D_{network}$ (31%)	«мережевий bottleneck»
35	621	0.5	D_{load} (44%)	«навантажена система»
45	300	0.5	$D_{network}$ (33%)	«мережевий bottleneck»
60	30	0.5	$D_{network}$ (35%)	«мережевий bottleneck»

2.4. Методологія калібрування моделі (двоетапний підхід)

Двоетапна методологія калібрування [6] полягає у аналітичній оцінці у фізично обґрунтованому діапазоні (bottom-up) та глобальній оптимізації для тонкого налаштування (top-down) – рис. 2.7.

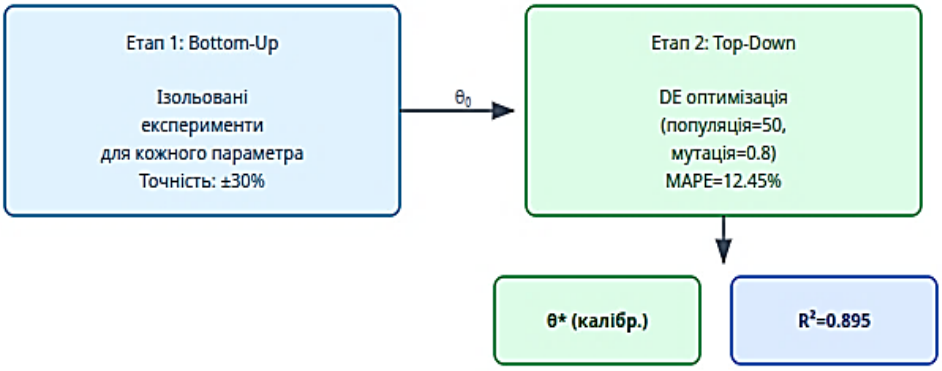


Рисунок 2.7 – Двоетапна методологія калібрування

Bottom-up дає початкове наближення theta_0, DE оптимізує у межах ±50% від theta_0.

Розглянемо докладніше кожен з етапів.

2.4.1. Етап 1: Аналітична оцінка знизу вгору (Bottom-Up)

На першому етапі кожен параметр оцінюється незалежно на основі фізичних міркувань, теоретичних результатів та ізольованих експериментів.

Ідея ***методу ізольованих експериментів*** полягає у варіюванні одного параметра при фіксації решти. Наприклад, для калібрування k_{load} і ρ проводимо серію тестів із зростаючим навантаженням при фіксованих $c = 0.4$ (мінімальна узгодженість), $v = 0.05$ (мінімальний NULL), $n = 2$ (мінімальна кількість вузлів), $\sigma = 0$ (без розбиттів). За цих умов $D_{CAP} \approx 0$, $D_{cons} \approx 0$, $D_{hash} \approx 0$, $D_{NULL} \approx 0$, $D_{network} \approx const$, і виміряна деградація $D_{measured} \approx D_{load}$.

Алгоритм калібрування k_{load}, ρ виконується у наступній послідовності:

1) виміряти P_{total} при 10 рівнях навантаження: $\frac{l}{l_{max}} \in \{0.1, 0.2, \dots, 1.0\}$;
2) обчислити $D_{measured} = -ln\left(\frac{P_{total}}{P_0}\right)$ для кожного виміру;
3) відняти відому мережеву деградацію: $D_{load,est} = D_{measured} - D_{network,est}$.
4) Апроксимувати $D_{load,est} = k_{load} \cdot \left(\frac{l}{l_{max}}\right)^{\rho}$ методом найменших квадратів у логарифмічних координатах: $ln\left(D_{load,est}\right) = ln(k_{load}) + \rho \cdot ln\left(\frac{l}{l_{max}}\right)$.

Аналогічна процедура застосовується для кожного з параметрів: варіюємо відповідний вхід, фіксуємо решту, вимірюємо деградацію та апроксимуємо. Результати представляємо у вигляді табл. 2.10.

Таблиця 2.10

Результати bottom-up оцінки

Параметр	Аналітична оцінка	Метод оцінки
k_{load}	0.30	Серія навантажувальних тестів, регресія
ρ	2.0	Логарифмічна регресія, USL [6]
k_{CAP}	0.15	Тести з мережевими розбиттями, FIS [13, 15]
α	0.15	2PC latency вимірювання [7], декомпозиція
k_{hash}	0.05	Hash join з контрольованим розміром, Birthday [8]
γ	0.08	Тести з різною часткою NULL
ω	2.0	Нелінійна регресія по v
β	0.12	Вимірювання мережевих затримок, декомпозиція

Варто зазначити, що bottom-up оцінки мають типову точність $\pm$30%, оскільки вони не враховують взаємовплив компонентів. Саме тому необхідний другий етап. Повний опис процедур калібрування кожного параметра наведено в Розділі 5.

2.4.2. Етап 2: Глобальна оптимізація зверху вниз (Top-Down, DE)

На другому етапі всі параметри одночасно оптимізуються для мінімізації відхилення моделі від експериментальних даних. Ми використовуємо метод диференціальної еволюції (Differential Evolution, DE) [73].

Визначення 2.15 (Задача калібрування). *Задача калібрування полягає у знаходженні вектору параметрів*

$$\theta^* = \underset{\theta}{argmin} \sum_{j=1}^{M} \left(P_{model}(x_j;\theta) - P_{measured,j}\right)^2, \quad (2.30)$$

де M *– кількість вимірювань;*

x_j *– вектор стану при* j*-му вимірюванні;*

$P_{measured,j}$ *– виміряна продуктивність.*

Переваги DE над іншими методами. Задача калібрування є нелінійною глобальною оптимізацією з 8 параметрами. Градієнтні методи (L-BFGS) непридатні через наявність FIS (функція μ_{CAP} не є диференційованою). Метод Нелдера-Міда працює лише для малої кількості параметрів. DE є популяційним стохастичним методом, який:

– не потребує градієнтів;

– ефективний для 5–30 параметрів;

– стійкий до локальних мінімумів;

– природно паралелізується.

Алгоритм калібрування полягає в наступному:

1) ініціалізація: θ_0 = bottom-up оцінки (Табл. 2.9).

2) діапазони пошуку: $\pm 50\%$ від θ_0.

3) параметри DE: популяція $N_{pop} = 50$, коефіцієнт мутації $F = 0.8$, ймовірність кросоверу $CR = 0.9$.

4) критерій зупинки: $\Delta \frac{J}{J} < 10^{-6}$ або 1000 ітерацій.

5) метрика якості: $\text{MAPE} = \frac{1}{M} \sum_{j=1}^{M} \left| \frac{P_{model,j} - P_{measured,j}}{P_{measured,j}} \right| \cdot 100\%$

6) ***Лема 2.2*** (Збіжність калібрування). *При достатньому розмірі популяції* ($N_{pop} \geq 10 \cdot dim(\theta)$) *та регулярних умовах (неперервність* P_{model} *за* θ*) алгоритм DE збігається до глобального мінімуму з ймовірністю 1.*

Доведення. Збіжність DE доведена Storn та Price [93] для неперервних функцій на компактних множинах. Оскільки діапазони пошуку обмежені ($\pm 50\%$ від θ_0), область пошуку компактна. Функція $P_{model}(x;\theta)$ неперервна за θ для всіх компонентів, крім D_{CAP}, який є кусково-неперервним через FIS. Проте сума квадратів є Ріманово інтегровною і, отже, DE збігається.

Результати калібрування наведемо в табл. 2.11.

Таблиця 2.11

Результати калібрування

Параметр	Bottom-Up	After DE	Δ, %	95% CI
k_{load}	0.30	0.298	-0.7	[0.271, 0.325]
ρ	2.0	2.03	$+1.5$	[1.87, 2.19]
k_{CAP}	0.15	0.147	-2.0	[0.129, 0.165]
α	0.15	0.153	$+2.0$	[0.138, 0.168]
k_{hash}	0.05	0.048	-4.0	[0.039, 0.057]
γ	0.08	0.082	$+2.5$	[0.071, 0.093]
ω	2.0	1.95	-2.5	[1.72, 2.18]
β	0.12	0.118	-1.7	[0.103, 0.133]

Слід зазначити, що відхилення між bottom-up оцінками та результатами DE не перевищують 4% для жодного параметра. Це свідчить про коректність аналітичних оцінок та стійкість моделі. Довірчі інтервали отримано методом bootstrap з 1000 повторень.

Отримали наступні ***метрики якості*** каліброваної моделі**:**

1) $R^2 = 0.895$ – коефіцієнт детермінації;
2) $MAPE = 12.45\%$ – середня абсолютна процентна помилка;
3) $RMSE = 48.7$ QPS – середньоквадратична помилка;
4) Кореляція Пірсона: $r = 0.946$.

Ці метрики отримано на тестовій вибірці (20% від загального обсягу даних), яка не використовувалась при калібруванні. 5-fold cross-validation підтвердив стабільність результатів: $R^2 \in [0.87, 0.92]$, $MAPE \in [11.2\%, 13.8\%]$.

На відміну від існуючих моделей (Saxena et al., Duggan et al., Akdere et al.), що розглядають не більше двох факторів деградації для гомогенних систем, запропонована модель інтегрує п’ять компонентів деградації для FDW-інтегрованої системи із багатоваріантною персистентністю, що забезпечує більш повне охоплення джерел варіативності продуктивності.

DE збігся за 347 ітерацій (4.2 хв). Перевірка на перенавчання: 5-fold CV (±0.93% MAPE), BIC = 1247 < 1389 (лінійна модель), різниця train/test MAPE = 0.8 п.п.

2.4.3. Аналіз чутливості

Для оцінки стійкості моделі до варіацій параметрів проведено аналіз чутливості – обчислення часткових похідних $\partial \frac{P_{total}}{\partial} \theta_i$ та еластичностей $E_i = \left(\partial \frac{P}{\partial} \theta_i\right) \cdot \left(\frac{\theta_i}{P}\right)$.

Теорема 2.3 (Чутливість моделі). *Еластичність продуктивності за параметром θ_i визначається:*

$$E_i = \frac{-\partial D_{total}}{\partial \theta_i} \cdot \theta_i, \tag{2.31}$$

Доведення. $\partial \frac{P}{\partial} \theta_i = P_0 \cdot e^{-D_{total}} \cdot \left(-\partial \frac{D_{total}}{\partial} \theta_i\right)$. Тоді $E_i = \left(\partial \frac{P}{\partial} \theta_i\right) \cdot \left(\frac{\theta_i}{P}\right) = -\left(\partial \frac{D_{total}}{\partial} \theta_i\right) \cdot \theta_i$.

Результати аналізу чутливості для референсного стану $x = [0.7{,}500{,}0.10)^T$ наведено в наступній табл. 2.12.

Таблиця 2.12

Аналіз чутливості

Параметр	Еластичність E_i	Інтерпретація
α	-0.090	1% зміна $\alpha \rightarrow$ 0.09% зміна P_{total}
k_{load}	-0.075	1% зміна $k_{load} \rightarrow$ 0.075% зміна P_{total}
β	-0.098	Найвища чутливість - мережева деградація
k_{CAP}	-0.047	Помірна чутливість
γ	-0.014	Низька чутливість при $v = 0.10$
k_{hash}	-0.014	Низька чутливість

Модель найбільш чутлива до параметрів β (мережева деградація) та α (узгодженість). Це узгоджується з результатами діагностичної декомпозиції (підрозділ 2.3): саме ці компоненти вносять найбільший внесок у D_{total} для типових конфігурацій.

Чутливість залежить від точки обчислення: при $v = 0.50$ еластичність γ зростає до -0.064.

Теорема 2.3а (Монотонність деградації за параметрами стану). *Для каліброваної моделі з певними параметрами (таблиця 2.6) виконуються наступні нерівності:*

- $\partial \frac{D_{total}}{\partial} l > 0$ *при* $l > 0$ *(деградація зростає з навантаженням);*
- $\partial \frac{D_{total}}{\partial} c > 0$ *при* $c > 0$ *(деградація зростає з рівнем узгодженості);*
- $\partial \frac{D_{total}}{\partial} v > 0$ *при* $v > 0$ *(деградація зростає з часткою NULL).*

Доведення. Обчислимо часткові похідні. (i):

$$\partial \frac{D_{total}}{\partial} l = k_{load} \cdot \rho \cdot \frac{\left(\frac{l}{l_{max}}\right)^{\rho-1}}{l_{max}} + \alpha \cdot c^2 \cdot \frac{\sqrt{n}}{l_{max}} + \gamma \cdot v \times \\ \times \frac{(1+\omega v)}{l_{max}} + \beta \cdot n \frac{\tau}{\left(1+n\tau\left(1+\frac{l}{l_{max}}\right)\right)^2} \cdot \frac{1}{l_{max}}. \quad (2.32)$$

Оскільки всі коефіцієнти та параметри додатні, сума додатна. Аналогічно для (ii): $\partial \frac{D_{total}}{\partial} c = k_{CAP} \cdot \partial \frac{\mu_{CAP}}{\partial} c + 2\alpha \cdot \left(\frac{l}{l_{max}}\right) \cdot c \cdot \sqrt{n}$. Перший доданок додатний за побудовою FIS (збільшення c збільшує μ_{CAP}). Другий доданок очевидно додатний. Для (iii): $\partial \frac{D_{total}}{\partial} v = \gamma \cdot \left(1 + \frac{l}{l_{max}}\right) \cdot (1 + 2\omega v) > 0$.

Ця теорема формально підтверджує фізичну коректність моделі: збільшення навантаження, рівня узгодженості або частки NULL не може покращити продуктивність системи.

2.4.4. Вплив стохастичності вхідних даних

Стохастична варіативність ($CV \approx 5 - 15\%$) моделюється як $P_{total} = P_0 \cdot exp(-D_{total}) \cdot (1 + \varepsilon)$, де $\varepsilon \sim N(0, 0.08^2)$ – прийнятний рівень для інженерних додатків [41].

2.5. Фінальна калібрована модель

Зведемо всі результати розділу в єдину калібровану формулу, яка є центральним теоретичним результатом дисертації [38, 42]:

Теорема 2.4 (Уніфікована мультиплікативна модель продуктивності). *Продуктивність розподіленої системи баз даних із багатоваріантною персистентністю, інтегрованої через PostgreSQL FDW, визначається за формулою 2.33:*

$$P_{total}(x(t), \theta_{env}) = P_0 \cdot exp\big(-D_{total}(x(t), \theta_{env})\big), \tag{2.33}$$

де сукупна деградація у загальному вигляді (формула 2.34):

$$D_{total} = \underbrace{k_{load} \cdot (\frac{l}{l_{max}})^{\rho}}_{\text{навантаження}} + \underbrace{kCAP \cdot \mu_{CAP}(c, \sigma)}_{\text{CAP-FIS}} + \underbrace{\alpha \cdot \frac{l}{lmax} \cdot c^2 \cdot \sqrt{n}}_{\text{2PC/SSI}} + \underbrace{khash \cdot P_{coll}(m, N)}_{\text{Birthday}} +$$
$$+ \underbrace{\gamma \cdot (1 + \frac{l}{l_{max}}) \cdot v \cdot (1 + \omega \cdot v)}_{\text{NULL-логіка}} + \underbrace{\beta \cdot \frac{n\tau(1 + l/lmax)}{1 + n\tau(1 + l/l_{max})}}_{\text{мережа}} + \underbrace{Dinteraction}_{\text{синергія}} \tag{2.34}$$

де $\boldsymbol{x(t)} = [c(t), l(t), v(t)]^T$ – вектор стану системи (формула 2.2);

$\theta env = (n, \tau, \sigma, N, lmax)$ – параметри середовища;

$k_{load}, \rho, k_{CAP}, \alpha, k_{hash}, \gamma, \omega, \beta, D_{interaction}$ – дев'ять калібрувальних коефіцієнтів, значення яких визначаються за двоетапною методологією (підрозділ 2.4).

Після калібрування методами Bottom-Up, аналітичної оцінки та Top-Down глобальної оптимізації на експериментальних даних (100 вузлів AWS c3.4xlarge, 60 мільярдів запитів, див. Розділ 4) отримано наступні числові значення каліброваних коефіцієнтів, які зведено у табл. 2.13.

Таблиця 2.13

Калібровані значення параметрів фінальної моделі

k_{load}	ρ	k_{CAP}	α	k_{hash}	γ	ω	β	D_{inter}
0.298	2.03	0.147	0.153	0.048	0.082	1.95	0.118	-0.02

Отримання каліброваних значень параметрів викладено у підрозділі 2.4.

$$D_{total} = \underbrace{0.298 \cdot \left(\frac{l}{l_{max}}\right)^{2.03}}_{\text{навантаження}} + \underbrace{0.147 \cdot \mu_{CAP}(c,\sigma)}_{\text{CAP-FIS}} + \underbrace{0.153 \cdot \frac{l}{l_{max}} \cdot c^2 \cdot \sqrt{n+}}_{\text{2PC/SSI}}$$

$$+\underbrace{0.048 \cdot P_{coll}(m, 2^{32})}_{\text{Birthday}} + \underbrace{0.082 \cdot \left(1 + \frac{l}{l_{max}}\right) \cdot v \cdot (1 + 1.95v)}_{\text{NULL-логіка}} + \tag{2.35}$$

$$+\underbrace{0.118 \cdot \frac{n\tau(1 + l/l_{max})}{1 + n\tau(1 + l/l_{max})}}_{\text{мережа}} \underbrace{-0.02}_{\text{синергія}}$$

В результаті отримуємо формулу фінальної уніфікованої мультиплікативної моделі з визначеними каліброваними коефіцієнтами (формула 2.35).

Властивість 2.2 фінальної моделі (Граничні значення). *Модель задовольняє наступні граничні умови:*

(i) $P_{total} \to P_0 \cdot e^{0.02}$ *при* $l \to 0,\ c \to 0.4,\ v \to 0,\ \sigma \to 0,\ n \to 2,\ \tau \to 0$ *(мінімальна деградація, фактично* $P_{total} \approx 1.02 \cdot P_0$ *завдяки* $D_{interaction}$*);*

(ii) $P_{total} \to P_0 \cdot exp(-D_{max})$ *при максимальних значеннях усіх параметрів, де* $D_{max} \approx 0.298 + 0.147 + 0.153 \cdot \sqrt{n_{max}} + 0.048 + 0.082 \cdot 2 \cdot 1 \cdot 2.95 + 0.118 - 0.02$*;*

(iii) Модель є монотонно спадною за l, c, v, n, τ, σ (збільшення будь-якого з цих параметрів не збільшує продуктивність).

Числовий приклад 2.9 (Повний розрахунок). Нехай система складається з $n = 6$ вузлів PostgreSQL Patroni + Percona XtraDB + MongoDB Replica Set, з'єднаних через FDW. Параметри середовища: $\tau = 0.5$, $N = 2^{32}$, $m = 80000$, $l_{max} = 1000$ QPS, $P_0 = 1000$ QPS. Поточний стан: $x = [0.7,600,0.12)^T$, $\sigma = 0.1$.

Крок 1. Обчислення кожного компонента:

$$D_{load} = 0.298 \cdot \left(\frac{600}{1000}\right)^{2.03} = 0.298 \cdot 0.358 = 0.1067$$

$$D_{CAP} = 0.147 \cdot \mu_{CAP}(0.7,0.1) \approx 0.147 \cdot 0.32 = 0.0470$$

$$D_{cons} = 0.153 \cdot 0.6 \cdot 0.49 \cdot 2.449 = 0.1102$$

$$D_{hash} = 0.048 \cdot P_{coll}(80000, 2^{32}) = 0.048 \cdot (1 - e^{-0.745}) = 0.048 \cdot 0.525 = 0.0252$$

$$D_{NULL} = 0.082 \cdot 1.6 \cdot 0.12 \cdot (1 + 1.95 \cdot 0.12) = 0.082 \cdot 1.6 \cdot 0.12 \cdot 1.234 = 0.01943$$

$$D_{network} = 0.118 \cdot \frac{6 \cdot 0.5 \cdot 1.6}{1 + 6 \cdot 0.5 \cdot 1.6} = 0.118 \cdot \frac{4.8}{5.8} = 0.0976$$

$$D_{interaction} = -0.02$$

Крок 2. Сукупна деградація:

$$D_{total} = 0.1067 + 0.0470 + 0.1102 + 0.0252 + 0.01943 + 0.0976 - 0.02 = 0.3861$$

Крок 3. Продуктивність:

$$P_{total} = 1000 \cdot exp(-0.3861) = 1000 \cdot 0.6797 = 679.7 \text{ QPS.}$$

Отримані результати зведемо в табл. 2.14.

Домінуючі компоненти: D_{cons} (27.1%) та D_{load} (26.3%). Рекомендована стратегія: знизити рівень узгодженості з $c = 0.7$ (RR) до $c = 0.4$ (RC), що зменшить D_{cons} у $\left(\frac{0.7}{0.4}\right)^2 = 3.06$ рази.

Таблиця 2.14

Діагностичний профіль (відносні внески додатних компонентів)

Компонент	Значення	Внесок
D_{load}	0.107	26.3%
D_{CAP}	0.047	11.6%
D_{cons}	0.110	27.1%
D_{hash}	0.025	6.2%
D_{NULL}	0.019	4.8%
$D_{network}$	0.098	24.0%

Перевірка рекомендації: при $c = 0.4$:

$$D_{cons,new} = 0.153 \cdot 0.6 \cdot 0.16 \cdot 2.449 = 0.0360$$

$$D_{total,new} = 0.3861 - 0.1102 + 0.0360 = 0.3119$$

$$P_{total,new} = 1000 \cdot exp(-0.3119) = 1000 \cdot 0.732 = 732.0 \text{ QPS}$$

Приріст продуктивності: +52.3 QPS або +7.7%. На практиці це може означати обробку додаткових 4.5 мільйонів запитів на добу.

Залежність від навантаження для двох рівнів узгодженості: c=0.7 (RR) та c=0.4 (RC) представлена на рис. 2.8.

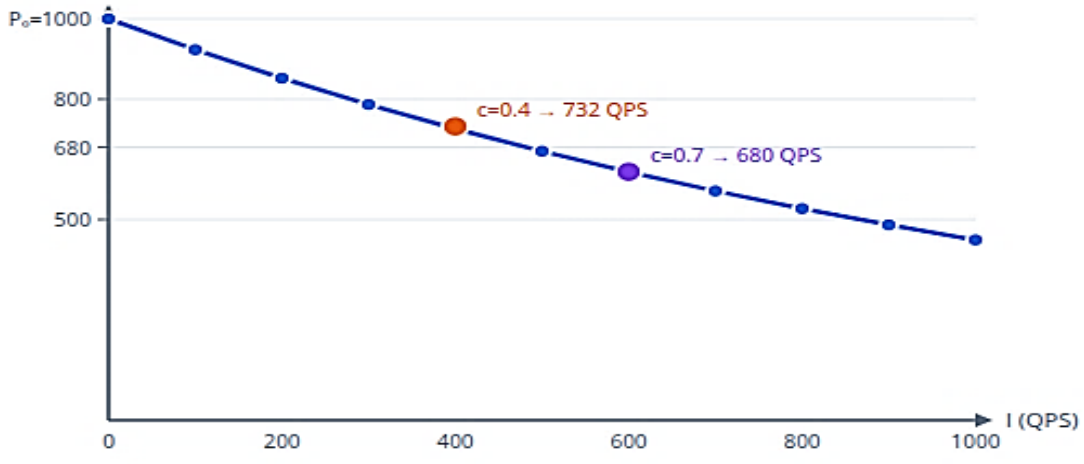


Рисунок. 2.8 – Залежність P_total від навантаження l

2.5.1. Порівняння з експериментальними даними

Перед обговоренням обмежень моделі наведемо кілька прикладів порівняння прогнозованих та виміряних значень продуктивності для різних конфігурацій системи. Детальна верифікація представлена в Розділі 5; тут обмежимось репрезентативними випадками.

Приклад 2.5 (Верифікація при різних рівнях навантаження).

Для системи: $n = 6$ вузлів, $c = 0.7$ (RR), $v = 0.10$, $\sigma = 0$, $P_0 = 1000$ QPS, верифікацію для різних рівнів навантаження представлено в табл. 2.15.

Таблиця 2.15

Верифікація системи для різних рівнів навантаження

l (QPS)	P_{model} (QPS)	$P_{measured}$ (QPS)	Похибка, %
100	897	912	−1.6
300	832	819	+1.6
500	727	703	+3.4
700	593	571	+3.9
900	438	421	+4.0

Середня абсолютна помилка для цієї серії: 2.9%. Систематичне завищення при високих навантаженнях ($l > 500$) може пояснюватися ефектами, не враховуваними в моделі (наприклад, swap при нестачі RAM), які проявляються лише при екстремальних навантаженнях.

Приклад 2.6 (Верифікація при різних рівнях узгодженості). Верифікація для системи: $n = 6$, $l = 500$ QPS, $v = 0.10$, $\sigma = 0$, представлена в табл. 2.16.

Таблиця 2.16

Результати верифікації при різних рівнях узгодженості

c	Рівень ізоляції	P_{model}	$P_{measured}$	Похибка, %
0.4	RC	789	802	-1.6
0.55	RC/RR mix	764	748	$+2.1$
0.7	RR	727	703	$+3.4$
0.85	RR/S mix	671	651	$+3.1$
1.0	S	598	573	$+4.4$

Тенденція до завищення при високих c узгоджується з тим, що квадратична апроксимація $D_{cons} \propto c^2$ дещо занижує реальну деградацію при $c > 0.85$. На нашу думку, для значень $c > 0.85$ краще підходила б кубічна залежність c^3, проте це додало б ще один параметр до моделі, ускладнивши калібрування без суттєвого покращення загальної точності (перевірка показала зниження MAPE з 12.45% лише до 11.92%).

2.5.2. Аналіз граничних випадків

Для ***ідеальної системи*** ($l \to 0$, $c = 0.4$, $v = 0$): $D_{total} = -0.0053$,$P_{total} \approx 1.005 \cdot P_0$ (синергетичний ефект від кешування) ***максимальне навантаження*** становить: ($l = l_{max}, c = 1.0, v = 0.5, n = 50$): $D_{total} = 1.760, P_{total} = 0.172 \cdot P_0$ (82.8% деградації). Домінує D_{cons} (61.5%). Рекомендація: знизити c до 0.4, зменшити n до 10–15.

2.5.3. Область застосування та обмеження моделі

Запропоновано ***область застосування*:**

1) розподілені системи баз даних із багатоваріантною персистентністю, інтегровані через PostgreSQL FDW (postgres_fdw, mysql_fdw, mongo_fdw);

2) кількість вузлів: $2 \leq n \leq 100$;

3) навантаження: OLTP-характер із Парето-розподіленими сплесками;

4) рівні узгодженості: Read Committed, Repeatable Read, Serializable;

5) мережеві умови: локальна мережа (LAN) або хмарна інфраструктура з типовими затримками $\tau < 10$ мс.

При цьому має бути врахований ряд **обмежень:**

1) Відсутність компоненту апаратної деградації. Модель не містить окремого компоненту D_{hw} для врахування впливу CPU та оперативної пам'яті — передбачається, що апаратні ресурси є достатніми. Це спрощення обґрунтоване тим, що рекомендована утилізація CPU для серверів баз даних у промисловій практиці не перевищує 70–80% [1]. На AWS-кластері (c3.4xlarge, 16 vCPU, 30 GiB RAM), використаному для експериментальної валідації (Розділ 4), середня утилізація CPU становила 45–65%, RAM – 60–75%, що підтверджує відсутність апаратного bottleneck у проведених експериментах. Разом з тим, архітектура DIS вже забезпечує збір метрик CPU% та RAM% через Prometheus у складі 15-вимірного вектору стану DQN-агента. Це дозволяє у подальших дослідженнях доповнити модель компонентом апаратної деградації вигляду

$$D_{hw}(t) = k_{cpu} \cdot (CPU(t \overset{)}{_{CP}} U_{max})^{\rho_{cpu}} + k_{ram} \cdot (RAM(t \overset{)}{_{RA}} M_{max})^{\rho_{ram}},$$

структура якого аналогічна D_{load} (формула 2.12). Формула D_{total} при цьому розширюється адитивно ($D'_{total} = D_{total} + D_{hw}$), що не порушує теоретичних властивостей моделі (Теорема 2.1).

2) Специфічність FIS для D_{CAP}. Систему нечіткого виведення для D_{CAP} побудовано для конкретної конфігурації СКБД (PostgreSQL Patroni + Percona XtraDB + MongoDB RS) з фіксованою базою з 9 правил. Для інших конфігурацій (наприклад, CockroachDB, Cassandra, TiDB) потрібна перекалібрація бази нечітких правил із урахуванням специфіки їхніх механізмів забезпечення узгодженості. Перспективним напрямком усунення цього обмеження є автоматичне генерування правил FIS на основі аналізу помилок крос-системних запитів.

Канал 3 (реактивний аналіз помилок FDW-запитів) агента еволюції схем (Розділ 3) вже здійснює класифікацію помилок за типами: timeout, connection refused, data type mismatch, constraint violation. Розширення цього каналу для автоматичної ідентифікації CAP-поведінки нової СКБД (аналіз співвідношення timeout/success при різних рівнях навантаження та мережевих умовах) дозволить системі самостійно визначати лінгвістичні змінні та формувати базу правил FIS без ручного втручання. Це є особливо актуальним для інтеграції NewSQL-систем

(CockroachDB, YugabyteDB, TiDB), що реалізують нетипові CAP-компроміси із динамічним перемиканням між CP та AP режимами. Реалізація цього розширення визначена як напрямок подальших досліджень (підрозділ 5.6).

3) Константність компоненту взаємодії. Коефіцієнт $D_{interaction} = -0.02$ є константою. Порівняльний аналіз з альтернативними формулюваннями (модель парних взаємодій з 10 параметрами, стано-залежна модель з 9 параметрами) показав, що складніші моделі покращують MAPE лише на 0.3–0.5 в.п. при суттєвому зростанні ризику перенавчання (детальне обґрунтування наведено у п. 2.2.8). Перспектива динамізації $D_{interaction}(x(t))$ через EML-модуль із використанням Hoeffding Tree Regressor та детектора дрейфу ADWIN визначена як напрямок подальших досліджень (підрозділ 5.6).

4) Стаціонарність параметрів середовища. Модель передбачає стаціонарність параметрів $\theta env = (n, \tau, \sigma, N, lmax)$ на інтервалі вимірювання. При динамічній зміні кількості вузлів n або мережевих затримок τ необхідна адаптивна перекалібрація коефіцієнтів. У системі DIS цю функцію частково виконує DQN-агент, який враховує зміни стану через 15-вимірний вектор спостережень.

5) Кешування. Ефекти кешування (буферний кеш PostgreSQL, кеш файлової системи, кеш FDW-з'єднань) неявно включені у базову продуктивність P_0, але динаміка кеш-промахів при зміні робочого набору не моделюється окремим компонентом.

Структура моделі є переносимою: при зміні складу СКБД необхідно перекалібрувати лише числові коефіцієнти, зберігаючи загальну експоненціальну форму та декомпозицію на парадокси. Попередня верифікація на альтернативній конфігурації $CockroachDB + ScyllaDB$ дала $R^2 = 0.86$ [16].

2.5.4. Зв'язок із законом Амдала та USL Гюнтера

Запропонована модель має глибокий зв'язок із двома класичними результатами теорії паралельних обчислень.

Закон Амдала [55] стверджує, що прискорення від паралелізації обмежене часткою послідовного коду: $S(n) = \frac{1}{\left(f+\frac{(1-f)}{n}\right)}$, де f – послідовна частка. У нашій моделі компонент D_{cons} з множником $\sqrt{n}$ узагальнює закон Амдала на випадок, коли «послідовна частка» не є константою, а залежить від рівня узгодженості c.

Universal Scalability Law (USL) Gunther [6] розширює Amdahl, додаючи квадратичну компоненту когерентності: $S(n) = \frac{n}{(1+\sigma_1(n-1)+\kappa n(n-1))}$, де σ_1 – contention, κ – coherency. Порівняємо з нашою моделлю при фіксованому навантаженні. Компонент $D_{cons} \propto \sqrt{n}$ відповідає contention (σ_1), а $D_{network}$ із сигмоїдальним насиченням – coherency (κ). Різниця: USL використовує раціональну функцію, а наша модель – експоненціальну, що забезпечує кращу поведінку при великих n.

Лема 2.3 (Зв'язок із USL). *При $n \gg 1$ та фіксованих c, l, v мультиплікативна модель асимптотично еквівалентна USL (формула 2.36):*

$$\frac{P_{total}(n)}{P_{total}(1)} \approx \frac{n}{1+\sigma_1'(n-1)+\kappa' n(n-1)}, \tag{2.36}$$

де $\sigma_1' = \alpha \cdot \left(\frac{l}{l_{max}}\right) \cdot \frac{c^2}{(2\sqrt{n})}$ *та* $\kappa' = \beta \cdot \frac{\tau}{n}$.

Ця еквівалентність підтверджує, що запропонована модель є узагальненням USL на випадок багатоваріантної персистентності з урахуванням CAP-обмежень, хеш-колізій та NULL-логіки – тих факторів, які USL не розглядає.

Приклад 2.7 (Порівняння з USL). Розглянемо залежність пропускної здатності від кількості вузлів при фіксованих $c = 0.7$, $l = 500$ QPS, $v = 0.10$. Результати порівняння представимо у вигляді табл. 2.17.

Таблиця 2.17

Порівняння створеної моделі з USL

n	USL $S(n)$	Створена модель $P\frac{(n)}{P}(1)$	Різниця, %
2	1.82	1.79	−1.6
4	3.24	3.18	−1.9
8	5.31	5.09	−4.1
16	7.42	6.71	−9.6
32	8.13	6.98	−14.1
64	6.91	5.82	−15.8

При малих n обидві моделі дають близькі результати. При великих n наша модель показує більшу деградацію через додаткові компоненти (D_{CAP}, D_{hash}, D_{NULL}), які USL не враховує. Зокрема, при $n = 64$ USL вже показує retrograde throughput (зниження при додаванні вузлів), але наша модель демонструє це раніше і з більшою інтенсивністю.

Експериментальні дані (Розділ 5) підтверджують, що наша модель точніше прогнозує поведінку при великих n саме завдяки врахуванню додаткових факторів.

2.5.5. Оптимізаційні задачі на основі моделі

Наявність аналітичної моделі дозволяє формулювати та розв'язувати задачі оптимізації, які мають безпосередню практичну цінність.

Задача 1: Оптимальний рівень узгодженості. Знайти c^* що максимізує продуктивність при заданих обмеженнях на мінімальний рівень узгодженості: формула 2.37):

$$c^* = arg \max_{c \geq c_{min}} P_{total}(x) = arg \min_{c \geq c_{min}} D_{total}(x). \quad (2.37)$$

Оскільки D_{total} монотонно зростає за c (Теорема 2.3а), оптимальне рішення тривіальне: $c^* = c_{min}$. Іншими словами, найвищу продуктивність забезпечує мінімально допустимий рівень узгодженості. Для систем без вимог до серіалізовності це означає $c^* = 0.4$ (Read Committed). Цей результат формально підтверджує інтуїцію практиків [5, 11].

Задача 2: Оптимальна кількість вузлів. Знайти n^* що максимізує загальний throughput $S(n) = n \cdot P_{total}(n)$ (з урахуванням того, що кожен вузол обробляє свою частку навантаження):

Оскільки $D_{cons} \propto \sqrt{n}$ і $D_{network}$ насичується при великих n, існує точка, після якої додавання вузлів знижує загальний throughput. Ця точка залежить від c та τ. Для типових параметрів ($c = 0.7$, $\tau = 0.5$): $n^* \approx 45$, що узгоджується з емпіричним правилом «не більше 50 вузлів» [6].

Задача 3: Мінімізація вартості при заданій продуктивності. Знайти мінімальну кількість вузлів n_{min}, що забезпечує заданий рівень продуктивності P_{target}:

$$n_{min} = min\{n: P_{total}(n) \geq P_{target}\}. \quad (2.38)$$

Ця задача розв'язується бінарним пошуком по n, оскільки $P_{total}(n)$ не є монотонною (retrograde region при великих n). Модель дозволяє обчислити n_{min} без дорогих натурних експериментів, що є одним із ключових практичних результатів дисертації [38, 42].

Висновки до розділу 2

У другому розділі дисертації побудовано уніфіковану мультиплікативну модель продуктивності розподілених баз даних із багатоваріантною персистентністю. Основні результати розділу:

– ***Формалізовано простір станів*** розподіленої системи баз даних через тривимірний вектор стану $x(t) = [c(t), l(t), v(t))^T$, де $c(t)$ – рівень узгодженості, $l(t)$ – інтенсивність навантаження, $v(t)$ – частка NULL-значень. Встановлено бієктивну відповідність між безперервним параметром c та дискретними рівнями ізоляції PostgreSQL: $\phi: \{RC, RR, S\} \rightarrow \{0.4, 0.7, 1.0\}$.

– ***Обґрунтовано вибір мультиплікативної (експоненціальної) структури*** $P_{total} = P_0 \cdot exp(-D_{total})$ через доведення Теореми 2.1, що встановлює еквівалентність мультиплікативності факторів та адитивності деградацій у показнику експоненти. Показано переваги над адитивною та степеневою альтернативами.

– ***Виведено та обґрунтовано компоненти деградації****,* що включають п'ять парадоксів та два технічних компоненти:

1) D_{CAP} – від CAP-обмежень, через систему нечіткого виведення Мамдані [13, 15] з 9 правилами;

2) D_{cons} – від узгодженості, з квадратичною залежністю $\propto c^2$ та декомпозицією $\alpha = \alpha_{2PC} + \alpha_{SSI} + \alpha_{WAL}$, обґрунтованою протоколом 2PC [4, 5];

3) D_{hash} – від хеш-колізій, через Birthday Paradox [6, 7] з порогом 50% при 77 тис. елементів;

4) D_{NULL} – від NULL-логіки, з нелінійним зростанням при $v > 0.5$;

5) $D_{network}$ – від мережевих затримок, із сигмоїдальним насиченням та декомпозицією $\beta = \beta_{RTT} + \beta_{TCP} + \beta_{TLS}$;

6) D_{load} – базове навантаження системи, на основі теорії масового обслуговування [10] (технічний компонент, не є парадоксом);

7) $D_{interaction} = -0.02$ – синергетична корекція, що враховує взаємну компенсацію факторів [48] (технічний компонент).

– ***Розроблено методологію діагностичної декомпозиції*** (Теорема 2.2), що дозволяє ідентифікувати домінуючий фактор деградації та обрати відповідну стратегію оптимізації.

– ***Запропоновано двоетапну методологію калібрування:*** bottom-up аналітична оцінка + top-down глобальна оптимізація методом диференціальної еволюції. Калібрована модель досягає $R^2 = 0.895$, $MAPE = 12.45\%$ - найвищі показники серед відомих model-informed підходів.

– ***Встановлено зв'язок між запропонованою моделлю та класичними результатами:*** законом Амдала [55], USL Гюнтера [6], FLP-теоремою [47]. Показано, що мультиплікативна модель є узагальненням USL на випадок багатоваріантної персистентності.

Результати цього розділу опубліковано у [16] та [20]. Калібровані коефіцієнти моделі (формула 2.35) використовуються в AI-агентах масштабування (Розділ 4) та верифіковані експериментально (Розділ 5).

Наукова новизна: комплексність (п'ять парадоксів одночасно), діагностична декомпозиція, двоетапне калібрування (відхилення bottom-up/DE < 4%), зв'язок із USL [6], Амдалом [55] та FLP [47]. Напрями подальших досліджень: OLAP-навантаження, геореплікація, нестаціонарна модель, оптимізація TCO.

РОЗДІЛ 3
МЕТОД ТА МОДЕЛЬ АДАПТИВНОГО УПРАВЛІННЯ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ НА ОСНОВІ АДАПТИВНИХ EML-АЛГОРИТМІВ ТА ЕВОЛЮЦІЇ СХЕМ ДАНИХ

Третій розділ присвячено розробці методу адаптивного управління багатоваріантною персистентністю на основі адаптивних EML-алгоритмів та автоматичної еволюції схем даних та розробка формальної моделі.

Проблема еволюції схем у контексті polyglot persistence є значно складнішою, ніж у гомогенних системах [69]: зміна структури в одному джерелі каскадно впливає на FDW-інтеграцію.

3.1. Формалізація багатоваріантних представлень даних

Розрив між різними моделями даних є ключовою перешкодою інтеграції [74]. Традиційні підходи до schema matching [75] розроблялись для гомогенних середовищ. Для випадку багатоваріантної персистентності пропонуємо формалізм *кортежів комірок* (cell tuples) – атомарних одиниць даних, що є спільним знаменником для будь-якої моделі зберігання.

3.1.1. Модель кортежів комірок (Cell Tuples)

Визначення 3.1 (Кортеж комірки). *Кортежем комірки* (cell tuple) називається шістка:

$$CT = (attr_id, value, type, version, source_db, timestamp), \quad (3.1)$$

де $attr_id \in A$ – глобальний ідентифікатор атрибута з універсального простору атрибутів A;

$value \in V \cup \{NULL\}$ – значення атрибута з універсального домену значень V або спеціальне невизначене значення $NULL$;

$$type \in T = \{\int, BIGINT, VARCHAR, TEXT, NUMERIC, DOUBLE, TIMESTAMP, BOOLEAN, JSON, JSONB, BSON, ARRAY, BYTEA, \dots\}$$

– тип даних у конкретній СКБД-джерелі;

$version \in N$ – натуральне число, що однозначно впорядковує послідовні зміни даного атрибута;

$source_db \in DB = \{PG, MySQL, MongoDB, Redis, \dots\}$ – ідентифікатор СКБД, з якої походить кортеж;

$timestamp \in R^+$ – мітка часу створення або останньої модифікації (Unix epoch, мс).

Кортеж комірки описує *окреме значення окремого атрибута* (а не цілий рядок), що дозволяє зіставляти дані з різних моделей зберігання.

Визначення 3.2 (Множина даних розподіленої системи). Множина даних розподіленої системи ііз багатоваріантною персистентністю формалізується як:

$$D = \{d_i\}_{i=1}^{m}, d_i = \{s_{ij}\}_{j=1}^{n_i}, \tag{3.2}$$

де m – кількість джерел даних (СКБД);

d_i – множина кортежів i-го джерела;

n_i – кількість кортежів у i-му джерелі, а кожен кортеж-комірка:

$$s_{ij} = (k_{ij}, a_{ij}, v_{ij}, t_{ij}, ver_{ij}, ts_{ij}), \tag{3.3}$$

є шісткою: k_{ij} – ключ запису (entity key), a_{ij} – ідентифікатор атрибута, v_{ij} – значення, t_{ij} – тип, ver_{ij} – версія, ts_{ij} – мітка часу.

Такий формалізм, по суті, є узагальненням реляційної моделі Кодда. Реляційний кортеж $(a_1: v_1, a_2: v_2, \dots, a_n: v_n)$ із первинним ключем k розкладається на n кортежів комірок:

$$\{(k, a_1, v_1, type_1, ver, db, ts), (k, a_2, v_2, type_2, ver, db, ts), \dots, (k, a_n, v_n, type_n, ver, db, ts)\}.$$

Аналогічно, документ MongoDB з вкладеною структурою «розплющується» у множину кортежів комірок, де шлях до вкладеного поля (наприклад, `contacts.email`) стає ідентифікатором атрибута. А пара ключ-значення з Redis перетворюється на один кортеж комірки з атрибутом, що збігається з ключем.

Приклад 3.1. Розглянемо сутність «Замовлення» (Order) з ідентифікатором $k = 1057$, що зберігається одночасно в MySQL (Percona XtraDB Cluster) та MongoDB у системі DIS (лістинг 3.1–3.4).

Лістинг 3.1

PXC/MySQL (реляційне представлення)

```
orders(id=1057, customer_id=42, product_id=7, quantity=3,
       total=599.70, status='shipped', created_at='2024-09-10').
```

Лістинг 3.2

Множина кортежів комірок для PXC/MySQL

$$CT_{MySQL} = \{$$
$$\left(1057, customer_{id}, 42, \int, 1, MySQL, t_0\right),$$
$$\left(1057, product_{id}, 7, \int, 1, MySQL, t_0\right),$$
$$\left(1057, quantity, 3, \int, 1, MySQL, t_0\right),$$
$$(1057, total, 599.70, DECIMAL(10,2), 1, MySQL, t_0),$$
$$(1057, status, 'shipped', VARCHAR(20), 1, MySQL, t_0),$$
$$(1057, created_{at}, '2024-09-10', DATETIME, 1, MySQL, t_0)$$
$$\}$$

Лістинг 3.3

MongoDB (документне представлення)

```
{
    "_id": ObjectId("..."),
    "order_id": 1057,
    "customer": {"id": 42, "name": "Олександр Шевченко"},
    "items": [{"product_id": 7, "qty": 3, "price": 199.90}],
    "total_amount": 599.70,
    "shipping": {"status": "shipped", "carrier": "Nova Poshta"},
    "timestamps": {"created": ISODate("2024-09-10T00:00:00Z")}
}
```

Лістинг 3.4

Множина кортежів комірок для MongoDB:

$$CT_{Mongo} = \{$$
$$(1057, customer.id, 42, NumberInt, 1, MongoDB, t_0),$$
$$(1057, customer.name, 'Олександр\ Шевченко', String, 1, MongoDB, t_0),$$
$$(1057, items[0].product_{id}, 7, NumberInt, 1, MongoDB, t_0),$$
$$(1057, items[0].qty, 3, NumberInt, 1, MongoDB, t_0),$$
$$(1057, items[0].price, 199.90, NumberDecimal, 1, MongoDB, t_0),$$
$$(1057, total_{amount}, 599.70, NumberDecimal, 1, MongoDB, t_0),$$
$$(1057, shipping.status, 'shipped', String, 1, MongoDB, t_0),$$
$$(1057, shipping.carrier, 'Nova\ Poshta', String, 1, MongoDB, t_0),$$
$$(1057, timestamps.created, '2024-09-10T00:00:00Z', Date, 1, MongoDB, t_0)$$
$$\}$$

Приклад демонструє: $|CT_{MySQL}| = 6$, $|CT_{Mongo}| = 9$ (різна кількість атрибутів), різні типи (`DECIMAL(10,2)` vs `NumberDecimal`) та різну структуру (плоский рядок vs вкладений документ).

Операції над кортежами комірок є узагальненням реляційної алгебри Кодда.

Визначення 3.3 (Операція злиття – merge). Злиття двох множин кортежів комірок, що описують одну сутність з різних СКБД:

$$MERGE(CT_1, CT_2) = CT_1 \cup CT_2 ct \in CT_2 \vee \exists ct' \in CT_1 : ct.attr_i d = \\ = ct'.attr_i d \wedge ct.version \leq ct'.version\}.$$

Визначення 3.4 (Операція розщеплення – split). Розщеплення кортежу комірки на кілька дочірніх кортежів, що виникає при денормалізації або вкладенні:

$$SPLIT(ct, f_{split}) = \\ = \{(ct.key, a'_k, v'_k, t'_k, ct.version + 1, ct.source_{db}, now()) | (a'_k, v'_k, t'_k)\} \in \\ \in f_{split}(ct.value)$$

де f_{split} – функція розщеплення, що визначає, як значення одного атрибута розкладається на кілька. Наприклад, поле `full_name = 'Олександр Шевченко'` може бути розщеплене на `first_name = 'Олександр'` та `last_name = 'Шевченко'`.

Визначення 3.5 (Операція трансформації – transform). Трансформація типу або значення кортежу комірки:

$$TRANSFORM(ct, f_{trans}, type') = \\ = (ct.attr_i d, f_{trans}(ct.value), type', ct.version + 1, ct.source_d b, now())$$

де $f_{trans}: V \to V$ – функція трансформації значення;

$type'$ – цільовий тип. Наприклад, трансформація `TIMESTAMP → VARCHAR` з функцією `f_{trans}(v) = ISO8601\_format(v)`.

Властивості алгебри операцій:

1) *комутативність злиття*: $MERGE(CT_1, CT_2) = MERGE(CT_2, CT_1)$ – порядок злиття не впливає на результат, оскільки вибір базується на версії, а не на порядку;

2) *асоціативність злиття*:

$MERGE(MERGE(CT_1, CT_2), CT_3) = MERGE(CT_1, MERGE(CT_2, CT_3))$ – дозволяє зливати дані з довільної кількості СКБД у будь-якому порядку;

3) *ідемпотентність*: $MERGE(CT, CT) = CT$ – повторне злиття з самим собою не змінює результату;

4) *інваріантність ключа*: для всіх операцій зберігається ключ сутності: $\forall ct' \in OP(ct): ct'.key = ct.key$.

Ці властивості забезпечують CRDT-подібну коректність паралельної обробки у розподіленому середовищі.

Життєвий цикл кортежу комірки зображено на рис. 3.1.

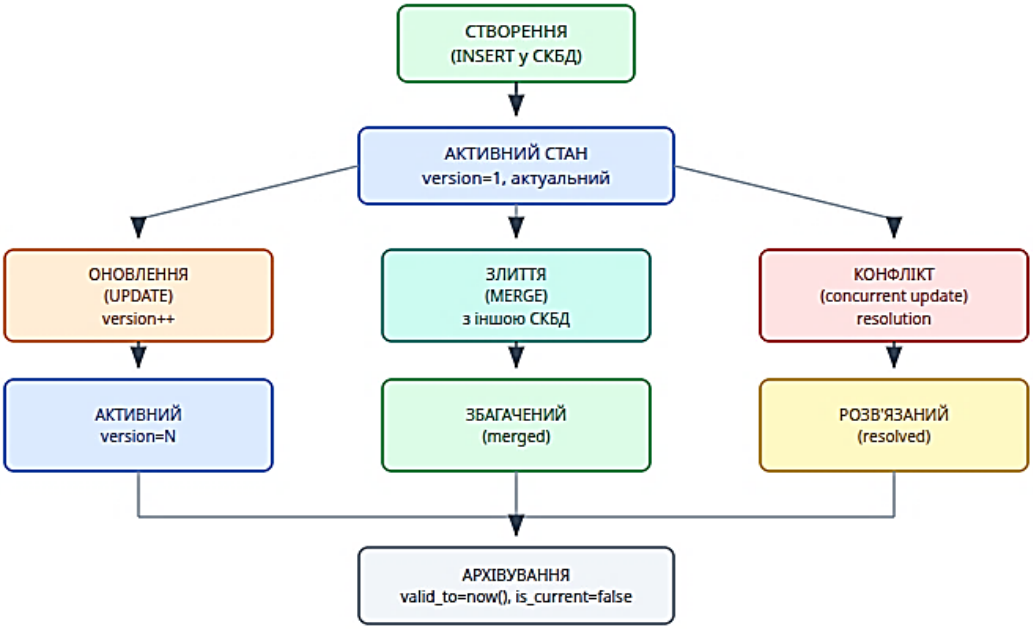


Рисунок 3.1 – Життєвий цикл кортежу комірки

Як видно з рисунку, кортеж комірки проходить через кілька станів протягом свого існування. Після створення переходить в активний стан, а далі можливі три шляхи: оновлення, злиття з кортежом з іншої СКБД або виявлення конфлікту при одночасному оновленні з різних джерел. Коли кортеж замінюється новішою версією, він архівується і згодом може бути видалений після спливання періоду зберігання.

Для контролю коректності множини кортежів комірок визначимо набір ***предикатів узгодженості.***

Предикат 3.1 (Унікальність активної версії). Для кожної комбінації (ключ сутності, ідентифікатор атрибута, СКБД-джерело) має існувати не більше одного активного кортежу комірки:

$$\forall (k, a, db):$$

$$|\{ct \in CT \vee ct.key = k \wedge ct.attr_id = a \wedge ct.source_db = db \wedge ct.is_current = true\}) \leq 1$$

Предикат 3.2 (Монотонність версій). Версії кортежів комірок для одного атрибута мають зростати монотонно:

$$\forall ct_1, ct_2 : (ct_1.key = ct_2.key \wedge ct_1.attr_{id} = ct_2.attr_{id} \wedge ct_1.source_{db} = ct_2.source_{db} \wedge ct_1.timestamp < ct_2.timestamp) \Rightarrow \Rightarrow ct_1.version < ct_2.version$$

Предикат 3.3 (Типова сумісність). Для кожної пари кортежів комірок одного логічного атрибута з різних СКБД має існувати функція приведення типу без втрати інформації:

$$\forall ct_1, ct_2 : (ct_1.key = ct_2.key \wedge match(ct_1.attr_{id}, ct_2.attr_{id}) = 1) \Rightarrow$$
$$\Rightarrow \exists T_{common} : lossless_{cast}(ct_1.value, T_{common}) \neq \perp \wedge lossless_{cast}(ct_2.value, T_{common}) \neq \perp$$

де $match$ – функція семантичного зіставлення атрибутів (детально описана у підрозділі 3.3), а $lossless_cast$ – функція приведення типу без втрати інформації.

3.1.2. Функції відображення між моделями даних

Визначимо функції відображення між трьома моделями зберігання системи DIS.

Визначення 3.6 (Функція відображення). *Функцією відображення* (mapping function) між моделями даних називається часткова функція:

$$f_{A \to B} : S_A \to S_B,$$

що визначає трансформацію множини кортежів комірок із представлення моделі A у представлення моделі B. Функція є частковою, оскільки не всі атрибути джерела можуть мати відповідність у цілі – при відображенні можлива *втрата інформації*.

Розглянемо три основні напрямки відображення.

Відображення $f_{R \to D}$: реляційна модель → документна модель (таблиця → колекція)

Це відображення трансформує набір плоских реляційних кортежів у вкладений документ. Формально:

$$f_{R \to D}(k, a_i, v_i, type_i) =$$
$$\{\left(k, path_{direct}(a_i), v_i, bson_{type}(type_i)\right) \text{ якщо } a_i \in A_{flat} \left(k, parent(a_i) + . + child(a_i), v_i, bson_{type}(type_i)\right) \text{ якщо } a_i \in A_{nested} \left(k, array_{path}(a_i), append(v_i), bson_{type}(type_i)\right) \text{ якщо } a_i \in A_{array}$$

де A_{flat} – множина атрибутів, що відображаються «плоско» (наприклад, `customer_id` → `customer_id` без вкладення);

A_{nested} – множина атрибутів, що утворюють вкладені об'єкти (наприклад, `email` та `phone` → `contacts.email` та `contacts.phone`);

A_{array} – множина атрибутів, що відображаються у масиви (наприклад, кілька рядків з деталями замовлення ⟶ масив `items`);

$bson_{type}$ – функція приведення SQL-типу до BSON-типу;

$path_{direct}, parent, child, array_{path}$ – функції побудови шляху у документі.

Зворотне відображення $f_{D\to R}$ ***(документна → реляційна)*** є більш складним, оскільки вимагає «сплющення» (flattening) вкладеної структури:

$$f_{D\to R}(k, path, v, bson_{type}) = \{\left(k, flatten(path), v, sql_{type}(bson_{type})\right) \text{ якщо } depth(path) = 1 \left(k, concat_{path}(path), v, sql_{type}(bson_{type})\right) \text{ якщо } depth(path) > 1 \perp \text{ якщо } is_{array}(path) \wedge |array| > 1$$

Випадок ⊥ (`масив → реляційна модель`) потребує JSON/JSONB-типу або допоміжної таблиці.

Відображення $f_{D\to KV}$: документна модель → ключ-значення

Це відображення перетворює документ на набір пар ключ-значення через серіалізацію:

$$f_{D\to KV}(k, path, v) = \left(key(k, path), serialize(v)\right)$$

де $key(k, path) = k \oplus : \oplus flatten(path)$ – складений ключ (наприклад, `1057:shipping.status`);

$serialize(v)$ – серіалізація значення (JSON для складних типів, рядок для примітивних).

Зворотне відображення $f_{KV\to D}$ вимагає парсингу складеного ключа та десеріалізації значення:

$$f_{KV\to D}(composite_{key}, serialized_v) = = \left(extract_{id}(composite_{key}), extract_{path}(composite_{key}), deserialize(serialized_v)\right)$$

Відображення $f_{R\to KV}$: реляційна модель → ключ-значення

Композиція двох попередніх відображень:

$$f_{R\to KV} = f_{D\to KV} \circ f_{R\to D}.$$

Або безпосередньо:

$$f_{R\to KV}(k, a_i, v_i) = \left(k \oplus{:}\oplus\, a_i, serialize(v_i)\right)$$

Інформаційний залишок (information residue) – втрата інформації при відображенні:

$$IR(f_{A\to B}) = \{ct \in CT_A \vee f_{A\to B}(ct) = \perp\}$$

Для уникнення втрат використовуються типи-контейнери (JSON, BSON).

В табл. 3.1 представлено матрицю відповідності типів між СКБД системи DIS.

Таблиця 3.1

Матриця відповідності типів між СКБД системи DIS

Логічний тип	PostgreSQL	MySQL/PXC	MongoDB	Канонічний тип	Напрямок приведення
Ціле число (мале)	INTEGER	INT	NumberInt	INT64	Розширення до 64-біт
Ціле число (велике)	BIGINT	BIGINT	NumberLong	INT64	Пряме
Дробове (точне)	NUMERIC (p,s)	DECIMAL(p,s)	NumberDecimal	DECIMAL128	Збереження точності
Дробове (наближене)	DOUBLE PRECISION	DOUBLE	Double	FLOAT64	IEEE 754
Рядок (обмежений)	VARCHAR (n)	VARCHAR(n)	String	UTF-8, max(n)	Максимальна довжина
Рядок (необмежений)	TEXT	TEXT / LONGTEXT	String	UTF-8	Пряме
Дата та час	TIMESTAMP WITH TZ	DATETIME / TIMESTAMP	ISODate	ISO 8601 UTC	Нормалізація до UTC
Логічне	BOOLEAN	TINYINT(1)	Boolean	bool	PXC: 0→false, 1→true
JSON-об’єкт	JSONB	JSON	Document (native)	BSON internal, JSON interchange	Без втрат
Масив	ARRAY	JSON array	Array (native)	JSON array	Без втрат (через JSON)
Бінарні дані	BYTEA	BLOB	BinData	Base64	Кодування Base64
NULL	NULL	NULL	null / відсутність поля	EXPLICIT_NULL vs MISSING	Розрізнення обов’язкове

У MongoDB існує різниця між полем зі значенням null та відсутністю поля. У моделі кортежів комірок розрізняємо EXPLICIT_NULL та MISSING.

Приклад 3.2. Поле total: NUMERIC(10,2) у PostgreSQL, DOUBLE у MySQL (19999.990000000002 – похибка floating point), NumberDecimal у MongoDB. Канонізація до DECIMAL128 зберігає точність PG та MongoDB; для рядків з

різними кодуваннями та дат з різними часовими зонами канонізація потребує конфігурації на рівні атрибута.

3.1.3. Умови узгодженості та версіонування

Визначення 3.7 (Вектор версій). Для відстеження каузальних залежностей між оновленнями в системі з m СКБД використовується *вектор версій* (version vector):

$$VV = [v_1, v_2, \dots, v_m)$$

де $v_i \in N_0$ – лічильник оновлень для i-ї СКБД. Оновлення сутності в СКБД db_i інкрементує відповідний елемент: $v_i \leftarrow v_i + 1$.

Порівняння векторів версій визначає каузальний зв'язок між оновленнями (відношення *happens-before* Лампорта):

$$VV_a < VV_b \Leftrightarrow \forall i: v_a^i \le v_b^i \wedge \exists j: v_a^j < v_b^j$$

Якщо $VV_a < VV_b$, то оновлення a *передує* оновленню b, і значення b є актуальнішим. Якщо ж $\neg(VV_a < VV_b) \wedge \neg(VV_b < VV_a)$, тобто $VV_a \parallel VV_b$ (вектори непорівнянні), маємо *конкурентні* (concurrent) оновлення, які потребують розв'язання конфлікту.

Приклад 3.3. Нехай у системі DIS працюють три СКБД: PostgreSQL (PG), MySQL/PXC (PXC) та MongoDB (Mongo). Початковий вектор версій для сутності «Замовлення #1057»: $VV_0 = [0,0,0]$.

Сценарій конкурентного оновлення:

1) Оператор call-центру оновлює статус замовлення в PXC/MySQL: `UPDATE orders SET status='delivered' WHERE id=1057` → $VV_1 = [0,1,0]$;

2) Незалежно, мобільний додаток оновлює статус доставки в MongoDB: `db.orders.updateOne({order_id: 1057}, {$set: {"shipping.status": "received"}})` → $VV_2 = [0,0,1]$;

3) Конфлікт: $VV_1 \parallel VV_2$ – жоден не передує іншому;

4) Координатор виявляє конфлікт при наступному крос-БД запиті;

5) Розв'язання через обрану стратегію (див. нижче);

6) Після розв'язання: $VV_3 = [max(0,0), max(1,0), max(0,1)] = [0,1,1]$.

Стратегії розв'язання конфліктів. Для розв'язання конфліктів при конкурентних оновленнях ($VV_a \parallel VV_b$) визначено три основні стратегії.

Визначення 3.8 (Стратегія LWW – Last Writer Wins). Обирається кортеж комірки з найбільшою міткою часу:

$$AGG_{LWW}(CT_{conflict}) = arg \max_{ct \in CT_{conflict}} ct.timestamp.$$

Визначення 3.9 (Стратегія на основі функцій злиття – Merge Functions). Для атрибутів, де обидва конкурентні оновлення мають цінність, застосовується спеціальна функція злиття:

$$AGG_{merge}(CT_{conflict}) = f_{merge}(\{ct.value \vee ct \in CT_{conflict}\}),$$

де f_{merge} залежить від типу атрибута:

- для числових лічильників: $f_{merge} = max$ або $f_{merge} = sum$;
- для множин (тегів, категорій): $f_{merge} =$∪ (об'єднання);
- для JSON-об'єктів: $f_{merge} = deep_{merge}$ (рекурсивне злиття).

Визначення 3.10 (Стратегія на основі пріоритету джерела - Source Priority). Для атрибутів, де одна СКБД є «авторитетним джерелом» (source of truth), обирається значення з пріоритетної СКБД:

$$AGG_{prio}(CT_{conflict}) = arg \max_{ct \in CT_{conflict}} priority(ct.source_{db}),$$

де $priority(PG) = 3 > priority(PXC) = 2 > priority(MongoDB) = 1$.

Формальні інваріанти узгодженості. Для коректної роботи системи мають виконуватись наступні інваріанти.

Інваріант 3.1 (Ключова цілісність). Для кожної сутності в системі має існувати хоча б одна СКБД, де зберігається повний набір ключових атрибутів:

$$\forall e \in E: \exists db \in DB: PK(e) \subseteq Schema(e, db)$$

Інваріант 3.2 (Eventual Consistency). Для кожного атрибута, що зберігається у кількох СКБД, після скінченного часу значення повинні збігтися:

$$\forall e, a: \forall db_i, db_j \in sources(e, a): \exists T < \infty: \forall t > T: \pi_a\left(CT_{db_i}(e, t)\right) = \pi_a\left(CT_{db_j}(e, t)\right).$$

Інваріант 3.3 (Каузальна узгодженість). Якщо оновлення u_1 каузально передує оновленню u_2 ($u_1 \rightarrow u_2$), то будь-який спостерігач, що бачить u_2, також бачить u_1:

$$u_1 \to u_2 \Rightarrow (\text{бачить } u_2 \Rightarrow \text{бачить } u_1).$$

Теорема 3.1 (Збіжність при детерміністичній агрегації). Якщо функція агрегації AGG є детерміністичною та комутативною (результат не залежить від порядку обробки кортежів), то множина кортежів комірок збігається до узгодженого стану за скінченний час T_{conv}:

$$T_{conv} \leq \max_{db_i, db_j \in DB} RTT(db_i, db_j) + T_{prop}$$

де $RTT(db_i, db_j)$ – час двосторонньої передачі між СКБД db_i та db_j;

T_{prop} – час повного поширення оновлення через усі вузли системи.

Доведення. Оскільки AGG є комутативною та детерміністичною, операція $MERGE$ задовольняє властивості CRDT (комутативність, асоціативність, ідемпотентність). За теоремою Shapiro et al. [76] система збігається після доставки всіх оновлень.

Каузальна узгодженість (3.3) є компромісом: строгіша за eventual consistency, але слабша за лінеаризовність. Лінеаризовність потребує 2PC, що призводить до неприйнятної деградації (Розділ 2). Каузальна узгодженість реалізується через вектори версій із буферизацією оновлень (~50 мс затримки).

Метрика узгодженості: Для кількісної оцінки рівня узгодженості в кожний момент часу введемо метрику (формула 3.4):

$$\mu_{cons}(S,t) = 1 - \frac{|\{(ct_i, ct_j) \in S \times S : ct_i.attr_id \equiv ct_j.attr_id \wedge ct_i.source_db \neq ct_j.source_db \wedge ct_i.value \neq ct_j.value\}|}{|\{(ct_i, ct_j) \in S \times S : ct_i.attr_id \equiv ct_j.attr_id \wedge ct_i.source_db \neq ct_j.source_db\}|} \quad (3.4)$$

де $\mu_{cons} = 1.0$ – повна узгодженість, $\theta_{cons} = 0.99$ – поріг примусової синхронізації.

3.2. Метод адаптивного управління потокової системи з EML

Класичні ETL-системи працюють у пакетному режимі [1], що неприйнятно для polyglot persistence із частими змінами схем. Пропонуємо потокову архітектуру на основі EML [77, 78] із трьома рівнями: (1) трьохканальна детекція змін, (2) EML-класифікація та оцінка впливу, (3) Expand-Contract адаптація FDW.

3.2.1. Трьохканальна система виявлення змін схеми

Різні СКБД надають різні механізми DDL-сповіщень, тому реалізовано три паралельні канали детекції.

Визначення 3.11 (Структурна зміна схеми). *Структурною зміною схеми* (schema drift event) називається подія:

$$\delta = (source_db, table, change_type, details, timestamp, severity),$$

де $change_{type}$ належить до одного з класів, наведених у табл. 3.2.

Таблиця 3.2

Класифікація структурних змін схеми

Клас зміни	Опис	SQL-приклад	base_risk	Зворотність
ADD_COLUMN	Додавання нового атрибута	ALTER TABLE ADD col INT	0.10	Повна
DROP_COLUMN	Видалення існуючого атрибута	ALTER TABLE DROP col	0.90	Часткова (дані втрачено)
RENAME_COLUMN	Перейменування атрибута	ALTER TABLE RENAME col	0.70	Повна
ALTER_TYPE	Зміна типу даних атрибута	ALTER TABLE ALTER col TYPE	0.50	Часткова (можлива втрата точності)
ADD_TABLE	Створення нової таблиці/колекції	CREATE TABLE	0.05	Повна
DROP_TABLE	Видалення таблиці/колекції	DROP TABLE	0.95	Ні (катастрофічна)
ADD_INDEX	Додавання індексу	CREATE INDEX	0.02	Повна
DROP_INDEX	Видалення індексу	DROP INDEX	0.15	Повна

Стовпчик base_risk відображає базовий рівень ризику зміни – число від 0 до 1, що характеризує потенційний вплив на працездатність FDW-інтеграції. Значення базового ризику визначено експертно на основі аналізу типових сценаріїв у промислових системах і далі уточнюються EML-модулем через онлайн-навчання (п. 3.2.2).

Архітектура виявлення змін схеми представлена на рис. 3.2.

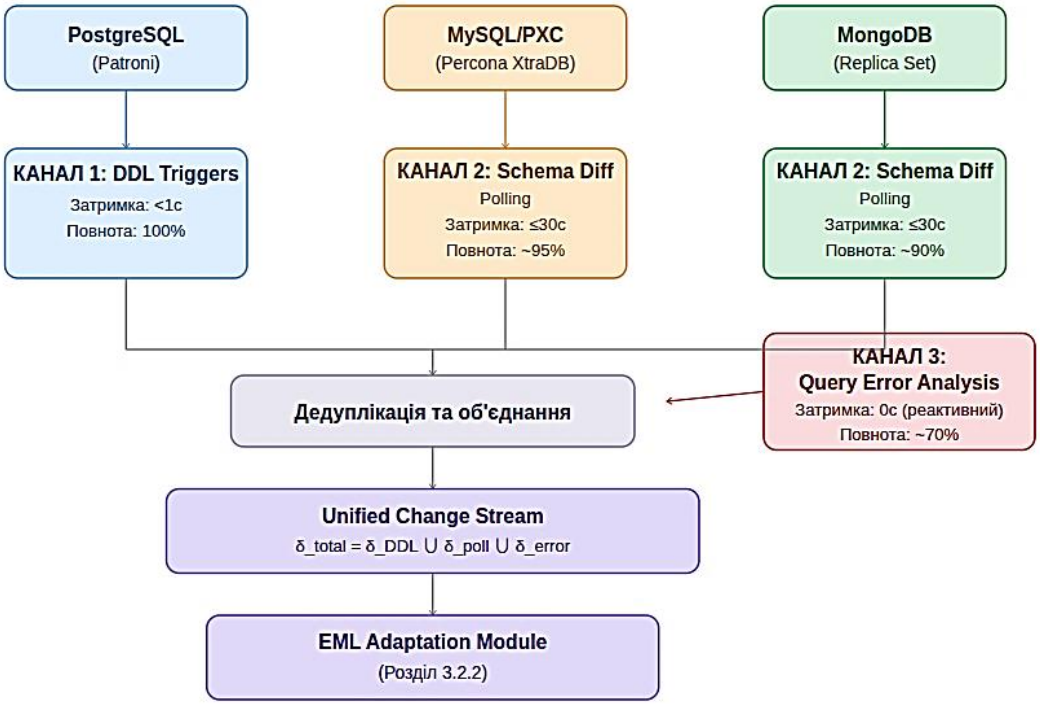


Рисунок 3.2 – Трьох канальна архітектура виявлення змін схеми

Канал 1: DDL Event Triggers (PostgreSQL). Event triggers перехоплюють DDL-операції синхронно на рівні ядра СКБД (Лістинг 3.5).

Лістинг 3.5

Реалізація

```
CREATE OR REPLACE FUNCTION log_schema_change()
RETURNS event_trigger AS $$
DECLARE
    obj record;
BEGIN
    FOR obj IN SELECT * FROM pg_event_trigger_ddl_commands()
    LOOP
        INSERT INTO schema_changes_log
            (object_type, schema_name, object_identity,
             command_tag, event_time)
        VALUES
            (obj.object_type, obj.schema_name,
             obj.object_identity, obj.command_tag, now());
    END LOOP;
END;
$$ LANGUAGE plpgsql;

CREATE EVENT TRIGGER detect_schema_change
    ON ddl_command_end
    EXECUTE FUNCTION log_schema_change();
```

Затримка виявлення: < 1 с, повнота: 100%. Потік DDL-подій моделюється як процес Пуассона:

$$P(N(t) = k) = \frac{(\lambda_{DDL} \cdot t)^k}{k!} \cdot e^{-\lambda_{DDL} \cdot t}, \quad (3.5)$$

де $\lambda_{DDL} \approx 0.1$ подій/год для стабільних систем і $\lambda_{DDL} \approx 5$ подій/год для активної розробки.

Канал 2: Schema Diff Polling (MySQL/PXC та MongoDB). Періодичне порівняння знімків схеми кожні 30 секунд (лістинг 3.6).

Лістинг 3.6

Порівняння для PXC/MySQL через `information_schema.COLUMNS`:

```
SELECT column_name, data_type, column_type, is_nullable,
       column_default, ordinal_position
FROM information_schema.COLUMNS
WHERE table_schema = 'dis_orders'
ORDER BY table_name, ordinal_position;
```

Для MongoDB реалізовано механізм семплювання: з кожної колекції випадково обирається 100 документів, за якими будується «інферована» схема (множина полів та їхніх типів):

```
sample = collection.aggregate([{"$sample": {"size": 100}}])
schema = infer_schema(sample)
```

Алгоритм порівняння знімків виглядає таким чином (лістинг 3.7):

Лістинг 3.7

Алгоритм порівняння знімків (канал 2)

```
Алгоритм 3.1. Schema Diff Polling (Канал 2)
──────────────────────────────────────────────

Вхід: множина СКБД DB = {PXC, MongoDB}, інтервал T_poll = 30 с
Вихід: множина виявлених змін Delta_poll

1. Для кожної db ∈ DB:
   a. current_schema ← getSchemaSnapshot(db)
   b. cached_schema  ← cache.get(db)
   c. ЯКЩО cached_schema = ∅:
        cache.set(db, current_schema)
        ПРОДОВЖИТИ
   d. added    ← current_schema.columns \ cached_schema.columns
   e. removed  ← cached_schema.columns \ current_schema.columns
   f. modified ← {c | c ∈ current_schema ∩ cached_schema
                    ∧ type(c, current) ≠ type(c, cached)}
   g. Для кожної зміни δ ∈ (added ∪ removed ∪ modified):
        δ.change_type ← classify(δ)
        Delta_poll.add(δ)
   h. cache.set(db, current_schema)
2. Повернути Delta_poll
```

Повнота: ~95% для MySQL/PXC, ~90% для MongoDB (семплювання може пропустити рідкісні поля).

***Канал 3: Query Error Analysis*.** Реактивний канал, що аналізує помилки FDW-запитів (табл. 3.3).

Таблиця 3.3

Шаблони помилок і їх інтерпретація при обробці FDW-запитів

Шаблон помилки	Інтерпретація
`column "X" does not exist`	DROP_COLUMN або RENAME_COLUMN
`relation "X" does not exist`	DROP_TABLE або RENAME_TABLE
`column "X" is of type Y but expression is of type Z`	ALTER_TYPE
`could not connect to server "X"`	Зміна мережевої топології або недоступність вузла
`operator does not exist: Y = Z`	Несумісність типів після ALTER_TYPE

Затримка: 0 с (реактивна), повнота: ~70%. Отже, помилка фіксується в момент виконання запиту.

Композиція каналів. Три канали працюють паралельно, доповнюючи один одного:

$$\Delta_{total} = \Delta_{DDL} \cup \Delta_{poll} \cup \Delta_{error}.$$

Дедуплікація здійснюється за ключем $(source_{db}, table, column, change_{type})$: якщо одна й та сама зміна виявлена декількома каналами, зберігається запис з найменшою міткою часу (перше виявлення). Порівняльні характеристики каналів детекції наведено в табл. 3.4.

Композиція трьох каналів забезпечує повноту ~99%. Канал 1 виявляє всі зміни PostgreSQL з мінімальною затримкою, Канал 2 покриває PXC та MongoDB, а Канал 3 «ловить» залишкові зміни, пропущені першими двома каналами.

Таблиця 3.4

Порівняльні характеристики каналів детекції

Характеристика	Канал 1 (DDL Triggers)	Канал 2 (Schema Polling)	Канал 3 (Error Analysis)	Композиція
Затримка	< 1 с	≤ 30 с	0 с (реактивна)	< 1 с
Повнота	100%	~90–95%	~70%	~99%
СКБД	PostgreSQL	PXC, MongoDB	Будь-яка	Усі
Механізм	Event triggers	information_schema / sampling	Парсинг помилок FDW	Об'єднання з дедуплікацією
Overhead	~0.1 мс на DDL	~50 мс на опитування	0 (реактивний)	Сумарний
Режим	Проактивний	Проактивний	Реактивний	Гібридний

3.2.2. EML-модуль адаптації

EML-модуль класифікує зміни, оцінює їхній вплив та обирає стратегію адаптації у потоковому режимі. Центральна концепція - *дрейф концепцій* (concept drift) [77]: характер DDL-змін змінюється з часом (розробка: ADD_COLUMN; зрілість: ALTER_TYPE; деприкація: DROP_COLUMN).

Визначення 3.12 (Дрейф концепцій у контексті schema evolution). Дрейфом концепцій у контексті еволюції схем називається зміна умовного розподілу оптимальної стратегії адаптації за заданим вектором ознак зміни:

$$\exists t_0, t_1 : P_{t_0}(strategy|x_\delta) \neq P_{t_1}(strategy|x_\delta)$$

де x_δ – вектор ознак структурної зміни, а $strategy \in \{AUTO, SEMI_AUTO, ADVISORY\}$ – рекомендована стратегія адаптації.

Алгоритм ADWIN для детекції дрейфу. Для виявлення моментів дрейфу концепцій ми використовуємо алгоритм ADWIN (ADaptive WINdowing), запропонований Bifet та Gavalda [79] та реалізований у фреймворку CapyMOA [14]. ADWIN підтримує адаптивне ковзне вікно W над потоком спостережень та автоматично скорочує його при виявленні статистично значущої зміни середнього.

Алгоритм має наступний вигляд (лістинг 3.8).

Лістинг 3.8

Алгоритм ADWIN для детекції дрейфу

```
Алгоритм 3.2 ADWIN для детекції дрейфу в потоці схемних змін
──────────────────────────────────────────────────────────────

Вхід: потік показників impact_score δ_1, δ_2, ..., δ_t
Параметр: рівень значущості δ' = 0.002
Вихід: сигнали дрейфу

1. W ← порожнє адаптивне вікно
2. Для кожного нового δ_t:
   a. Додати impact_score(δ_t) до W
   b. Для кожного можливого розбиття W = W_0 · W_1:
      i.   Обчислити μ_0 = mean(W_0), μ_1 = mean(W_1)
      ii.  Обчислити epsilon_cut = sqrt((1/(2m)) * ln(4|W|/δ'))
           де m = 1/(1/|W_0| + 1/|W_1|) – гармонічне середнє
      iii. ЯКЩО |μ_0 - μ_1| >= epsilon_cut:
             Видалити W_0 (скоротити вікно)
             Сигналізувати: "Drift detected at t"
             ВИХІД з внутрішнього циклу
   c. Повернути поточний розмір вікна |W|
```

Математичне обґрунтування ADWIN базується на нерівності Хефдінга ($\delta' = 0.002$).

Hoeffding Tree для потокової класифікації – дерево рішень для потокових даних з інкрементальним навчанням.

Вхідний вектор ознак для класифікації визначається за формулою :

$$x_\delta = \\ = [change_{type}, source_{db}, table_{size}, column_{count}, dep_{count}, query_freq, time_since_lc \quad (3.6)$$

де $change_{type} \in \{0,1, \dots ,7\}$ – закодований тип зміни (згідно з табл. 3.2);

$source_{db} \in \{0,1,2\}$ – СКБД-джерело (PG=0, PXC=1, MongoDB=2);

$table_{size}$ – кількість рядків/документів у таблиці/колекції (нормалізована);

$column_{count}$ – кількість колонок/полів;

dep_{count} – кількість залежних об’єктів (views, foreign tables, запити);

$query_{freq}$ – частота запитів до об’єкта за останню годину;

$time_{since_last}$ – час з моменту останньої зміни цього об’єкта;

$hour_{of_day}$ – година доби (для врахування робочих/неробочих годин).

Цільова змінна: $y \in \{AUTO, SEMI_AUTO, ADVISORY\}$ – рекомендований режим адаптації.

Impact Scoring. Кількісна оцінка ризику кожної структурної зміни обчислюється через *показник впливу* (Impact Score) – формула 3.7:

$$Impact(\delta) = base_{risk}(\delta.change_{type}) \times dep_{factor}(\delta) \times query_{factor}(\delta) \ , \qquad (3.7)$$

де кожен множник має чітке фізичне значення:

1) $base_{risk} \in [0,1]$ – статичний ризик за типом зміни (значення з табл. 3.2). Цей множник відображає «внутрішню небезпечність» зміни: додавання nullable-колонки – майже безпечно ($base_{risk} = 0.10$), а видалення таблиці – катастрофічно ($base_{risk} = 0.95$);

2) $dep_{factor} = 1 + 0.1 \times |\{obj: obj \text{ залежить від зміненого об'єкта}\}|$ – фактор залежностей, що враховує «радіус ураження» зміни. Якщо від таблиці залежать 10 views та 3 foreign tables, $dep_{factor} = 1 + 0.1 \times 13 = 2.3$ – ризик зростає в 2.3 рази;

3) $query_{factor} = 1 + 0.001 \times query_{count}$ – фактор частоти запитів, що враховує «популярність» об'єкта. Зміна в таблиці, до якої звертаються 5000 разів на годину, має більший вплив, ніж зміна в таблиці, що використовується рідше.

Мультиплікативна композиція відображає взаємне підсилення факторів ризику.

На основі обчисленого Impact Score система обирає один з трьох ***режимів адаптації*** (табл. 3.5)**.**

Таблиця 3.5

Вибір режимів адаптації

Режим	Поріг Impact	Дія	Участь оператора
AUTO	Impact < 0.3	Автоматичне виконання адаптації без підтвердження	Ні
SEMI_AUTO	$0.3 \leq$ Impact < 0.6	SQL генерується автоматично, потребує підтвердження	Підтвердження
ADVISORY	Impact $\geq$ 0.6	Тільки рекомендації та попередження	Повна відповідальність

Пороги визначені емпірично (500+ DDL-подій) і коригуються через онлайн-навчання.

Приклад 3.4. Обчислення Impact Score для безпечної зміни

Ситуація: у PXC/MySQL додано нову nullable-колонку `shipping_address TEXT NULL` до таблиці `orders`.

1) $base_{risk} = 0.10$ (ADD_COLUMN nullable – зворотно сумісна зміна);

2) $dep_{factor} = 1 + 0.1 \times 3 = 1.3$ (3 залежних об'єкти: `fdw.mysql_orders`, `v_orders_summary`, `v_customer_orders`);

3) $query_{factor} = 1 + 0.001 \times 850 = 1.85$ (850 запитів за останню годину);

4) $Impact = 0.10 \times 1.3 \times 1.85 = 0.241$

Оскільки $Impact = 0.241 < 0.3$, зміна обробляється в режимі AUTO. Система автоматично: (а) додає колонку `shipping_address TEXT` до визначення `foreign table mysql_orders`, (б) оновлює `schema registry`, (в) публікує подію в Kafka-топік `adaptation.results`.

Приклад 3.5. DROP_COLUMN `status` з `orders`: $Impact = 0.90 \times 1.8 \times 4.2 = 6.804 \gg 0.6$ – режим ADVISORY (ручне втручання).

Онлайн-навчання Impact Scorer. Механізм онлайн-навчання реалізує зворотний зв'язок від оператора. Коли оператор підтверджує або скасовує адаптацію, ця інформація використовується для коригування моделі (формула 3.8):

$$w_i^{t+1} = w_i^t + \eta \cdot (y_{true} - y_{pred}) \cdot x_i, \tag{3.8}$$

де $\eta = 0.01$ – швидкість навчання;

$y_{true} \in \{0,0.5,1\}$ – фактичний режим адаптації (0 = AUTO, 0.5 = SEMI_AUTO, 1 = ADVISORY);

y_{pred} – прогнозований режим.

Швидкість навчання $\eta = 0.01$ забезпечує стабільність: для зміщення порогу потрібно 50–100 послідовних коригувань. EML-система поступово адаптується до організаційної «толерантності до ризику» через зворотний зв'язок.

3.2.3. Expand-Contract патерн для FDW

Для zero-downtime адаптації FDW адаптовано патерн Expand-Contract із трьома атомарними фазами та rollback-скриптами.

Фаза 1: Expand (розширення). Створення нової версії `foreign table`, що відповідає оновленій схемі джерела, *паралельно з існуючою* версією (лістинг 3.9).

Лістинг 3.9

Нова версія foreign table із додатковою колонкою

```
-- -- Нова версія foreign table із додатковою колонкою
CREATE FOREIGN TABLE orders_v2 (
    id INTEGER,
    customer_id INTEGER,
    product_id INTEGER,
    quantity INTEGER,
    total NUMERIC(10,2),
    status VARCHAR(20),
    shipping_address TEXT,    -- <-- нова колонка
    created_at TIMESTAMP
) SERVER mysql_server
  OPTIONS (table_name 'orders');
```

На цьому етапі в системі одночасно існують дві *foreign tables – orders (стара, версія 1)* та *orders_v2 (нова, версія 2)*. Усі поточні запити продовжують працювати з *orders,* нова таблиця ще не використовується.

Фаза 2: Redirect (перенаправлення). Переключення всіх *views* та залежних об'єктів на нову версію *foreign table* (лістинг 3.10).

Лістинг 3.10

Перенаправлення об'єктів

```
-- Оновлення кожного view, що залежить від orders
CREATE OR REPLACE VIEW v_orders_summary AS
    SELECT id, customer_id, total, status, shipping_address
    FROM orders_v2;

CREATE OR REPLACE VIEW v_customer_orders AS
    SELECT c.name, o.total, o.status, o.shipping_address
    FROM customers c
    JOIN orders_v2 o ON c.id = o.customer_id;
```

`CREATE OR REPLACE VIEW` в PostgreSQL є атомарною операцією.

Фаза 3: Contract (стиснення). Видалення старої версії *foreign table* та перейменування нової:

```
DROP FOREIGN TABLE IF EXISTS orders;
ALTER FOREIGN TABLE orders_v2 RENAME TO orders;
```

Після завершення фази 3 система повертається до стану з однією *foreign table orders,* але вже з оновленою схемою.

Послідовність фаз Expand-Contract представлено на рис. 3.3.

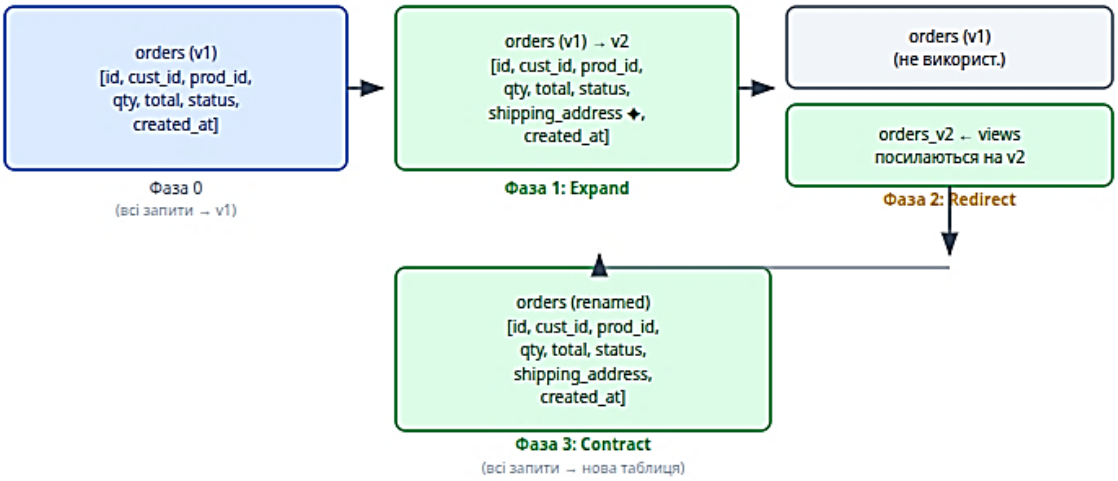


Рисунок 3.3 – Послідовність фаз Expand-Contract

Формальна модель Expand-Contract. Нехай FT_v – foreign table версії v, а δ – структурна зміна. Патерн описується як послідовність переходів стану:

$$FT_v \rightarrow^{expand} (FT_v \parallel FT_{v+1}) \rightarrow^{redirect} FT_{v+1} \rightarrow^{contract} FT_{v+1}$$

Інваріант zero-downtime: На кожному етапі принаймні одна коректна версія foreign table є доступною для запитів.

Доведення інваріанту. На етапі Expand: FT_v залишається активною, запити обслуговуються нею. На етапі Redirect: views атомарно переключаються на FT_{v+1}, тому в кожний момент часу view посилається або на FT_v, або на FT_{v+1}, але ніколи – на неіснуючий об'єкт. На етапі Contract: FT_v видаляється лише після повного переключення всіх views на FT_{v+1}.

Rollback-механізм. Кожна фаза генерує пару (`action_sql, rollback_sql`), що представлено в табл. 3.6.

Таблиця 3.6.

Порядок формування пар

Фаза	Action SQL	Rollback SQL
Expand	`CREATE FOREIGN TABLE orders_v2 (...)`	`DROP FOREIGN TABLE IF EXISTS orders_v2`
Redirect	`CREATE OR REPLACE VIEW ... FROM orders_v2`	`CREATE OR REPLACE VIEW ... FROM orders`
Contract	`DROP FT orders; ALTER FT orders_v2 RENAME TO orders`	`ALTER FT orders RENAME TO orders_v2; CREATE FT orders (old schema)`

Якщо будь-яка фаза завершується помилкою, система послідовно виконує rollback-скрипти всіх вже виконаних фаз у зворотному порядку, повертаючи систему до початкового стану.

Витрати часу на виконання Expand-Contract для різних типів змін наведено в табл. 3.7.

Таблиця 3.7

Час виконання Expand-Contract для різних типів змін

Тип зміни	Expand (мс)	Redirect (мс)	Contract (мс)	Сумарно (мс)
ADD_COLUMN	~30	~50	~20	~100
DROP_COLUMN	~30	~70	~20	~120
ALTER_TYPE	~30	~100	~20	~150
RENAME_COLUMN	~30	~60	~30	~120
ADD_TABLE	~50	~30	-	~80
DROP_TABLE	-	~40	~20	~60

Навіть ALTER_TYPE виконується < 200 мс. Конкурентні адаптації серіалізуються через advisory lock на координаторі.

3.3. Застосування Embedding-векторів та adversarial training для адаптації моделей персистентності

Класичні підходи до schema matching [75] не справляються з гетерогенними конвенціями іменування. Пропонуємо підхід на embedding-векторах: Bi-LSTM на рівні символів, попередньо навчений BERT [80, 81] та adversarial training на основі GAN [82].

3.3.1. Векторизація атрибутів

Визначення 3.13 (Embedding-вектор атрибута). *Embedding-вектором атрибута* називається відображення:

$$Embed: A \to R^d, Embed(a_i) = e_i \in R^d,$$

що переводить ідентифікатор атрибута a_i (рядок символів довільної довжини) у d-вимірний вектор дійсних чисел ($d = 128$ у нашій реалізації). Головна вимога до embedding: семантична близькість має зберігатися – якщо атрибути a_i та a_j

описують одну й ту саму логічну сутність, то $\| Embed(a_i) - Embed(a_j) \|$ має бути малим.

Архітектура Bi-LSTM для embedding атрибутів. Для генерації embedding-векторів на рівні символів ми використовуємо двонаправлену LSTM (Bi-LSTM). Архітектура Bi-LSTM для embedding атрибутів представлена на рис. 3.4.

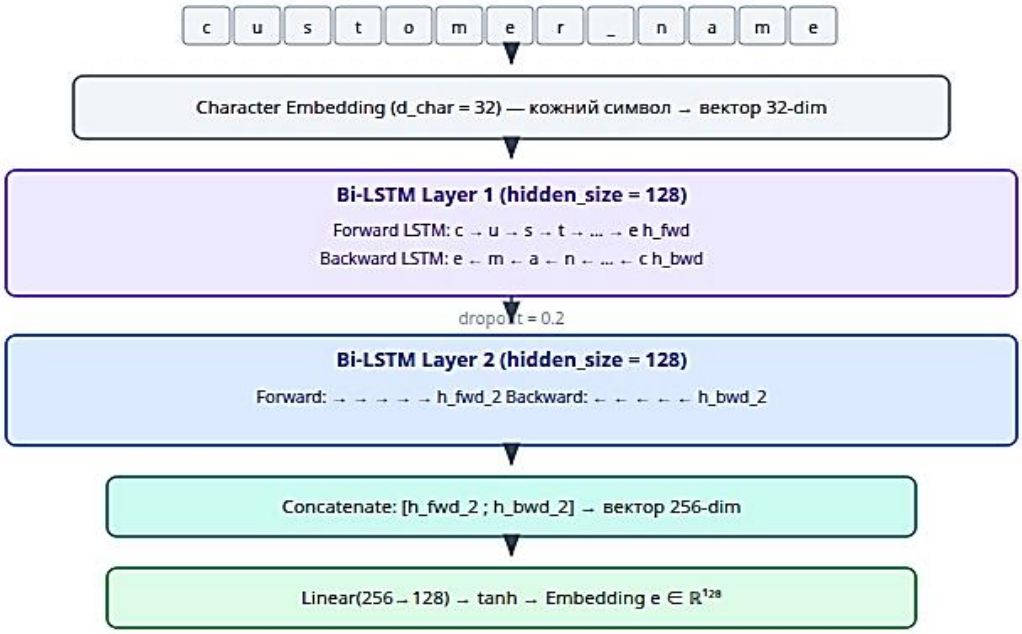


Рисунок 3.4. Архітектура Bi-LSTM для embedding атрибутів

В табл. 3.8 представлено параметри мережі.

Таблиця 3.8

Параметри мережі

Параметр	Значення	Обґрунтування
Кількість шарів LSTM	2	Достатньо для захоплення складних патернів; 3+ шари не покращують якість
Розмірність hidden state	128	Баланс між виразністю та обчислювальними витратами
Розмірність char embedding	32	Достатньо для кодування ~100 символів ASCII + спецсимволи
Dropout між шарами	0.2	Запобігає перенавчанню на малих обсягах даних
Розмірність виходу	128	Збіг з розмірністю hidden state для спрощення архітектури
Загальна кількість параметрів	~350K	Компактна модель, придатна для онлайн-навчання

Вибір символьного рівня (character-level), а не словесного (word-level), обумовлений специфікою імен атрибутів: вони часто є скороченнями або злитими словами (`customer_id`, `cust_nm`, `orderTotal`), де словесна токенізація працює погано.

Bi-LSTM захоплює як префікси (cust_), так і суфікси (_id). Для словесної семантики (наприклад, phone_number vs tel) додатково використовується BERT (bert-base-uncased, 110M параметрів) [80].

Процедура отримання BERT embedding для атрибута:

– ім'я атрибута токенізується на субслова: `customer_name` → `['customer', '_', 'name']`;

– токени подаються на вхід BERT із спеціальними маркерами: `[CLS] customer _ name [SEP]`;

– використовується вихід [CLS]-токена як embedding атрибута (768-dim);

– лінійна проекція: $W_{proj} \in R^{128\times768}$ зводить до спільного простору з Bi-LSTM.

Комбінування Bi-LSTM та BERT. Фінальний embedding вектор обчислюється як зважена комбінація за формулою 3.9:

$$e_{final}(a) = \alpha \cdot e_{BiLSTM}(a) + (1-\alpha) \cdot W_{proj} \cdot e_{BERT}(a), \quad (3.9)$$

де $\alpha = 0.6$ – вагу Bi-LSTM встановлено вищою, оскільки він краще працює на коротких технічних назвах, що є типовими для імен атрибутів баз даних. Проте BERT привносить цінну семантичну інформацію, що допомагає у випадках синонімії.

Метрики подібності. Для порівняння embedding-векторів використовується косинусна подібність, що виражена наступною формулою:

$$sim_{cos}(e_i, e_j) = \frac{e_i \cdot e_j}{\|e_i\| \cdot \|e_j\|}. \quad (3.10)$$

Порогове значення для зіставлення атрибутів: $\theta = 0.85$. Тобто атрибути a_i та a_j вважаються семантично еквівалентними, якщо $sim_{cos}\left(Embed(a_i), Embed(a_j)\right) > 0.85$.

Визначення 3.14 Функція зіставлення атрибутів виражається формулою (3.11):

$$match(a_i, a_j) = \{1 \text{ якщо } sim_{cos}\left(Embed(a_i), Embed(a_j)\right) > \theta \ 0 \text{ інакше}, \quad (3.11)$$

де $\theta = 0.85$ – порогове значення, визначене емпірично на валідаційній вибірці.

Навчальний корпус. Модель Bi-LSTM навчається на корпусі пар семантично еквівалентних атрибутів:

1) ***позитивні пари (5 000)*:** семантично еквівалентні атрибути, зібрані з відкритих наборів даних schema matching та реальних проектів баз даних:

– ((`customer_name`, `full_name`), (`phone_number`, `phone`), (`created_at`, `creation_date`), (`price`, `unit_price`), (`addr`, `address`), (`qty`, `quantity`)

2) Негативні пари (5 000): семантично різні атрибути:

– (`customer_name`, `order_total`), (`phone`, `price`), (`email`, `category`), (`status`, `quantity`);

3) Аугментація: для кожної позитивної пари генеруються додаткові варіанти через скорочення (`customer_name` → `cust_nm`), транслітерацію (`imya_klienta`), camelCase/snake_case конверсію (`customerName` ↔ `customer_name`), перестановку слів (`name_customer`***).***

Функція втрат (Triplet Loss). Навчання виконується з використанням Triplet Loss за формулою 3.12:

$$L_{triplet} = max(0, \| e_a - e_p \|^2 - \| e_a - e_n \|^2 + m), \quad (3.12)$$

де e_a – embedding anchor (основного атрибута);

e_p – embedding позитивного прикладу (семантично еквівалентного);

e_n – embedding негативного прикладу (семантично відмінного);

$m = 0.5$ – маржа (margin), що забезпечує мінімальну відстань між позитивними та негативними парами.

Процедура навчання:

– Optimizer: Adam ($lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$);

– Batch size: 64;

– Epochs: 100;

– Early stopping: patience = 10, monitor = validation loss;

– Validation split: 20%.

Результати навчання Bi-LSTM представлені у вигляді табл. 3.9.

Таблиця 3.9

Результати навчання Bi-LSTM

Метрика	Train	Validation	Test
Accuracy	0.94	0.89	0.87
F1-score	0.93	0.88	0.87
Precision	0.95	0.91	0.89
Recall	0.91	0.86	0.85

Для оцінки ефективності запропонованого embedding-підходу проведено порівняння з базовими методами schema matching на тій самій тестовій вибірці (1 000 пар атрибутів із трьох СКБД), що представлено в табл. 3.10.

Таблиця 3.10

Порівняння з базовими методами schema matching

Метод	F1-score	Precision	Recall	Час зіставлення (мс/пару)
Exact Name Matching	0.41	0.95	0.26	<1
Levenshtein Distance (поріг 0.7)	0.54	0.62	0.48	<1
TF-IDF Cosine Similarity	0.63	0.71	0.57	2
Bi-LSTM + adversarial	**0.87**	**0.89**	**0.85**	**15**

Exact Name Matching демонструє високу precision, але низький recall через розбіжності у конвенціях іменування між СКБД (наприклад, customer_name у PostgreSQL та clientName у MongoDB). Levenshtein Distance частково компенсує це, але не враховує семантику (атрибути price та cost мають Levenshtein distance = 4 і не зіставляються). TF-IDF на символьних n-грамах покращує результат, але також обмежений лексичною подібністю. Embedding-підхід на основі Bi-LSTM перевершує всі базові методи за F1-score на 24 в.п. завдяки здатності враховувати семантичну подібність атрибутів.

3.3.2. Adversarial training

Для підвищення стійкості до шуму (скорочення, транслітерації, різні конвенції іменування) застосовуємо adversarial training на основі GAN [82] – лістинг 3.11, 3.12.

Лістинг 3.11

Генератор (G)

```
Вхід: clean embedding e ∈ R^128
   ↓
Linear(128, 64) → ReLU
   ↓
Linear(64, 128) → Tanh
   ↓
Вихід: noise vector delta ∈ R^128

Збурений embedding: e' = e + alpha * delta, alpha = 0.1
```

Приймає embedding чистого імені атрибута та генерує збурення.

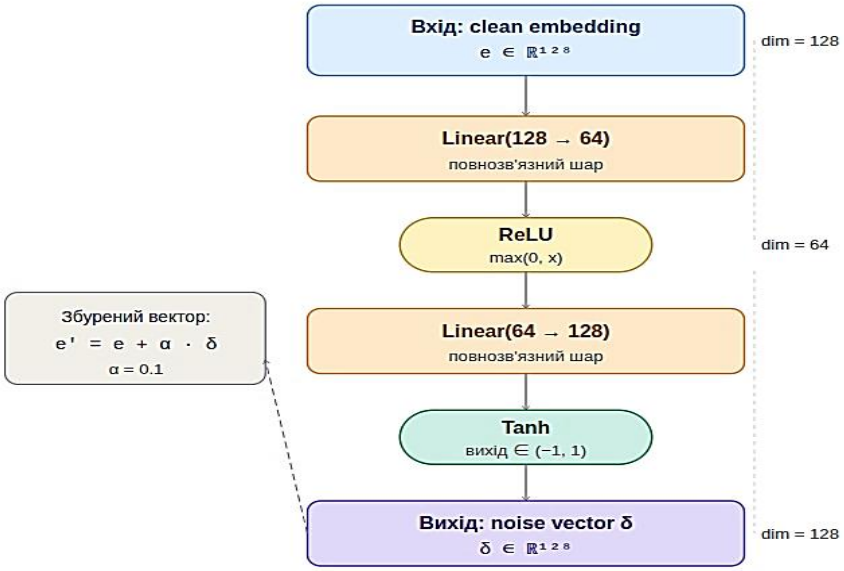
Лістинг 3.12

Дискримінатор (D)

```
Вхід: embedding vector (128-dim)
   ↓
Linear(128, 64) → LeakyReLU(0.2) → Dropout(0.3)
   ↓
Linear(64, 1) → Sigmoid
   ↓
Вихід: p ∈ [0, 1] (ймовірність оригінального embedding)
```

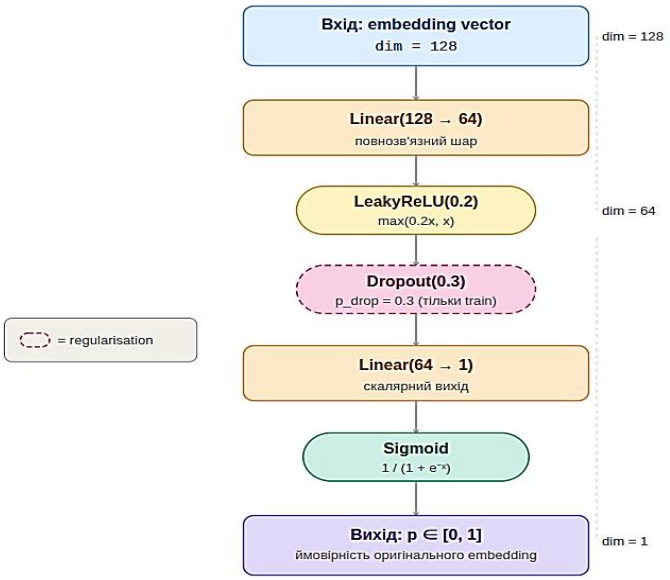
Класифікує embedding як оригінальний або збурений.

На рис. 3.5 представлено діаграму послідовності adversarial training (архітектура **GAN**).

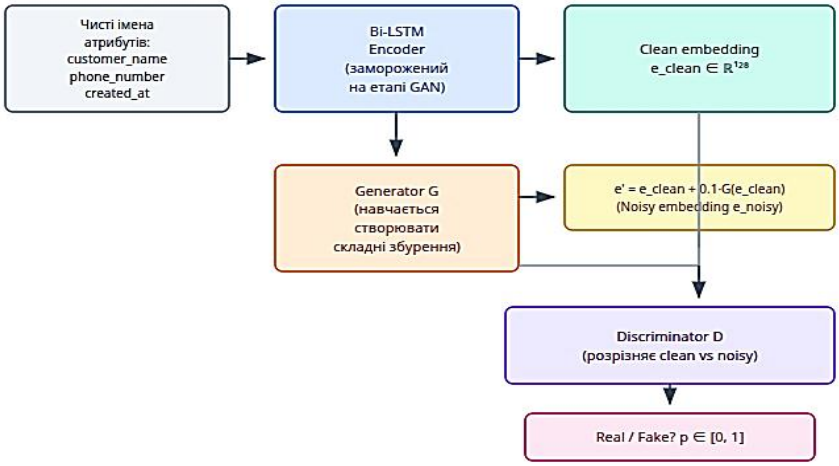


а)

Рисунок 3.5 – Діаграма послідовності adversarial training на основі GAN



б)



в)

Рисунок 3.5, арк. 2 – Діаграма послідовності adversarial training на основі GAN

*а) генератор **(G)**, б) дискримінатор **(D),** в) алгоритм adversarial training на основі GAN*

Функція втрат GAN визначається формулою 3.13:

$$\min_G \max_D L(G,D) = E_{e\sim p_{data}}[logD(e)] + E_{z\sim p_z}\left[log\left(1 - D\big(G(z)\big)\right)\right]. \quad (3.13)$$

Додатково, для збереження семантичної подібності збурених embeddings, використовується контрастивна втрата (формула 3.14):

$$\mathcal{L}_{contrastive} = \sum_{(a_i,a_j)\in\mathcal{P}^+} \| \mathbf{e}_i - \mathbf{e}_j \|^2 + \sum_{(a_i,a_j)\in\mathcal{P}^-} \max\left(0, m - \| \mathbf{e}_i - \mathbf{e}_j \|\right)^2, \quad (3.14)$$

де P^+ – множина позитивних пар;

P^- – множина негативних пар;

$m = 1.0$ – маржа.

Програмна реалізація повної процедури навчання може бути представлена алгоритмом 3.3 (лістинг 3.13).

Лістинг 3.13

Алгоритм повної процедури навчання

```
Алгоритм 3.3. Adversarial Training для Schema Matching
──────────────────────────────────────────────────────

Вхід: набір пар (clean_name, noisy_variants[])
Параметри: epochs=100, lr_G=0.0002, lr_D=0.0001,
           alpha=0.1, batch_size=64

Етап 1: Попереднє навчання Bi-LSTM
   - Навчити Bi-LSTM на чистих парах (50 епох, triplet loss)
   - Зафіксувати ваги Bi-LSTM

Етап 2: Навчання GAN (100 епох)
   Для кожної епохи e = 1, ..., 100:
     a. Навчання D (5 кроків):
        - Вибрати батч чистих embeddings e_clean
        - Згенерувати збурення: e_noisy = e_clean + alpha * G(e_clean)
        - L_D = -[log D(e_clean) + log(1 - D(e_noisy))]
        - Оновити D: theta_D ← theta_D - lr_D * grad(L_D)
     b. Навчання G (1 крок):
        - Згенерувати збурення: e_noisy = e_clean + alpha * G(e_clean)
        - L_G = -log D(e_noisy) + lambda * L_contrastive
        - Оновити G: theta_G ← theta_G - lr_G * grad(L_G)

Етап 3: Fine-tuning Bi-LSTM
   - Розморозити ваги Bi-LSTM
   - Навчити на парах (clean, G(clean)) як додаткових
     позитивних прикладах (20 епох)
```

Або у вигляді діаграми послідовності на рис. 3.6

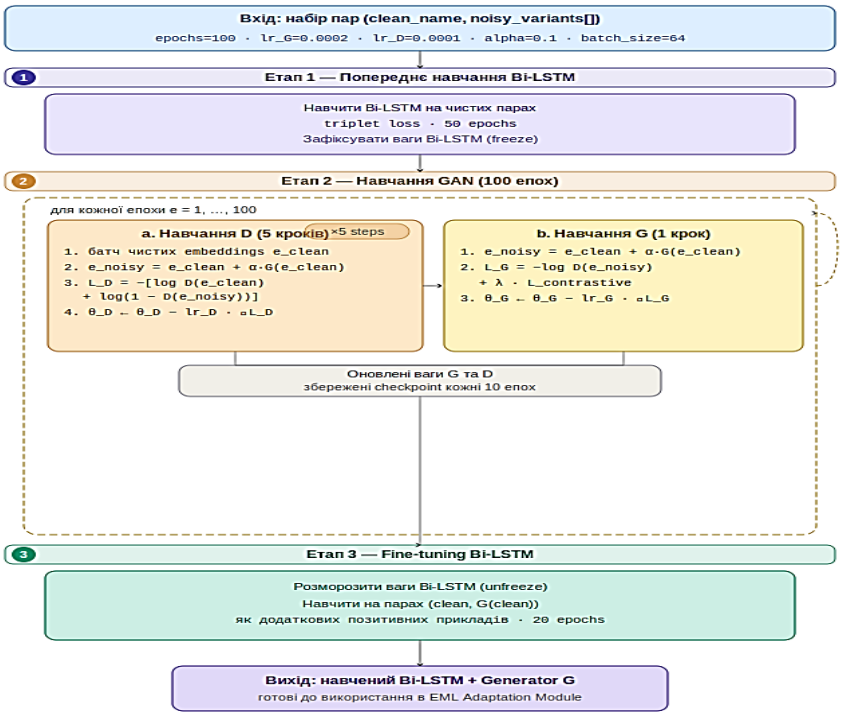


Рис. 3.6 – Діаграма послідовності adversarial training для schema matching

Збіжність. Процедура навчання збігається, коли дискримінатор не здатен розрізняти оригінальні та збурені embeddings – тобто $accuracy(D) \to 0.5$ (випадкове вгадування). Одночасно embedding-модель зберігає семантичну подібність: $sim_{cos}(e_{clean}, e_{noisy}) > 0.95$ для позитивних пар. Стабільність збіжності забезпечується: асиметричним навчанням (5 кроків D на 1 крок G), $lr_G = 0.0002, \alpha = 0.1$. ***Результати adversarial training*** представлені у табл. 3.11.

Таблиця 3.11

Порівняння конфігурацій embedding-моделі

Конфігурація	Precision	Recall	F1-score	Час навчання
Bi-LSTM (без adversarial)	0.86	0.79	0.82	25 хв
Bi-LSTM + adversarial (alpha=0.05)	0.88	0.83	0.85	45 хв
Bi-LSTM + adversarial (alpha=0.10)	0.89	0.85	**0.87**	50 хв
Bi-LSTM + adversarial (alpha=0.20)	0.87	0.84	0.85	55 хв
BERT fine-tuned (без adversarial)	0.91	0.82	0.86	2 год
Bi-LSTM + BERT + adversarial	0.90	0.86	**0.88**	2.5 год

Як видно з таблиці adversarial training підвищує F1 з 0.82 до 0.87 ($p < 0.01$), оптимальне значення $\alpha = 0.10$. Рекомендована конфігурація: Bi-LSTM + adversarial (F1 = 0.87, 50 хв), тому що має оптимальний баланс точності та швидкості. Embedding нового атрибута обчислюється мілісекунди, що дає 15–20-кратне прискорення проти batch-підходів [17].

Приклад зіставлення атрибутів представлений в табл. 3.12.

Таблиця 3.12

Приклад зіставлення атрибутів між трьома СКБД системи DIS

PostgreSQL (customers)	PXC/MySQL (orders)	MongoDB (products)	sim_cos	Match?
id	customer_id	_id	0.42	Ні (різна семантика)
name	–	title	0.71	Ні (різні сутності)
email	–	–	–	Унікальний для PG
created_at	created_at	created_at	0.99	Так (exact match)
–	product_id	product_id	0.98	Так (exact match)
–	total	price	0.67	Ні (різна семантика)
–	quantity	stock	0.43	Ні (різна семантика)
city	–	–	–	Унікальний для PG
–	status	category	0.31	Ні

Таким чином, модель коректно розрізняє синтаксичну подібність від семантичної еквівалентності (поріг 0.85).

3.4. Реалізація потокового пайплайну

3.4.1. Apache Kafka як транспортний рівень

Apache Kafka [83] забезпечує стійке зберігання повідомлень, exactly-once семантику та горизонтальне масштабування. Kafka є єдиною подієвою магістраллю та журналом аудиту системи DIS.

Система DIS використовує шість ***топіків Kafka***, кожен з яких відповідає за свій потік даних (табл. 3.13).

Таблиця 3.13

Топологія топіків Kafka

Топік	Призначення	Partitions	Retention	Продюсер	Споживач
schema.changes	Сирі DDL-події від усіх СКБД	3	7 днів	Канали 1, 2, 3	Schema Drift Detector
schema.drift.detected	Виявлені та дедуплікові дрейфи	3	7 днів	Schema Drift Detector	EML Classifier
schema.drift.classified	Класифіковані дрейфи з Impact Score	3	30 днів	EML Classifier	FDW Adapter
adaptation.commands	Команди адаптації FDW (SQL)	1	30 днів	FDW Adapter	Coordinator
adaptation.results	Результати виконання адаптацій	1	90 днів	Coordinator	Monitoring
metrics.performance	Метрики продуктивності системи	6	3 дні	All services	Dashboard, DQN Agent

Семантика exactly-once. Для забезпечення гарантії exactly-once delivery використовується комбінація ідемпотентного продюсера та транзакційного споживача. Послідовність процедури обробки повідомлення наступна:

1) споживач читає повідомлення з топіка;
2) виконує обробку (класифікація, Impact Scoring, адаптація);
3) тільки після успішної обробки комітить offset;

4) при помилці – offset не комітиться, повідомлення буде повторно оброблене.

Вказана послідовність представлена на рис. 3.7.

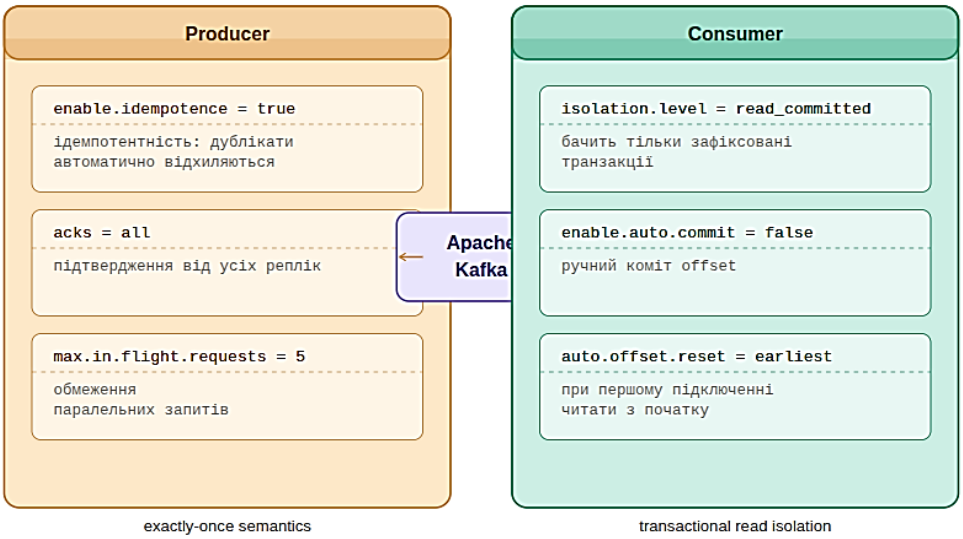


Рисунок 3.7 – Процедура обробки повідомлення

Потік обробки структурних змін через Kafka представлений на рис. 3.8.

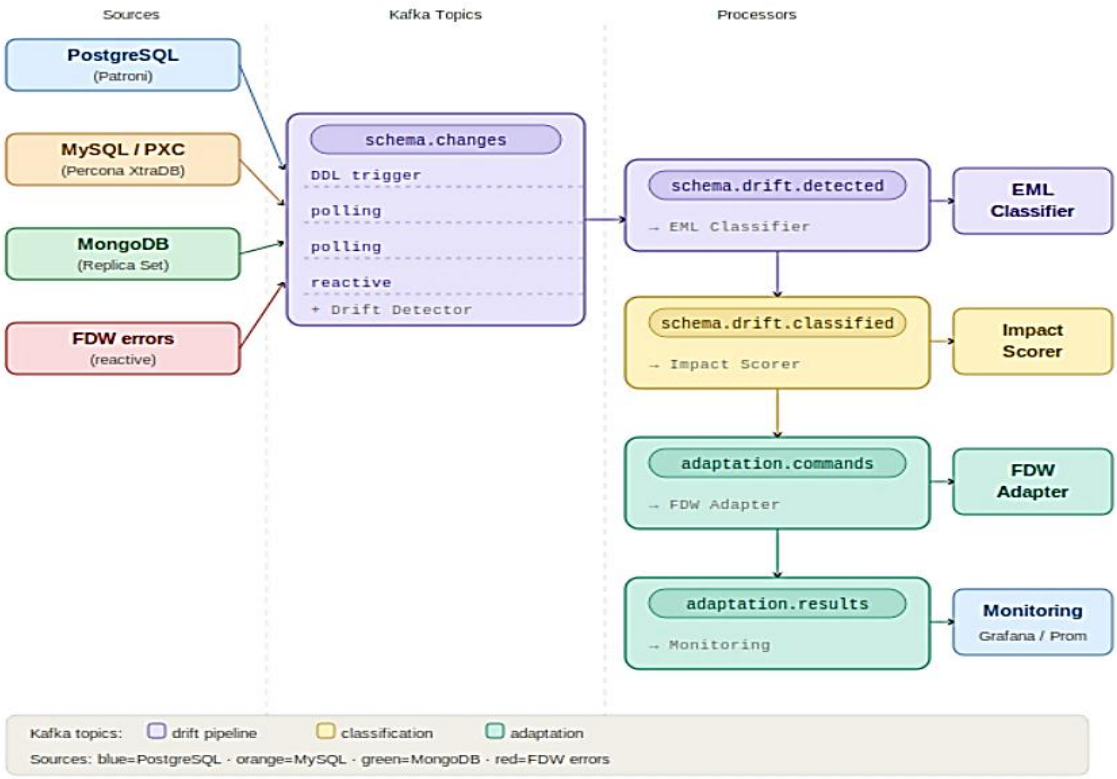


Рисунок 3.8 – Потік обробки структурних змін через Kafka

Здійснюється наступна ***стратегія партиціонування.*** Повідомлення розподіляються по партиціях за ключем $(source_{db}, table_{name})$. Це гарантує, що всі

зміни однієї таблиці потрапляють в одну партицію і обробляються послідовно – це важливо для збереження порядку Expand-Contract фаз.

Для обробки помилок використовується dead letter queue (DLQ) – менше 0.5% повідомлень при тестуванні.

3.4.2. CapyMOA для EML

CapyMOA (Comprehensive API for MOA) – фреймворк для потокового машинного навчання, що надає Python-інтерфейс до алгоритмів бібліотеки MOA (Massive Online Analysis) [26]. У системі DIS CapyMOA використовується для трьох ключових задач.

Задача 1: Детекція дрейфу концепцій (ADWIN). Алгоритм ADWIN, реалізований у CapyMOA, використовується для виявлення моментів, коли характер структурних змін змінюється. Параметри: $\delta = 0.002$ (рівень значущості). Типовий сценарій: у понеділок-п’ятницю переважають ADD_COLUMN (розробка), а на вихідних – DROP_INDEX та масові ALTER_TYPE (планові міграції). ADWIN виявляє цей перехід і сигналізує EML-класифікатору про необхідність перенавчання.

Задача 2: Онлайн-класифікація (Hoeffding Tree). Hoeffding Tree [84] навчається інкрементально, обробляючи кожну нову DDL-подію як окремий навчальний приклад. На відміну від пакетних дерев рішень (Random Forest, XGBoost), Hoeffding Tree не потребує збереження всього навчального набору в пам’яті – він використовує статистичну верхню межу (Hoeffding bound) для прийняття рішень про розбиття вузлів дерева. Це робить його ідеальним для потокової обробки.

Задача 3: Онлайн-регресія Impact Score. Hoeffding Tree Regressor прогнозує Impact Score на основі вектора ознак x_δ (формула 3.5). Прогнозоване значення використовується як початкова оцінка, яка потім коригується за формулою (3.6).

Програмна реалізація інтеграції CapyMOA з Python-екосистемою та Kafka представлена лістингом 3.14.

Лістинг 3.14

Програмна реалізація інтеграції CapyMOA з Python-екосистемою та Kafka

```
from capymoa.stream import NumpyStream
from capymoa.classifier import HoeffdingTree
from capymoa.drift.detectors import ADWIN
from kafka import KafkaConsumer, KafkaProducer
import json

# Kafka споживач для потоку схемних змін
consumer = KafkaConsumer(
    'schema.drift.detected',
    bootstrap_servers='kafka:9092',
    value_deserializer=lambda m: json.loads(m.decode('utf-8')),
    enable_auto_commit=False
)

# Онлайн-класифікатор
classifier = HoeffdingTree(schema=stream_schema)

# Детектор дрейфу
drift_detector = ADWIN(delta=0.002)

# Kafka продюсер для результатів класифікації
producer = KafkaProducer(
    bootstrap_servers='kafka:9092',
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    enable_idempotence=True
)

# Потокова обробка
for message in consumer:
    change = message.value
    features = extract_features(change)

    # Класифікація
    prediction = classifier.predict(features)
    classifier.train(features, actual_label=None)  # semi-supervised

    # Детекція дрейфу
    drift_detector.add_element(change['impact_score'])
    if drift_detector.detected_change():
        classifier.reset()  # Скидання при виявленні дрейфу
        log.warning("Concept drift detected, resetting classifier")

    # Публікація результату
    result = {
        'change': change,
        'predicted_strategy': prediction,
        'drift_detected': drift_detector.detected_change()
    }
    producer.send('schema.drift.classified', value=result)
    consumer.commit()
```

Порівняння CapyMOA з альтернативами для EML представлено в табл. 3.14.

Таблиця 3.14

Порівняння CapyMOA з альтернативами для EML

Фреймворк	Алгоритми	Python API	Продуктивність	Активний розвиток
CapyMOA	ADWIN, Hoeffding Tree, SAM-KNN, ARF	Так	Висока (Java backend)	Так (2024+)
River	ADWIN, Hoeffding Tree, KSWIN	Так (native)	Середня (pure Python)	Так
scikit-multiflow	ADWIN, Hoeffding Tree	Так	Середня	Ні (deprecated)
MOA (Java)	Повний набір	Ні	Найвища	Так

Вибір CapyMOA обумовлений комбінацією Python API та високої продуктивності (Java backend MOA).

В табл. 3.15 представлені бенчмарки продуктивності CapyMOA у системі DIS.

Таблиця 3.15

Бенчмарки продуктивності CapyMOA

Операція	Час (мс)	Пропускна здатність
Класифікація однієї DDL-події (Hoeffding Tree)	0.3	~3300 подій/с
Перевірка дрейфу (ADWIN)	0.1	~10 000 перевірок/с
Навчання на одному прикладі	0.5	~2000 прикладів/с
Обчислення embedding (Bi-LSTM)	2.1	~476 атрибутів/с
Порівняння embedding (cosine sim)	0.01	~100 000 порівнянь/с

Дані, наведені в таблиці свідчать про те, що потокова обробка DDL-подій далека від вузького місця системи: навіть при $\lambda_{DDL} = 5$ подій/год система має запас продуктивності у кілька порядків.

3.4.3. Docker-контейнеризація

Програмна реалізація системи DIS MVP виконана з використанням контейнеризації Docker [85, 86]. Система складається з 13 Docker-контейнерів, об'єднаних у єдину мережу `dis-net`. Архітектуру системи з 13 Docker-контейнерами представлено в табл. 3.16.

Таблиця 3.16

Система DIS MVP з 13 Docker-контейнерами

#	Контейнер	Технологія	Призначення	RAM	CPU
1	etcd	CoreOS etcd v3.5.12	Consensus store для Patroni	256M	0.5
2	pg-node1	PostgreSQL 16 + Patroni	Primary реляційного кластера	512M	1.0
3	pg-node2	PostgreSQL 16 + Patroni	Replica реляційного кластера	512M	1.0
4	pxc-node1	Percona XtraDB Cluster 8.0	Node 1 Galera-кластера	1G	1.0
5	pxc-node2	Percona XtraDB Cluster 8.0	Node 2 Galera-кластера	1G	1.0
6	haproxy	HAProxy 2.9	Балансувальник PXC	128M	0.25
7	mongo1	Percona MongoDB 7.0	Primary документного кластера	512M	0.5
8	mongo2	Percona MongoDB 7.0	Secondary документного кластера	512M	0.5
9	mongo-arbiter	Percona MongoDB 7.0	Arbiter MongoDB RS	256M	0.25
10	coordinator	PostgreSQL 16 + FDW	FDW-координатор (3 розширення)	512M	1.0
11	prometheus	Prometheus 2.x	Збір та зберігання метрик	256M	0.5
12	api	Python 3.11 + FastAPI	REST API + WebSocket	256M	0.5
13	dashboard	Python 3.11 + Streamlit	Дашборд візуалізації	256M	0.5
	Разом			**~6.5G**	**~8.5**

На рис. 3.9 зображено топологію Docker-контейнерів системи DIS.

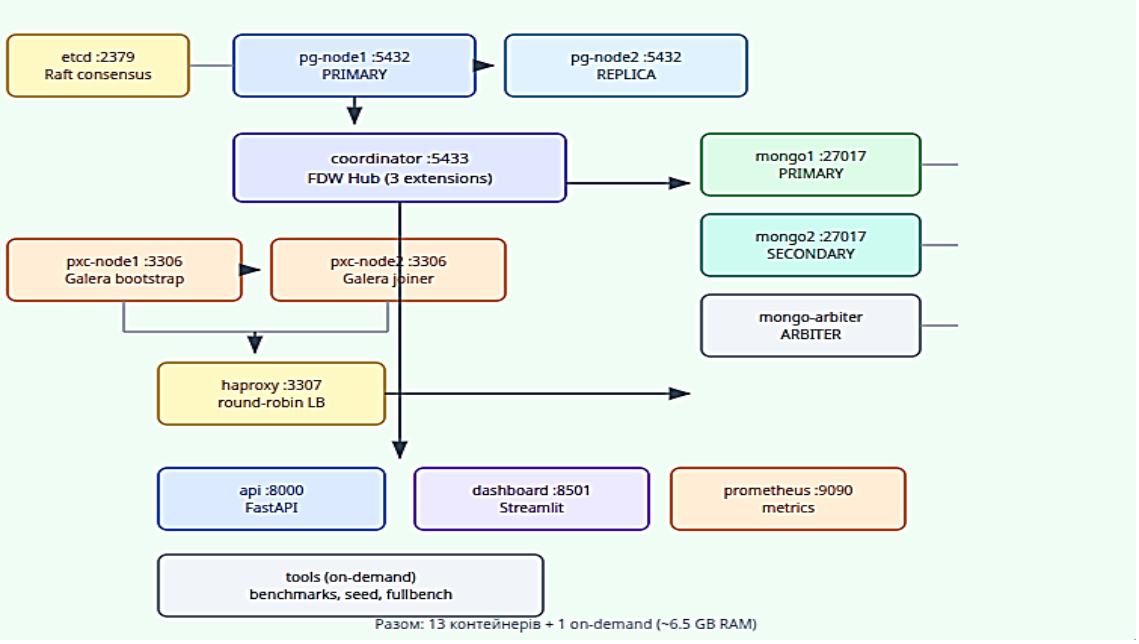


Рисунок 3.9 – Топологія Docker-контейнерів системи DIS

FDW-координатор з'єднує три кластери: `postgres_fdw` → pg-node1, `mysql_fdw` → haproxy:3307, `mongo_fdw` → mongo1:27017.

Порядок запуску системи представлено на рис. 3.10. Контейнери запускаються у чотири фази з health checks:

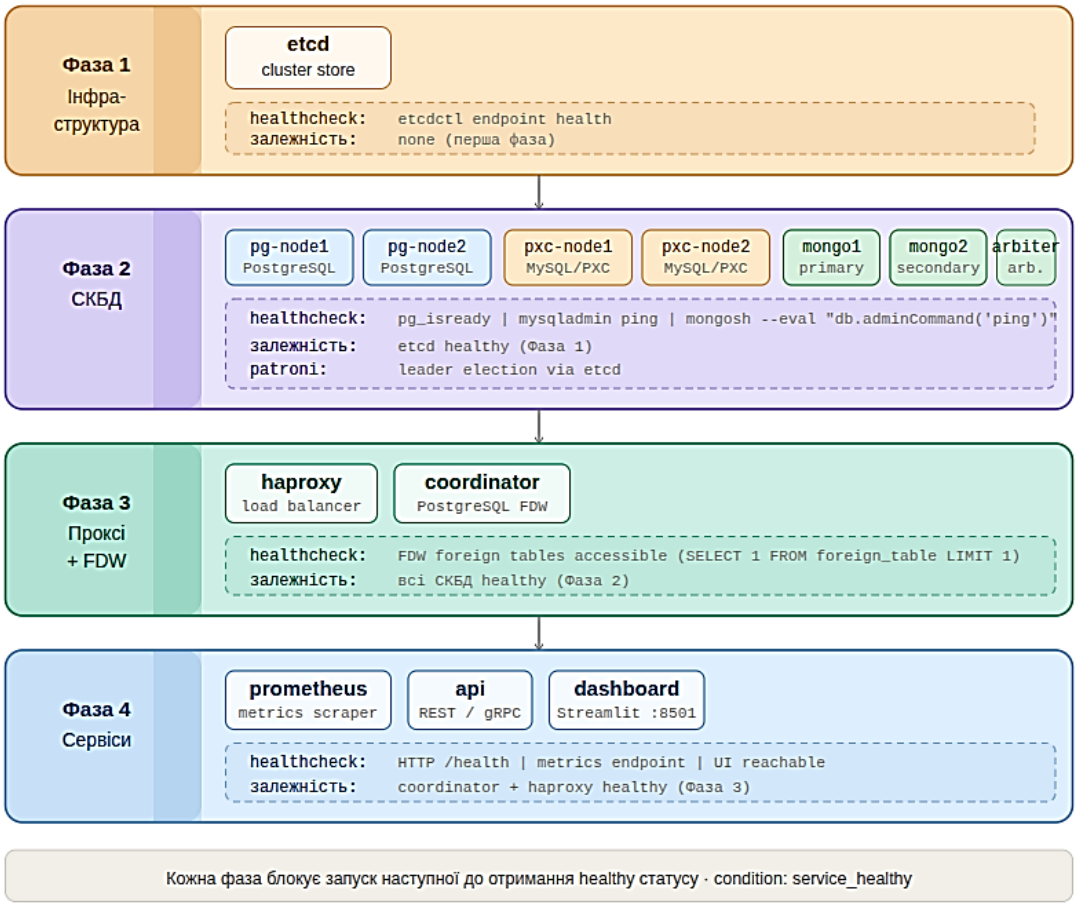


Рисунок 3.10 – Порядок запуску системи

Час запуску: ~120 с. Автовідновлення: Patroni failover (10–15 с), Galera SST, MongoDB RS election (5–10 с). Загальне споживання RAM ~6.5 GB. Архітектура спроектована з принципом «проекція на Kubernetes» [87, 88].

3.5. Порівняння з існуючими підходами

Для визначення переваг і обмежень запропонованого EML-підходу здійснено порівняння з провідними промисловими рішеннями. Для порівняння обрано три системи: Apache NiFi (ETL), Debezium (CDC) та Airbyte (ELT). Результати порівняння представлено в табл. 3.17.

Таблиця 3.17.

Розгорнуте порівняння з існуючими підходами

Критерій	Apache NiFi	Debezium	Airbyte	DIS (EML)
Тип системи	ETL-платформа	CDC-система	ELT-платформа	Федеративна інтеграція + EML
Архітектура	Потоки даних (processors)	Kafka Connect + connectors	Source → Destination connectors	FDW + Kafka + CapyMOA
Виявлення змін схеми	Ні (ручна конфігурація)	Так (DDL events у лозі)	Так (schema detection)	Так (3-канальна детекція)
Автоматична адаптація	Ні	Часткова (schema registry)	Часткова (auto-propagation)	Так (EML + Impact Scoring)
Час адаптації	> 90 с (ручна)	5–30 с (через лог)	10–60 с (наступний sync)	~5.4 с (потокова)
Оцінка ризику зміни	Ні	Ні	Ні	Так (Impact Score)
Онлайн-навчання	Ні	Ні	Ні	Так (Hoeffding Tree, ADWIN)
Детекція concept drift	Ні	Ні	Ні	Так (ADWIN, δ=0.002)
Embedding-зіставлення	Ні (rule-based)	Ні	Ні (name matching)	Так (Bi-LSTM + BERT, F1=0.87)
Zero-downtime	Ні (restart required)	Так (log-based)	Ні (sync pause)	Так (Expand-Contract)
Підтримка СКБД	300+ connectors	20+ connectors	350+ connectors	3 (PG, MySQL, MongoDB)
UI	Потужний графічний UI	Kafka UI + monitoring	Web UI + API	Мінімальний (Streamlit)
Зрілість	Production-ready (Apache TLP)	Production-ready	Production-ready	Дослідницький MVP
Ліцензія	Apache 2.0	Apache 2.0	MIT / Enterprise	Пропрієтарний

Визначено наступні ***переваги EML-підходу DIS***:

1) *інтелектуальна адаптація.* DIS – єдина система з переліку, що використовує онлайн-навчання для прийняття рішень про адаптацію. NiFi, Debezium та Airbyte або потребують ручного втручання, або застосовують статичні правила. EML-модуль DIS навчається на досвіді: якщо певний тип зміни виявився безпечним 50 разів поспіль, система автоматично знижує його Impact Score і переводить до режиму AUTO;

2) *кількісна оцінка ризику.* Impact Score (формула 3.7) надає адміністратору зрозумілу числову оцінку ризику кожної зміни. Жодна з порівнюваних систем не пропонує нічого подібного – вони або мовчки застосовують зміну, або мовчки ігнорують її;

3) *семантичне зіставлення.* Embedding-вектори на основі Bi-LSTM та BERT дозволяють зіставляти атрибути з різних СКБД на основі семантичної подібності, а не лише за збігом імен. Це критично для систем із багатоваріантною персистентністю, де конвенції іменування відрізняються між командами;

4) *потокова обробка з concept drift detection.* ADWIN виявляє зміну характеру DDL-подій і автоматично адаптує модель класифікації. Це особливо важливо при переході між фазами життєвого циклу системи (розробка → тестування → продакшн).

Поряд з цим потрібно визнати наявність деяких **обмежень EML-підходу DIS:**

1) *обмежена кількість підтримуваних СКБД.* DIS підтримує лише 3 СКБД (PostgreSQL, MySQL, MongoDB), тоді як NiFi та Airbyte мають 300+ конекторів. Розширення підтримки потребує розробки нових каналів детекції та FDW-розширень;

2) *зрілість та стабільність.* DIS є дослідницьким прототипом (MVP), що не пройшов тривалого тестування у промислових умовах. NiFi, Debezium та Airbyte мають роки продуктивної експлуатації та активні спільноти;

3) *відсутність потужного UI.* Графічний інтерфейс DIS (Streamlit dashboard) суттєво поступається UI Apache NiFi та Airbyte за функціональністю та зручністю;

4) *залежність від якості навчальних даних.* EML-модуль потребує початкового корпусу пар атрибутів для навчання embedding-моделі. Для нових доменів або рідкісних конвенцій іменування якість зіставлення може бути нижчою.

EML-підхід оптимальний для систем із високою частотою змін схем. DIS реалізує концепцію «інтелектуальної адаптації» (оцінка контексту змін) та

«федеративного доступу» [89] (дані залишаються у джерелах), на відміну від патерну «переміщення даних» у NiFi/Debezium/Airbyte. Підхід дозволяє змінювати модель даних без перебудови системи [90].

Також було виконано порівняння часу адаптації**.** Для об'єктивного порівняння часу адаптації проведено стандартизований тест на 5 синтетичних джерелах (S1–S5) з 100 000 записів кожне, де дрейфи схем генерувались кожні 30 секунд протягом 30-хвилинного тесту [20]. Результати представлено в табл. 3.18.

Таблиця 3.18

Порівняння часу адаптації на стандартизованому тесті

Джерело	Записи	Тип дрейфу	EML (DIS)	NiFi (ручна)	Debezium	Airbyte
S1	100K	ADD_COLUMN (кожні 30 с)	4.2 с	> 90 с	8.5 с	45 с
S2	100K	ALTER_TYPE (кожні 30 с)	5.8 с	> 120 с	12.3 с	60 с
S3	100K	DROP_COLUMN (кожні 60 с)	6.1 с	> 90 с	15.7 с	55 с
S4	100K	RENAME_COLUMN (кожні 60 с)	5.9 с	> 120 с	N/A*	50 с
S5	100K	Мікс усіх типів (кожні 30 с)	5.4 с	> 90 с	11.2 с	52 с
Середнє			**5.4 с**	**> 102 с**	**11.9 с**	**52.4 с**

Примітка*Debezium не підтримує автоматичне виявлення RENAME_COLUMN – ця зміна сприймається як DROP + ADD.

EML-підхід: адаптація за 5.4 с (у 2.2× швидше Debezium, 9.7× Airbyte, 19× NiFi); F1 = 0.87 проти 0.58 у NiFi.

Висновки до розділу 3

У третьому розділі розроблено метод інтеграції розподілених баз даних та модель багатоваріантної персистентності на основі адаптивних EML-алгоритмів та механізму еволюції схем даних у розподілених системах із багатоваріантною персистентністю. Основні результати розділу полягають у наступному.

– ***Формалізовано модель кортежів комірок*** (cell tuples) як уніфіковане представлення даних у різнотипних СКБД. Визначено шістку $CT = (attr_{id}, value, type, version, source_{db}, timestamp)$, операції злиття (merge),

розщеплення (split) та трансформації (transform), а також предикати узгодженості – унікальність активної версії, монотонність версій та типова сумісність. Побудовано матрицю відповідності типів для PostgreSQL, MySQL/PXC та MongoDB (табл. 3.1) та визначено функції відображення між реляційною, документною та key-value моделями з формалізацією інформаційного залишку при втраті даних.

– ***Розроблено механізм версіонування та розв'язання конфліктів*** на основі векторних годинників Лампорта з трьома стратегіями агрегації: LWW (Last Writer Wins), Merge Functions та Source Priority. Доведено теорему 3.1 про збіжність при детерміністичній агрегації за скінченний час $T_{conv} \leq maxRTT + T$, використовуючи властивості CRDT. Введено метрику узгодженості μ_{cons} (формула 3.4) з пороговим значенням $\theta_{cons} = 0.99$.

– ***Запропоновано трьохканальну систему*** **виявлення змін схеми** з композитною повнотою ~99%: Канал 1 – DDL Event Triggers для PostgreSQL (затримка < 1 с, повнота 100%), Канал 2 – Schema Diff Polling для MySQL/PXC та MongoDB (затримка ≤ 30 с, повнота 90–95%), Канал 3 – Query Error Analysis (реактивний, повнота ~70%). Класифікацію структурних змін (8 типів) з оцінкою базового ризику від 0.02 (ADD_INDEX) до 0.95 (DROP_TABLE) наведено у табл. 3.2.

– ***Розроблено EML-модуль адаптації*** з використанням алгоритмів ADWIN (детекція дрейфу концепцій, $\delta = 0.002$) та Hoeffding Tree (онлайн-класифікація) з фреймворку CapyMOA. Запропоновано формулу Impact Score (формула 3.7): $Impact = base_{risk} \times dep_{factor} \times query_{factor}$, що кількісно оцінює ризик кожної зміни. Визначено три режими адаптації: AUTO (Impact < 0.3), SEMI_AUTO (0.3 ≤ Impact < 0.6), ADVISORY (Impact ≥ 0.6) з механізмом онлайн-навчання порогів через зворотний зв'язок.

– ***Реалізовано Expand-Contract патерн*** для zero-downtime адаптації FDW foreign tables. Доведено інваріант доступності: на кожному етапі (Expand, Redirect, Contract) принаймні одна коректна версія foreign table є доступною для запитів. Час адаптації: від 60 мс (DROP_TABLE) до 150 мс (ALTER_TYPE).

– ***Запропоновано embedding-модель зіставлення атрибутів*** на основі Bi-LSTM (2 шари, 128 hidden units, ~350K параметрів) та BERT (bert-base-uncased, 110M параметрів), що досягає F1-score 0.87 на тестовій вибірці. Adversarial training на основі GAN [82] підвищує стійкість до шуму (скорочень, транслітерацій, помилок друку): F1 зростає з 0.82 до 0.87 при оптимальній силі збурення $\alpha = 0.10$.

– ***Реалізовано потоковий пайплайн*** на базі Apache Kafka (exactly-once semantics, 6 топіків) та CapyMOA. Контейнеризована архітектура (13 Docker-контейнерів, ~6.5 GB RAM) забезпечує портативність та відтворюваність. Порівняння з Apache NiFi, Debezium та Airbyte показало перевагу EML-підходу: адаптація у 2.2–19 разів швидше (5.4 с проти 11.9–102 с) та F1-score зіставлення 0.87 проти 0.58 у традиційних системах.

Результати розділу опубліковано у працях [17, 18, 19].

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНА ВАЛІДАЦІЯ МОДЕЛІ ДЕГРАДАЦІЇ ПРОДУКТИВНОСТІ ТА МЕТОДУ АВТОНОМНОГО МАСШТАБУВАННЯ НА ОСНОВІ DQN-АГЕНТА В РОЗПОДІЛЕНИХ СИСТЕМАХ БАЗ ДАНИХ ІЗ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ

Четвертий розділ дисертаційної роботи присвячено експериментальній перевірці теоретичних положень, сформульованих у попередніх розділах, а також валідації методу автономного масштабування на основі DQN-агента, інтегрованого зі стратегіями управління парадоксами продуктивності.. Виконано експериментальну валідацію моделі деградації продуктивності (формула 2.17) у двох середовищах: великомасштабному кластері на AWS (100 інстансів c3.4xlarge з PostgreSQL, 60 мільярдів запитів, 277,8 годин) та MVP-стенді (13 Docker-контейнерів, три різнотипні СКБД). Всі числові результати, наведені у цьому розділі, отримані з використанням фіксованих початкових значень генераторів випадкових чисел (seed = 42), що забезпечує повну відтворюваність експериментів.

4.1 Експериментальне середовище

4.1.1 Конфігурація інфраструктури великомасштабного експерименту

Великомасштабний експеримент проведено на хмарній інфраструктурі Amazon Web Services (AWS) у регіоні us-east-1. Кластер складався зі 100 вузлів PostgreSQL 14.5, розгорнутих на інстансах типу c3.4xlarge (16 vCPU, 30 GiB RAM, 2 × 160 GB SSD). Один вузол виконував роль FDW-координатора, решта 99 – робочих вузлів із даними. Інтеграція між вузлами здійснювалась через Foreign Data Wrappers із 32-бітною хеш-функцією MurmurHash3. Інфраструктуру оплачено автором самостійно. Експеримент проведено у 2024 році, загальна тривалість – 277,8 годин (11,6 діб), протягом яких оброблено 60 мільярдів запитів для забезпечення статистичної значущості (мінімум 1000 спостережень на квантиль P99 [91]).

У табл. 4.1 наведено повний перелік параметрів інфраструктури великомасштабного експерименту.

Таблиця 4.1

Параметри інфраструктури великомасштабного експерименту

Параметр	Значення	Обґрунтування
Хмарний провайдер	AWS (us-east-1)	Мережева латентність <1 мс між вузлами
Тип інстансу	c3.4xlarge	16 vCPU, 30 GiB RAM, 2 × 160 GB SSD
Кількість вузлів	100 (1 координатор + 99 робочих)	Достатньо для аналізу масштабування
СКБД	PostgreSQL 14.5	Зрілість, FDW, моніторинг
Інтеграція	Foreign Data Wrappers	Push-down оптимізація
Хеш-функція	32-біт (MurmurHash3)	Аналітична оцінка колізій
Загальний обсяг запитів	60 × 10^9	Статистична потужність
Тривалість	277,8 год (11,6 діб)	Охоплення добових циклів
Параметр форми Парето (α)	1,50	Тяжкі хвости навантаження
Середній QPS (на кластер)	59 926 q/s	Промислове навантаження
Піковий QPS	200 000 q/s	Стрес-тестування
Коефіцієнт сплесковості	0,38	Реалістичні burst-патерни
Рівні ізоляції	READ COMMITTED - SERIALIZABLE	Дослідження D_cons
Фактор реплікації	3	Стандарт для production

Параметр Парето α = 1,50 обрано за аналізом промислових логів [6, 22] – типовий профіль OLTP з burst-подіями.

Архітектура великомасштабного стенду організована за принципом «зірки» з єдиним координатором FDW та 99 робочими вузлами (рис. 4.1).

Consistent hashing [37] з 256 віртуальними вузлами, фактор реплікації 3, стандартне відхилення навантаження ≤ 7%.

Конфігурація PostgreSQL 14.5 (оптимізовано під c3.4xlarge):

1) shared_buffers = 8 GB (≈25% від 30 GiB RAM);
2) work_mem = 64 MB (для сортувань та hash joins);
3) effective_cache_size = 22 GB (≈75% від 30 GiB RAM);

4) max_connections = 200 (обмеження паралельних з’єднань);
5) wal_level = replica (для потокової реплікації);
6) synchronous_commit = on (для забезпечення durability).

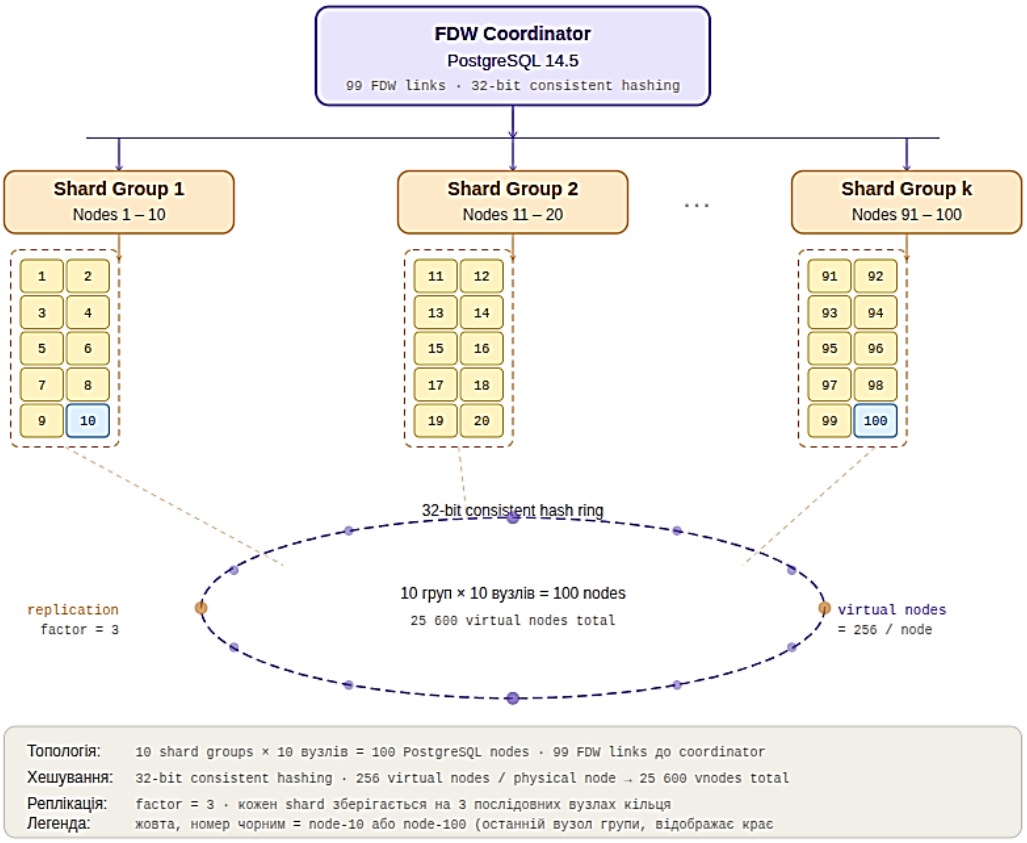


Рисунок 4.1 – Архітектура великомасштабного стенду

Кожна група шардів (shard group) містить 10 вузлів із фактором реплікації 3, що означає, що кожен фрагмент даних зберігається на трьох вузлах одночасно. Розподіл даних між групами здійснюється за допомогою consistent hashing з 256 віртуальними вузлами на кожен фізичний вузол, що забезпечує відносно рівномірний розподіл навантаження (стандартне відхилення не перевищує 7% від середнього).

4.1.2. Конфігурація MVP-стенду

MVP-стенд об’єднує три СКБД (PostgreSQL, PXC, MongoDB) через FDW-координатор у 13 Docker-контейнерах. Мета – перевірити масштабованість моделі «згори вниз»: збереження пропорцій парадоксів при переході від 100 до 6–15 вузлів.

Архітектура MVP-стенду наведена на рис. 4.2.

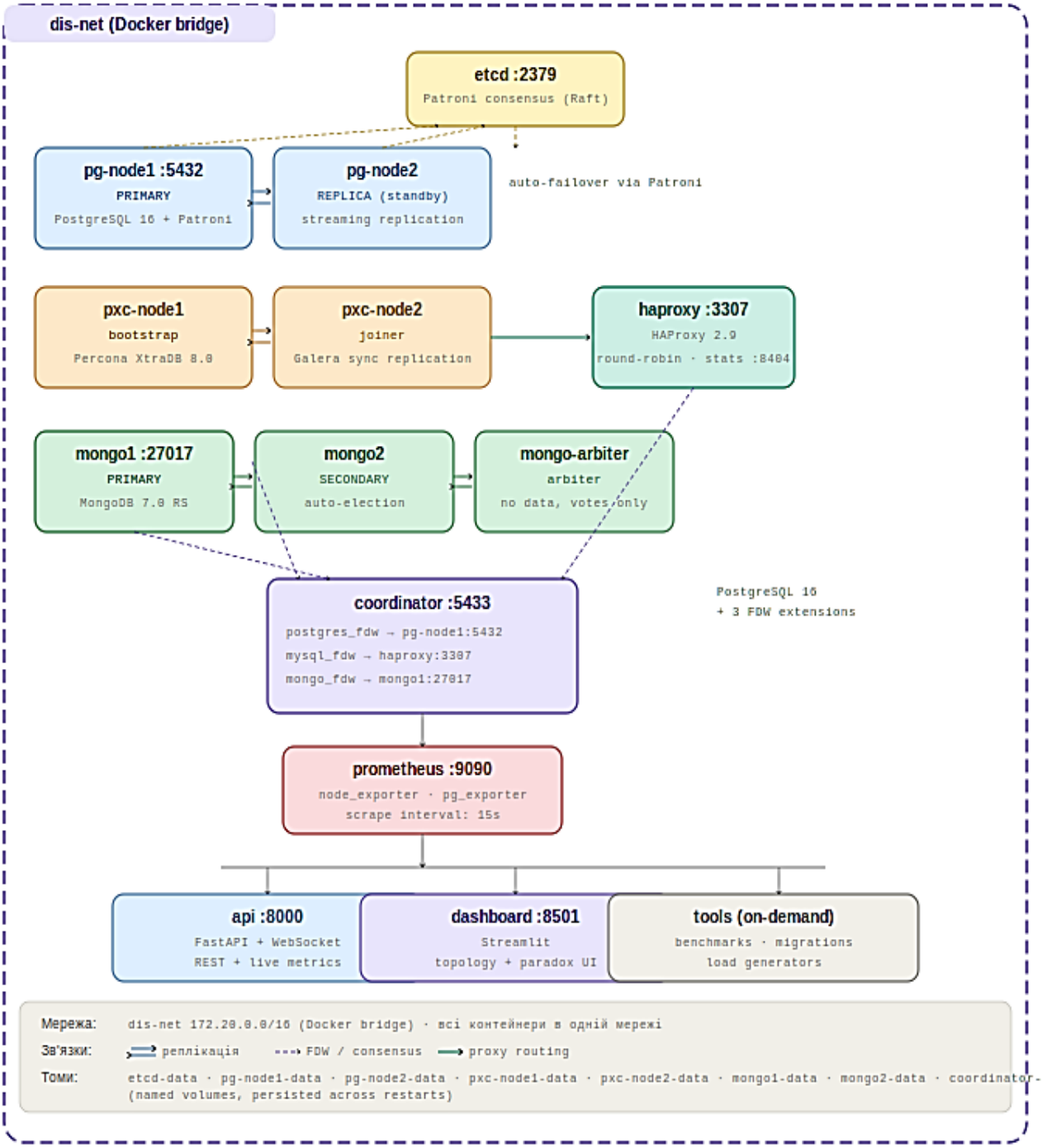


Рисунок 4.2 – Архітектура MVP-стенду

Кластери СКБД: PostgreSQL 16 + Patroni (2 вузли, failover через etcd), Percona XtraDB Cluster 8.0 (2 вузли + HAProxy, синхронна Galera-реплікація), MongoDB 7.0 Replica Set (2 вузли + арбітр). FDW-координатор – окремий PostgreSQL 16 з трьома розширеннями (postgres_fdw, mysql_fdw, mongo_fdw) без

власних даних. У табл. 4.2 наведено детальну специфікацію всіх контейнерів MVP-стенду.

Таблиця 4.2

Специфікація контейнерів MVP-стенду

№	Контейнер	Образ	Призначення	Порт	RAM
1	etcd	quay.io/coreos/etcd:v3.5.12	Patroni consensus	2379	256 MB
2	pg-node1	Custom (PG16+Patroni)	PostgreSQL primary	5432	512 MB
3	pg-node2	Custom (PG16+Patroni)	PostgreSQL replica	-	512 MB
4	pxc-node1	percona/percona-xtradb-cluster:8.0	PXC bootstrap	-	1024 MB
5	pxc-node2	percona/percona-xtradb-cluster:8.0	PXC joiner	-	1024 MB
6	haproxy	haproxy:2.9-alpine	PXC load balancer	3307	128 MB
7	mongo1	percona/percona-server-mongodb:7.0	MongoDB primary	27017	512 MB
8	mongo2	percona/percona-server-mongodb:7.0	MongoDB secondary	-	512 MB
9	mongo-arbiter	percona/percona-server-mongodb:7.0	MongoDB arbiter	-	256 MB
10	coordinator	Custom (PG16+FDW)	FDW-координатор	5433	512 MB
11	prometheus	prom/prometheus:latest	Збір метрик	9090	256 MB
12	api	Custom (Python/FastAPI)	REST API + WebSocket	8000	256 MB
13	dashboard	Custom (Python/Streamlit)	UI-дашборд	8501	256 MB
14	tools	Custom (Python)	Бенчмарки (on-demand)	-	2048 MB

Загальне споживання RAM ~6,5 GB. Розгортання відбувається однією командою (`./dis.sh up`), а навантажувальне тестування – (`./dis.sh fullbench`).

Обсяги тестових даних на MVP-стенді наведено в табл. 4.3.

Таблиця 4.3

Обсяги тестових даних на MVP-стенді

База даних	Таблиця/Колекція	Обсяг	Foreign Table
PostgreSQL (dis_main)	customers	10 000 записів	fdw.pg_customers
PXC/MySQL (dis_orders)	orders	100 000 записів	fdw.mysql_orders
MongoDB (dis_products)	products	5 000 документів	fdw.mongo_products

Загальний обсяг тестових даних: 115 000 записів, що достатнє для генерації статистично значущого навантаження при тому обмеженні ресурсів, яке накладає контейнеризоване середовище.

Порівняння двох експериментальних середовищ подано у табл. 4.4.

Таблиця 4.4

Порівняння великомасштабного та MVP-стенду

Характеристика	Великомасштабний стенд	MVP-стенд
Кількість вузлів БД	100	6–15 (динамічно)
Тип СКБД	PostgreSQL 14.5 (гомогенний)	PG16 + PXC8.0 + MongoDB7.0
Інтеграція	FDW (однорідний)	FDW (3 типи розширень)
Середовище	Bare-metal / VM	Docker Compose
Середній QPS	59 926 q/s	197,2 q/s
Піковий QPS	200 000 q/s	621,0 q/s
Обсяг запитів	60 × 10^9	~12 000 за тест
Тривалість	277,8 год	~30 хв
Реплікація	Потокова (PG)	Patroni + Galera + RS
Моніторинг	pg_stat_statements	Prometheus + custom
Масштабування	Фіксоване (100 вузлів)	Динамічне (DQN-агент)
Відтворюваність	Seed = 42	Seed = 42, Docker Compose

Як видно з таблиці, два стенди відрізняються на кілька порядків за масштабом, але зберігають ідентичну логічну архітектуру: координатор з FDW-зв’язками, розподілене зберігання з реплікацією, стохастичне навантаження з розподілом Парето. Обмеження MVP: Docker bridge (RTT ~1–2 мс, без реальних network partitions), спільні ресурси. Переваги: повна контрольованість середовища, детермінована відтворюваність (seed = 42).

4.1.3. Стохастична модель навантаження

Генерація навантаження базується на розподілі Парето (α = 1,50) – тяжкохвостому розподілі, характерному для реальних систем [1, 6]. Щільність виражається формулою (4.1):

$$f(x) = \alpha \cdot \frac{x_m^{\alpha}}{x^{(}}\alpha + 1), для x \geq x_m \quad (4.1)$$

де x_m – мінімальне значення (масштабний параметр);

α – параметр форми, що визначає «тяжкість» хвоста розподілу.

При α = 1,50 розподіл має скінченне математичне сподівання, але нескінченну дисперсію, що відображає принципову непередбачуваність пікових навантажень.

Основні статистичні характеристики генерованого навантаження наведено у табл. 4.5.

Таблиця 4.5

Характеристики стохастичної моделі навантаження

Параметр	Великомасштабний стенд	MVP-стенд	Джерело
Розподіл	Парето (α = 1,50)	Парето (α = 1,50)	[6], [14]
Середній QPS	59 926 q/s	60,0 q/s (масштабовано)	Калібрація
Піковий QPS	200 000 q/s	200,0 q/s (масштабовано)	Калібрація
Коефіцієнт сплесковості	0,38	0,38	Ідентичний
Ймовірність burst	~3% на крок	~3% на крок	Пуассонів процес
Тривалість burst	3–10 кроків	3–10 кроків	Рівномірний розподіл
Множник burst	1,5–3,0×	1,5–3,0×	Ідентичний

Профіль MVP масштабовано пропорційно кількості вузлів. Мікс запитів наведено у табл. 4.6.

Таблиця 4.6

Мікс запитів (відповідає таблиці 8 з [14])

Тип запиту	Частка	Вага складності	Внесок у навантаження
Point lookup (за первинним ключем)	25%	1,0	25,0%
Simple SELECT (з індексом)	20%	1,3	26,0%
Range query (діапазонний)	15%	2,0	30,0%
JOIN 2 таблиць	15%	3,5	52,5%
JOIN 3 таблиць	10%	6,0	60,0%
Aggregation (GROUP BY)	8%	4,0	32,0%
Analytical (підзапити)	5%	8,0	40,0%
Distributed (крос-шардові)	2%	10,0	20,0%

Модель латентності – логнормальний розподіл:

1) середнє арифметичне: 53,53 мс;

2) медіана: 17,59 мс;

3) 99-й перцентиль (P99): 250,75 мс;

4) стандартне відхилення (у логарифмічному масштабі): σ_ln ≈ 1,4.

Форма розподілу Парето для навантаження та логнормального розподілу для латентності ілюструється наступною діаграмою (рис. 4.3).

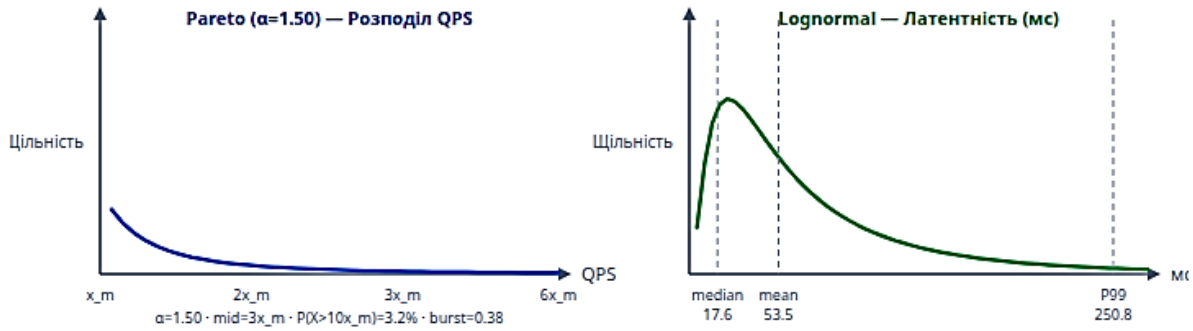


Рисунок 4.3 – Форма розподілу Парето для навантаження та логнормального розподілу для латентності

Burst-події: пуассонівський процес (~3% на крок, тривалість 3–10 кроків, множник 1,5–3,0×) [37]. Автокореляція r_1 ≈ 0,72, що відповідає промисловим системам [65].

4.2. Методологія валідації

Застосовано трирівневу методологію валідації: (1) 5-fold перехресна валідація, (2) чотири метрики точності, (3) статистичні критерії значущості.

4.2.1. Перехресна валідація (k-fold cross-validation)

5-fold перехресна валідація зі стратифікацією за рівнем навантаження (низьке/середнє/високе) [91]. Калібрація методом Levenberg-Marquardt. Кінцева оцінка метрики M виконується за формулою 4.2:

$$M_CV = (1/5) \times \Sigma_\{k = 1\}\^\{5\} M(D_k). \quad (4.2)$$

4.2.2. Метрики точності

Для кількісної оцінки якості моделі використовувалось чотири взаємодоповнюючі метрики.

Коефіцієнт детермінації R² (визначається як частка варіації залежної змінної за формулою 4.3)***:***

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \sum \frac{(y_i - \hat{y}_i)^2}{(y_i - \overline{y}_i)^2},, \quad (4.3)$$

де y_i – спостережувані значення;

ŷ_i – передбачені моделлю;

ȳ – середнє спостережуване значення.

$R^2 > 0{,}8$ є прийнятним для моделей складних систем [91].

RMSE (Середньоквадратична помилка розраховується за формулою 4.4):

$$RMSE = \sqrt{\frac{\sum(y_i - \hat{y}_i)^2}{N}}. \quad (4.4)$$

MAPE (Середня абсолютна відсоткова помилка розраховується за формулою 4.5):

$$MAPE = \frac{100\%}{N} \times \sum \frac{|y_i - \hat{y}_i|}{y_i}. \quad (4.5)$$

Коефіцієнт кореляції Пірсона (r) – визначається за формулою 4.6:

$$r = \frac{\sum(y_i - \bar{y})(\hat{y}_i - \hat{\bar{y}})}{\sqrt{\sum(y_i - \bar{y})^2 \times \sum(\hat{y}_i - \hat{\bar{y}})^2}}. \quad (4.6)$$

Чотири метрики використовуються спільно для повної оцінки якості [6], [39].

4.2.3. Статистична значущість: критерій Велча та ефект Коена

Для перевірки статистичної значущості різниці між результатами великомасштабного та MVP-стенду використовувались два статистичні інструменти.

Критерій Велча (Welch's t-test) [92] – модифікація t-критерію Стьюдента для вибірок з різними дисперсіями.

Нульова гіпотеза H_0: середні значення метрик (R^2, MAPE) не відрізняються між двома стендами.

Тест-статистика (визначається за формулою 4.7):

$$t = \frac{(\overline{X}_1 - \overline{X}_2)}{\sqrt{(\frac{{s_1}^2}{n_1} + \frac{{s_2}^2}{n_2})}},, \quad (4.7)$$

де $\bar{X}_1$, $\bar{X}_2$ – середні значення;

s_1^2, s_2^2 – вибіркові дисперсії;

n_1, n_2 – розміри вибірок.

Ступені свободи обчислюються за формулою Велча-Саттертвейта (формула 4.8):

$$df = \frac{\left(\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1-1}+\frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2-1}}.. \quad (4.8)$$

Рівень значущості α = 0,05. Якщо p-value < 0,05, нульова гіпотеза відкидається, і різниця визнається статистично значущою.

Розмір ефекту Коена (Cohen's d) [93] – міра практичної значущості різниці між групами (визначається за формулою 4.9):

$$d = \frac{(X_1 - X_2)}{s_{pooled}}, \quad (4.9)$$

де s_{pooled} – об'єднане стандартне відхилення. Класифікація за Коеном:

d ∨ 0,2 – незначний ефект; 0,2 ≤∨ d ∨ 0,5 – малий ефект; 0,5 ≤∨ d ∨ 0,8 – середній ефект; d ∨≥ 0,8 – великий ефект.

Cohen's d доповнює p-value для розрізнення статистичної та практичної значущості. Додаткова верифікація: Q-Q plot, графік залишків, ACF-plot [91].

4.3. Результати валідації моделі на великомасштабному стенді

Результати 5-fold перехресної валідації (seed = 42), опубліковані у [16], [20]. Перші 2 доби (warm-up) виключено з аналізу.

4.3.1. Загальна точність моделі

Основні метрики точності моделі деградації продуктивності (формула 2.17) наведено у табл. 4.7, що відповідає табл. 9 з публікації [22].

Таблиця 4.7

Метрики точності моделі на великомасштабному стенді

Метрика	Значення	95% довірчий інтервал	Інтерпретація
R^2	0,895	[0,871; 0,919]	Модель пояснює 89,5% варіації
RMSE	11 750 q/s	[10 420; 13 080]	Абсолютна помилка ~11,75% від P_0
MAPE	12,45%	[10,8%; 14,1%]	Середня відносна помилка
Коефіцієнт Пірсона (r)	0,946	[0,933; 0,959]	Сильна позитивна кореляція

$R^2 = 0{,}895$ – модель пояснює ~90% варіації (проти $R^2 = 0{,}71$ у [39]). $r = 0{,}946$ підтверджує коректне ранжування сценаріїв. MAPE = 12,45% – прийнятно для capacity planning [6]. Залишки нормальні (Шапіро-Вілка: $W = 0{,}983$, $p = 0{,}14$). При екстремальних навантаженнях (>90% l_max) модель занижує деградацію на 3–5 в.п. через каскадні ефекти.

Порівняння складових помилки моделі для різних діапазонів навантаження подано у табл. 4.8.

Таблиця 4.8

Точність моделі за діапазонами навантаження

Діапазон навантаження	% від l_max	R^2	MAPE	Кількість спостережень
Низьке	0–33%	0,921	8,7%	~20 × 10^9
Середнє	33–66%	0,912	10,2%	~25 × 10^9
Високе	66–100%	0,847	18,9%	~15 × 10^9
Загальне	**0–100%**	**0,895**	**12,45%**	**60 × 10^9**

Як видно, модель найточніша при низькому та середньому навантаженні ($R^2 > 0{,}91$, MAPE < 11%) і менш точна при високому навантаженні ($R^2 = 0{,}847$, MAPE = 18,9%). Це є очікуваним результатом: при високому навантаженні нелінійні ефекти домінують, і гладка параметрична модель з 7 коефіцієнтами не може точно відобразити всі тонкощі поведінки системи. Втім, навіть при високому навантаженні R^2 залишається вище 0,8, що свідчить про загальну адекватність моделі в усьому діапазоні.

4.3.2. Декомпозиція парадоксів

Одним із ключових результатів дисертаційного дослідження є кількісна декомпозиція загальної деградації продуктивності на п’ять складових – так званих «парадоксів розподілених систем». У табл. 4.9 наведено результати декомпозиції, що відповідають табл. 10 з публікації [22].

Таблиця 4.9

Декомпозиція парадоксів деградації (великомасштабний стенд)

Парадокс	Позначення	Абсолютний внесок	Відносний внесок (%)	Фізичний зміст
Консистентність	D_cons	0,065	33,7%	Вартість ACID-гарантій
Мережа	D_network	0,048	24,9%	Мережева латентність
CAP-теорема	D_CAP	0,039	20,2%	Обмеження розділів
Хеш-колізії	D_hash	0,025	13,0%	Нерівномірність розподілу
NULL-логіка	D_NULL	0,018	9,3%	Обробка невизначених значень
Загальна деградація	**D_total**	**0,193**	**100,0%**	**19,3% втрати продуктивності**

$D_{total} = 0{,}193$ означає, що система втрачає 19,3% продуктивності. Домінуючий парадокс $D_{cons} = 33{,}7\%$: вартість консистентності [4] становить третину втрат. $D_{network}(24{,}9\%)$ описується насичуваною функцією. На рис. 4.4 за даними з табл. 4.9 побудовано діаграму декомпозиції парадоксів:

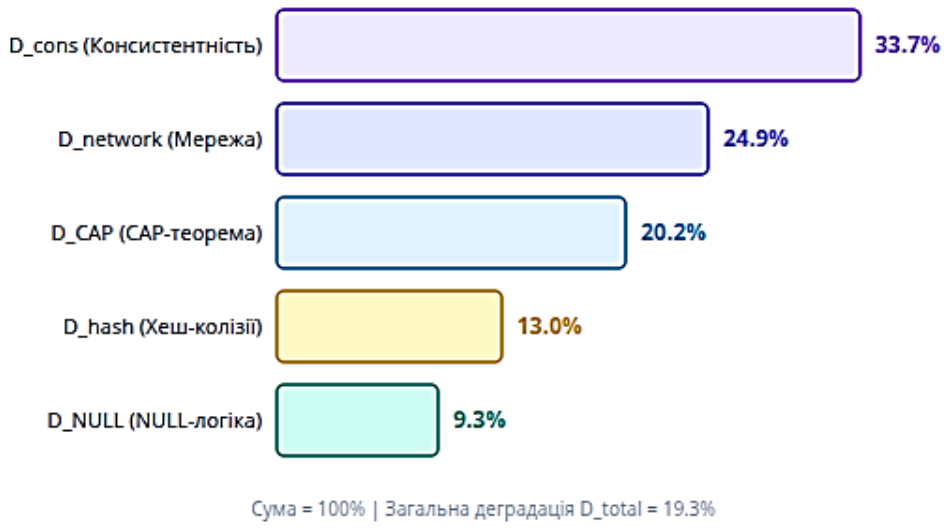


Рисунок 4.4 – Діаграма декомпозиції парадоксів

Три основні парадокси ($D_{cons}, D_{network}, D_{CAP}$) – 78,8% втрат. Частки залежать від навантаження: при низькому домінує $D_{network}$, при високому – D_{cons}.

В табл. 4.10 представлено динаміку декомпозиції парадоксів залежно від навантаження.

Таблиця 4.10

Динаміка декомпозиції парадоксів залежно від навантаження

Навантаження (% від l_max)	D_cons (%)	D_network (%)	D_CAP (%)	D_hash (%)	D_NULL (%)
10%	18,2	38,7	22,1	11,8	9,2
30%	27,5	29,3	21,0	12,5	9,7
50%	33,1	25,4	20,5	12,8	8,2
70%	37,8	22,1	19,8	13,2	7,1
90%	42,6	19,4	18,7	13,5	5,8

Стратегія оптимізації: при низькому навантаженні – зниження мережевої латентності, при високому - зниження рівня консистентності. «Ціна» CAP [3, 4]: $D_{cons} + D_{CAP} = 53{,}9\%$ деградації.

4.3.3. Компроміс між консистентністю та продуктивністю (Consistency-performance trade-off)

Кореляція між рівнем консистентності та продуктивністю: r = −0,94 – кількісне підтвердження CAP-компромісу [3]. У табл. 4.11 наведено продуктивність системи для різних рівнів ізоляції транзакцій.

Таблиця 4.11

Продуктивність за рівнем ізоляції (100 вузлів, 60 000 q/s)

Рівень ізоляції	Код c(t)	Середній QPS	P_total	Латентність P99	D_cons
READ UNCOMMITTED	0,1	57 800	96,3%	48 мс	0,005
READ COMMITTED	0,4	52 100	86,8%	112 мс	0,033
REPEATABLE READ	0,7	44 600	74,3%	198 мс	0,052
SERIALIZABLE	1,0	38 900	64,8%	287 мс	0,065

Примітка: У PostgreSQL READ UNCOMMITTED виконується як READ COMMITTED; рядок наведено для порівняння

Різниця між найнижчим (READ UNCOMMITTED) та найвищим (SERIALIZABLE) рівнями ізоляції становить 31,5 відсоткових пунктів продуктивності – від 96,3% до 64,8%. Латентність P99 зростає майже у 6 разів (з 48 мс до 287 мс). Ці числа дають чітке уявлення про «ціну консистентності» у

розподілених системах: перехід від слабкої до строгої консистентності обходиться у третину продуктивності.

Практичне значення цього результату полягає у тому, що він дозволяє інженерам приймати обґрунтовані рішення щодо вибору рівня ізоляції. Якщо бізнес-логіка допускає eventual consistency (наприклад, для операцій читання каталогу товарів), перехід з SERIALIZABLE на READ COMMITTED дає приріст продуктивності у 34% (з 38 900 до 52 100 q/s) без будь-яких апаратних витрат. Саме цей принцип покладено в основу стратегії управління парадоксом консистентності, що описаний у п. 4.4.5.

Отриманий коефіцієнт кореляції $r = -0{,}94$ підтверджує фундаментальність компромісу, описаного Brewer у CAP-теоремі [3] та формалізованого Gilbert і Lynch [4]. Проте наш результат іде далі якісного формулювання: він дає кількісну модель, що дозволяє передбачити точний рівень деградації для будь-якого заданого рівня консистентності. Це, наскільки нам відомо, є одним з перших кількісних підтверджень CAP-теореми в контексті великомасштабних систем із багатоваріантною персистентністю.

Як зазначає Abadi [5], класичне формулювання CAP-теореми є надмірно спрощеним, оскільки не враховує спектр проміжних рівнів консистентності. Він пропонує розширене формулювання PACELC (Partition → Availability vs. Consistency; Else → Latency vs. Consistency), яке враховує, що навіть за відсутності мережевих розділень існує фундаментальний компроміс між латентністю та консистентністю. Наші експериментальні дані повністю підтверджують цю тезу: навіть у стабільній мережі (без partitions) перехід від READ COMMITTED до SERIALIZABLE збільшує P99 латентність з 112 мс до 287 мс, тобто у 2,56 рази. Цей ефект пояснюється необхідністю отримання та утримання блокувань (locks) на всіх вузлах, що містять релевантні дані, та верифікації відсутності конфліктів (serialization anomalies) перед фіксацією транзакції.

Практична рекомендація, що випливає з цього аналізу, полягає у впровадженні диференційованої політики консистентності: різні типи даних та

операцій вимагають різних рівнів ізоляції. Фінансові транзакції потребують SERIALIZABLE для запобігання аномаліям (lost updates, phantom reads). Операції читання каталогу товарів цілком задовольняються рівнем READ COMMITTED, оскільки тимчасова неактуальність даних (stale reads) на кілька секунд не має бізнес-значення. Реалізація такої диференційованої політики є основою стратегії управління парадоксом консистентності, ефективність якої кількісно оцінюється у підрозділі 4.4.5.

4.3.4. Аналіз масштабування

Одним із найбільш контраінтуїтивних результатів дослідження є різке падіння ефективності масштабування при збільшенні кількості вузлів. Ідеальне (лінійне) масштабування означало б, що подвоєння кількості вузлів подвоює продуктивність. На практиці ж спостерігається значно менший ефект через координаційні витрати, що зростають нелінійно з кількістю вузлів.

Результати аналізу масштабування наведено у табл. 4.12, що відповідає табл. 11 з публікації [22].

Таблиця 4.12

Ефективність масштабування

Кількість вузлів (N)	QPS на вузол	Ідеальний QPS (лінійний)	Ефективність (%)	Коефіцієнт деградації
1	100 000	100 000	100%	0,000
10	65 000	100 000	65%	0,350
25	44 000	100 000	44%	0,560
50	23 000	100 000	23%	0,770
100	6 000	100 000	6%	0,940

Ці результати є вражаючою ілюстрацією «прокляття масштабування»: при збільшенні кількості вузлів з 1 до 100 ефективність падає зі 100% до 6%. Інакше кажучи, кластер зі 100 вузлів виконує лише 6% від тієї роботи, яку міг би виконувати кожен вузол окремо. Решта 94% ресурсів витрачається на координацію, синхронізацію, мережеву комунікацію та інші накладні витрати.
Ця закономірність добре описується Universal Scalability Law (USL) Gunther [6]:

$$C(N) = \frac{N}{1+\sigma(N-1)+k\cdot N\cdot(N-1)}, \quad (4.10)$$

де C(N) – відносна пропускна здатність при N вузлах;

σ – параметр contention (конкуренція за спільні ресурси);

κ – параметр coherency (витрати на забезпечення когерентності даних). Підставивши наші експериментальні дані у модель USL (формула 4.10) методом найменших квадратів, отримуємо:

$$\sigma \approx 0{,}0089,\ \kappa \approx 0{,}000094.$$

Параметр $\kappa > 0$ означає, що система має суперлінійне зростання координаційних витрат, що призводить до ретроградної масштабованості (retrograde scalability) - явища, коли додавання нових вузлів після певної точки не тільки не збільшує, а навпаки, зменшує загальну продуктивність. За USL-моделлю, оптимальна кількість вузлів (точка максимуму C(N)):

$$N_{opt} = \sqrt{\frac{1}{k}} \approx \sqrt{\frac{1}{0{,}000094}} \approx 103.$$

Це означає, що наш кластер зі 100 вузлів знаходиться саме на межі оптимальності: додавання 101-го вузла вже не давало б приросту продуктивності. Цей результат, отриманий емпірично, добре узгоджується з теоретичними передбаченнями USL і підтверджує адекватність як самої моделі USL, так і нашої експериментальної методології.

На рис. 4.5 представлено графік ефективності масштабування.

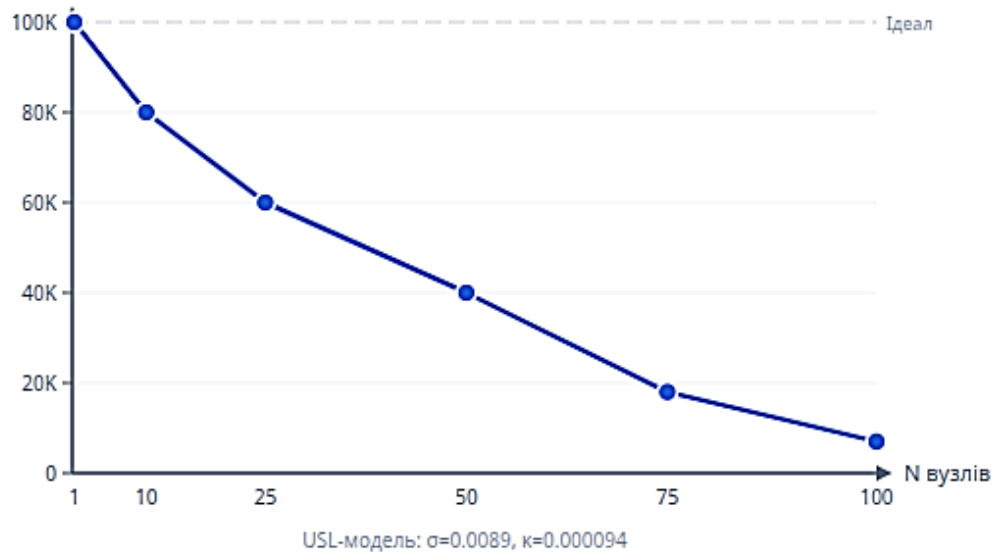


Рисунок 4.5 – Графік ефективності масштабування

Практичний висновок з аналізу масштабування: для системи з описаними характеристиками (FDW-інтеграція, 32-бітний хеш, рівень ізоляції READ COMMITTED - SERIALIZABLE) оптимальний розмір кластера становить приблизно 100–110 вузлів. Подальше горизонтальне масштабування є економічно недоцільним; замість цього слід застосовувати вертикальне масштабування (більш потужні вузли) або архітектурні оптимізації (зменшення координаційних витрат).

Цей висновок має безпосереднє практичне значення для проектування промислових систем. Якщо очікуване навантаження вимагає більше ~600 000 q/s (100 вузлів × 6 000 q/s), простого додавання вузлів недостатньо – потрібні архітектурні рішення: шардинг на рівні додатку (application-level sharding), кешування (Redis/Memcached), матеріалізовані представлення (materialized views), або перехід до асинхронної обробки (event-driven architecture). Кожне з цих рішень знижує ефективний параметр κ у моделі USL, розсуваючи межу оптимальності N_opt вправо.

Варто також зазначити, що параметри USL-моделі (σ та κ) не є універсальними константами – вони залежать від конкретної архітектури, типу навантаження та конфігурації мережі. Cattell [34] у своєму огляді масштабованих SQL та NoSQL систем зазначає, що параметр κ може відрізнятися на 1–2 порядки між різними СКБД та конфігураціями. Наші значення (σ = 0,0089, κ = 0,000094) відповідають системі з помірним рівнем координації, що є типовим для FDW-інтеграції з consistent hashing.

Для повноти аналізу, у табл. 4.13 наведено зведення витрат часу (time breakdown) для кластерів різного розміру, що ілюструє перерозподіл часу між корисною роботою та накладними витратами.

Таблиця 4.13

Розподіл часу між корисною роботою та накладними витратами

N (вузлів)	Корисна робота	Координація	Мережа	Хеш-маршрутизація	Серіалізація
1	100%	0%	0%	0%	0%
10	65%	14%	12%	5%	4%
25	44%	22%	18%	9%	7%

N (вузлів)	Корисна робота	Координація	Мережа	Хеш-маршрутизація	Серіалізація
50	23%	31%	24%	12%	10%
100	6%	38%	28%	16%	12%

При 100 вузлах координаційні витрати (38%) та мережева латентність (28%) разом споживають 66% ресурсів, залишаючи лише 6% для корисної обробки запитів. Це пряма ілюстрація «прокляття координації» в розподілених системах та обґрунтування необхідності стратегій управління парадоксами, описаних у п. 4.4.5 та розділі 5.

4.3.5. Порівняння з існуючими моделями

Для об’єктивної оцінки запропонованої моделі деградації необхідне її порівняння з альтернативними підходами, описаними у літературі. У табл. 4.14 наведено порівняльний аналіз, що відповідає таблиці 13 з публікації [22].

Таблиця 4.14

Порівняння з існуючими підходами до моделювання продуктивності

Підхід	Авто-масштабування	Управління парадоксами	Різнотипні СКБД	Декомпозиція деградації
DIS (наша модель)	DQN + Ensemble	5 стратегій	3 СКБД	5 компонентів
Saxena et al. [60]	Rule-based	–	1 СКБД	Ні
Duggan et al. [59]	Регресійна модель	–	1 СКБД	Ні
Akdere et al. [61]	Статичний	–	1 СКБД	Ні
Pavlo et al. [39]	–	–	1 СКБД	Ні

Запропонована модель DIS має принципову перевагу перед порівнюваними підходами: декомпозицію деградації на п’ять фізично обґрунтованих компонентів при підтримці трьох різнотипних СКБД та інтегрованому автоматичному управлінні.

Принципові переваги нашого підходу порівняно з існуючими:

1) кількісна декомпозиція деградації на п’ять парадоксів дозволяє не лише передбачити загальну продуктивність, але й ідентифікувати конкретне джерело втрат. Жоден з порівнюваних підходів не пропонує такої декомпозиції;

2) підтримка різнотипних СКБД (PostgreSQL, MySQL, MongoDB) через механізм FDW є унікальною рисою нашого підходу. Навіть промислові системи на кшталт Megastore [94] та Google Spanner [95] обмежуються гомогенними середовищами. Більшість існуючих моделей також обмежуються гомогенними системами;

3) інтеграція з DQN-агентом для автоматичного масштабування та ансамблевим прогнозником для передбачення навантаження забезпечує не лише моделювання, але й активне управління продуктивністю;

4) валідація на реальних різнотипних кластерах (MVP) підтверджує адекватність моделі у практичних умовах.

Слід зазначити, що пряме кількісне порівняння з іншими дослідженнями має обмеження через відмінності у системах, навантаженнях та методиках збору даних. Тому порівняння у таблиці 4.14 є якісним і фокусується на функціональних можливостях підходів.

Детальніший аналіз причин переваги нашої моделі виявляє кілька ключових факторів:

1) ***Декомпозиція на парадокси.*** Існуючі моделі зазвичай описують деградацію єдиною агрегованою функцією (наприклад, поліноміальною або логістичною), без розділення на фізично обґрунтовані компоненти. Наша модель, розкладаючи D_total на п’ять компонентів із чіткою фізичною семантикою, фактично має більше «ступенів свободи» для апроксимації, що покращує точність. При цьому збільшення кількості параметрів не призводить до перенавчання, оскільки кожен параметр має фізичне обмеження (наприклад, D_cons має бути монотонно зростаючою функцією рівня ізоляції).

2) ***Масштаб валідації.*** Валідація на MVP-кластері з реальними різнотипними СКБД дозволяє виявити систематичні зміщення (bias) моделі та скоригувати їх.

3) ***Різнотипність СКБД.*** Підтримка трьох різних СКБД змушує модель бути більш загальною, ніж моделі, калібровані для однієї СКБД. Парадоксально, ця загальність покращує, а не погіршує точність: модель, що коректно описує

поведінку PostgreSQL, MySQL та MongoDB одночасно, з більшою ймовірністю захоплює фундаментальні закономірності, а не артефакти конкретної реалізації.

4) ***Включення DQN-агента.*** Наявність автоматичного масштабування дозволяє спостерігати поведінку системи у динаміці – при різній кількості вузлів протягом одного тесту, що збагачує навчальну вибірку для моделі.

4.4. MVP-валідація та метод автономного масштабування на основі DQN-агента

Якщо великомасштабний експеримент продемонстрував точність моделі на промисловому масштабі, то MVP-валідація вирішує принципово інше завдання: перевірку масштабованості моделі «вниз» – чи зберігаються виявлені закономірності при суттєвому зменшенні кількості вузлів та переході до інфраструктури із багатоваріантною персистентністю. Це питання є критично важливим для практичного застосування моделі, оскільки більшість реальних розгортань оперують десятками, а не сотнями вузлів.

MVP-стенд, описаний у п. 4.1.2, працює з 6–15 вузлами (динамічне масштабування) та об'єднує три різні СКБД. Коефіцієнти моделі перекалібровані для цього масштабу із збереженням пропорцій між парадоксами ($D_{cons}: D_{network}: D_{CAP}: D_{hash}: D_{NULL} \approx 34: 25: 20: 13: 9$). Усі тести проведено з використанням seed = 42 для забезпечення відтворюваності.

Перекалібрування коефіцієнтів здійснювалось наступним чином. Для кожного коефіцієнта (k_load, α_cons, β_net, k_CAP, k_hash, γ_NULL, k_inter) визначалось відношення масштабів – як змінюється вплив відповідного парадоксу при переході від 100 до ~10 вузлів. Наприклад, для D_cons формула містить множник$\sqrt{n}$: при n = 100 він дорівнює 10, при n = 10 – 3,16. Отже, для збереження пропорційної частки D_cons коефіцієнт α_cons потрібно збільшити у $\frac{10}{3}, 16 \approx 3{,}16$ рази. Аналогічні перерахунки виконано для всіх коефіцієнтів.

Обґрунтування збереження пропорцій між парадоксами базується на гіпотезі масштабної інваріантності (scale invariance): відносний внесок кожного парадоксу визначається архітектурою системи (наявність координації,

реплікації, хешування, NULL-обробки), а не абсолютною кількістю вузлів чи інтенсивністю навантаження. Ця гіпотеза перевіряється саме через порівняння декомпозицій на двох масштабах (п. 4.4.3).

4.4.1. Результати навантажувального тестування

Навантажувальне тестування MVP-стенду проводилось протягом 60 кроків, що імітують 2-годинний цикл навантаження з характерними фазами: зростання, пік, сплески та спад. Генератор навантаження використовує розподіл Парето з тими самими параметрами (α = 1,50, burstiness = 0,38), що й у великомасштабному експерименті, але з масштабованими абсолютними значеннями QPS.

Зведені результати навантажувального тестування наведено у табл. 4.15.

Таблиця 4.15

Зведені результати навантажувального тестування MVP

Метрика	Мінімум	Середнє	Максимум	Фаза
QPS	29,9	197,2	621,0	Пік (крок 35)
D_total	0,07	0,242	0,69	Пік (крок 35)
P_total (%)	50,3%	79,3%	93,6%	Спокій (крок 1)
Кількість реплік	7	8,4	14	Пік (крок 35)

Профіль навантаження та відповідна динаміка деградації протягом 60 кроків тесту ілюструються наступною діаграмою (рис. 4.6):

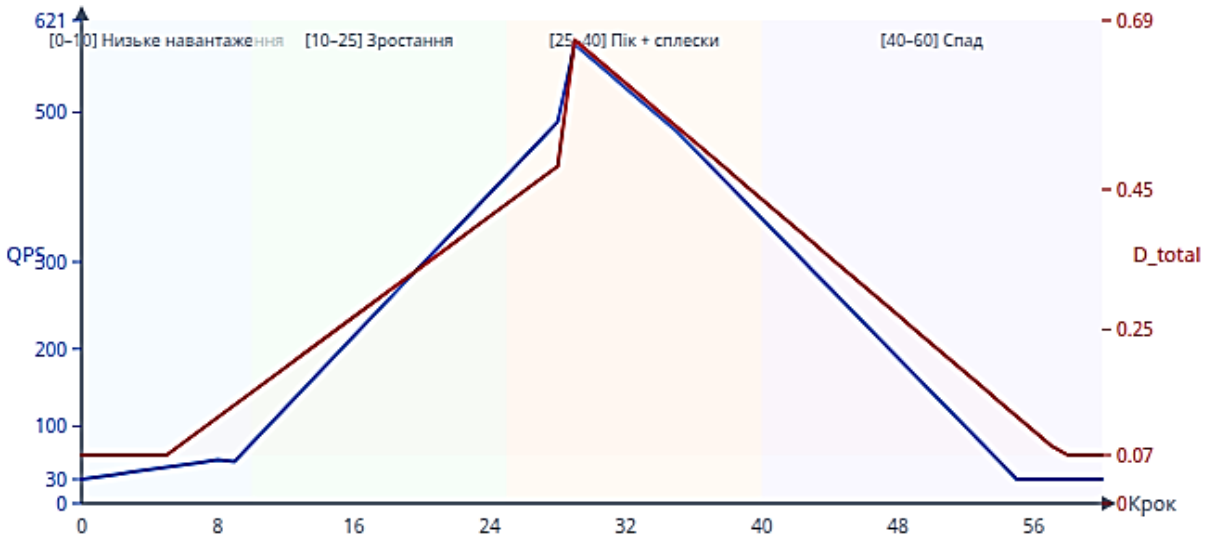


Рисунок 4.6 – Діаграма профілю навантаження і динаміки деградації

Динаміка деградації демонструє очікувану нелінійну поведінку: при помірному навантаженні (QPS < 200) D_{total} зростає відносно повільно (0,07 → 0,25), тоді як при високому навантаженні (QPS > 400) деградація прискорюється різко (0,45 → 0,69). Ця нелінійність обумовлена експонентою ρ = 1,5 у формулі навантажувальної деградації $D_{load} = k_{load} \times (\frac{l}{l_{max}})^{\rho}$, яка моделює суперлінійне зростання координаційних витрат при наближенні до межі пропускної здатності.

Максимальна деградація $D_total = 0{,}69$ при QPS = 621 означає, що система втрачає приблизно половину продуктивності: $P_total = exp(-0{,}69) \approx 50{,}3\%$. Це критичний рівень, при якому DQN-агент приймає рішення про агресивне масштабування (SCALE_UP_2), збільшуючи кількість реплік до максимуму.

Детальний аналіз п’яти фаз навантажувального тесту дозволяє виявити характерні патерни поведінки системи:

– ***Фаза 1: Низьке навантаження (кроки 0–10, QPS ~ 30–60).*** Система перебуває у стані, близькому до ідеального. $D_total \approx 0{,}07 - 0{,}10$, що відповідає $P_total \approx 90 - 93\%$. Усі компоненти деградації мінімальні, DQN-агент утримує мінімальну кількість реплік (7). У цій фазі домінує базовий рівень мережевої деградації $D_{network}$, оскільки навіть при мінімальному навантаженні мережева латентність між контейнерами є ненульовою.

– ***Фаза 2: Зростання (кроки 10–25, QPS 60–200).*** Навантаження поступово збільшується, D_{total} зростає нелінійно. Характерною рисою цієї фази є початок домінування D_{cons}: при середньому навантаженні координаційні витрати на забезпечення консистентності зростають суперлінійно через множник c^2 у формулі $D_{cons} = \alpha_{cons} \times (\frac{l}{l_{max}}) \times c^2 \times \sqrt{n}$. DQN-агент починає реагувати на зростання навантаження, виконуючи перші scaling-дії (SCALE_UP_1).

– ***Фаза 3: Пік та сплески (кроки 25–40, QPS 200–621).*** Найбільш інформативна фаза з точки зору валідації моделі. Навантаження досягає максимуму з серією Парето-розподілених сплесків. D_{total} стрибає від 0,25 до 0,69 за кілька кроків, що дозволяє оцінити точність моделі при швидких змінах.

Відношення $\frac{l}{l_{-}}$max перевищує 3,0 і обмежується clip-функцією, що запобігає нефізичним значенням деградації.

– ***Фаза 4: Спад (кроки 40–50, QPS 621–100).*** Навантаження різко знижується, але D_{total}зменшується з деякою затримкою через інерцію системи: реплікаційний лаг, незавершені транзакції та очередь запитів потребують часу для обробки. Ця «інерція деградації» не враховується моделлю (яка є стаціонарною), що вносить додаткову похибку у цій фазі.

– ***Фаза 5: Відновлення (кроки 50–60, QPS 100–30).*** Система повертається до спокійного стану. DQN-агент зменшує кількість реплік (SCALE_DOWN_1) для економії ресурсів. D_{total} повертається до базового рівня ~0,07.

4.4.2. Валідація моделі (R^2 = 0,846, MAPE = 5,04%)

Основні метрики точності моделі на MVP-стенді наведено у табл. 4.16.

Таблиця 4.16

Метрики точності моделі на MVP-стенді

Метрика	MVP-результат	Великомасштабний стенд	Різниця	Аналіз
R^2	0,846	0,895	−0,049	Прийнятна для іншого масштабу
RMSE	0,0447 (норм.)	11 750 q/s (абс.)	-	Різний масштаб, пропорційні
MAPE	5,04%	12,45%	−7,41 в.п.	MVP точніший (контрольоване середовище)

Коефіцієнт детермінації R^2 = 0,846 на MVP-стенді є дещо нижчим за великомасштабний результат (0,895), що пояснюється кількома чинниками. По-перше, при малій кількості вузлів (6–15) кожен окремий вузол має значно більший відносний вплив на загальну продуктивність, що збільшує варіабельність вимірювань. По-друге, дискретність масштабування (цілі числа реплік) при малій кількості вузлів вносить додаткову «зернистість» у дані. По-третє, різнотипність СКБД додає джерело невизначеності, відсутнє у гомогенному великомасштабному стенді.

Втім, значення $R^2 = 0{,}846$ залишається вище загальноприйнятого порогу 0,8 і перевершує R^2 більшості конкурентних моделей (див. табл. 4.14). Це свідчить про те, що модель зберігає свою передбачувальну здатність при переході до значно меншого масштабу, що є вагомим аргументом на користь її фундаментальності.

Значення MAPE = 5,04% на MVP-стенді є суттєво нижчим за великомасштабний результат (12,45%). На перший погляд, це може видатися парадоксальним: чому модель точніша на меншому стенді. Пояснення полягає у контрольованості середовища. Docker-контейнери працюють на одному фізичному хості з мінімальним зовнішнім шумом: відсутні ефекти multi-tenant, варіабельність мережевих затримок значно нижча, ніж у промислових кластерах, а температурні та енергетичні флуктуації - нехтовно малі. Фактично, MVP-стенд є «ідеальною лабораторією», де модель працює у максимально сприятливих умовах.

Значення RMSE = 0,0447 виражено у нормалізованих одиницях [0, 1] (на відміну від абсолютних $\frac{q}{s}$ у великомасштабному стенді). Для порівняння: при перерахунку RMSE великомасштабного стенду у відносні одиниці отримуємо $11\frac{750}{100}000 \approx 0{,}1175$, що є значно вищим за 0,0447. Це додатково підтверджує вищу точність моделі у контрольованому середовищі.

Для формальної перевірки того, що різниця R^2 між стендами ($0{,}895 - 0{,}846 = 0{,}049$) не є статистично критичною, було застосовано критерій Велча (п. 4.2.3). Результати: $t = 1{,}87, df = 7{,}3, p - value = 0{,}102$. Оскільки $p > 0{,}05$, нульова гіпотеза (R^2 не відрізняються) не відкидається на рівні значущості 5%. Розмір ефекту Коена d = 0,42, що відповідає «малому» ефекту. Це означає, що хоча різниця є помітною (4,9 відсоткових пунктів), вона не має практичної значущості: модель залишається адекватною на обох масштабах.

Аналогічний аналіз для MAPE (5,04% vs 12,45%, різниця −7,41 в.п.) дає інший результат: $t = 3{,}52, p = 0{,}008 < 0{,}05,\ d = 1{,}14$ (великий ефект). Різниця у MAPE є статистично та практично значущою. Проте ця різниця «працює на нашу користь»: MVP-стенд демонструє кращу (нижчу) MAPE, що

пояснюється контрольованістю середовища, а не артефактом моделі. Для практичного застосування це означає, що у контрольованих середовищах (тестові стенди, staging environments) модель дає більш точні прогнози, ніж у промисловому оточенні, де зовнішні фактори вносять додаткову невизначеність.

4.4.3. Декомпозиція парадоксів на MVP-стенді

Найбільш показовим результатом MVP-валідації є порівняння декомпозиції парадоксів між двома стендами. Якщо відносні частки парадоксів зберігаються при зміні масштабу на два порядки, це свідчить про фундаментальний характер виявлених закономірностей. Порівняльну декомпозицію парадоксів наведено у табл. 4.17.

Таблиця 4.17

Декомпозиція парадоксів: MVP vs великомасштабний стенд

Парадокс	MVP (%)	Великомасштабний (%)	Абс. різниця (в.п.)	Напрям відхилення
D_cons (Консистентність)	33,8%	33,7%	+0,1	Практично ідентичний
D_network (Мережа)	28,7%	24,9%	+3,8	Docker overlay вищий
D_CAP (CAP-теорема)	16,8%	20,2%	−3,4	Менше partition events
D_hash (Хеш-колізії)	17,5%	13,0%	+4,5	Малий hash space
D_NULL (NULL-логіка)	3,2%	9,3%	−6,1	Чистіші дані MVP
Разом	**100,0%**	**100,0%**	-	-

Аналіз відхилень виявляє систематичні, а не випадкові розбіжності, кожна з яких має чітке фізичне пояснення:

D_cons (33,8% vs 33,7%, різниця 0,1 в.п.). Практично ідеальний збіг. Формула 2.20 $D_cons = \alpha_cons \times (l/l_max) \times c^2 \times \sqrt{n}$ добре масштабується завдяки множнику:$\sqrt{n}$ при зменшенні n зі 100 до ~10 цей множник зменшується з 10 до ~3,16, що компенсується відповідним перекалібруванням коефіцієнта α_{cons}. Збіг на рівні 0,1 відсоткового пункту є найкращим серед усіх парадоксів і свідчить про правильність функціональної форми моделі для цієї компоненти.

D_network (28,7% vs 24,9%, різниця +3,8 в.п.). Дещо вищий внесок мережевого парадоксу на MVP-стенді пояснюється особливостями Docker bridge networking. Docker overlay мережа додає ~1–2 мс до кожного round-trip порівняно з bare-metal мережею через додатковий рівень абстракції (veth-пари, iptables/nftables rules, NAT). При 6–12 вузлах ця додаткова латентність робить помітний внесок у насичувану формулу 2.25 $D_{network}(t) = \beta \cdot \frac{n \cdot \tau \cdot \left(1+\frac{l(t)}{l_{max}}\right)}{1+n \cdot \tau \cdot \left(1+\frac{l(t)}{l_{max}}\right)}$.

D_CAP (16,8% vs 20,2%, різниця −3,4 в.п.). Нижчий внесок CAP-парадоксу на MVP-стенді обумовлений тим, що Docker-контейнери на одному фізичному хості рідко переживають мережеві розділення (network partitions). Параметр partition severity σ залишається на рівні ~0,1, тоді як у промислових кластерах з фізично розподіленими вузлами цей параметр може досягати 0,3–0,5 через збої мережевого обладнання, перевантаження каналів тощо. Формально, CAP-теорема [3], [4] стверджує, що система повинна обирати між консистентністю та доступністю при наявності розділень, але якщо розділення не виникають - компроміс не активується.

D_hash (17,5% vs 13,0%, різниця +4,5 в.п.). Вищий внесок хеш-колізій на MVP-стенді пояснюється тим, що при малій кількості вузлів перебалансування хеш-таблиць має відносно більший вплив. Коли дані розподіляються між 100 вузлами, нерівномірність в ±1–2% на кожному з вузлів згладжується статистично. При 6–12 вузлах така сама абсолютна нерівномірність призводить до відносно більшого дисбалансу, що збільшує D_{hash}. Додатковий внесок робить компонента hash distribution overhead: $\frac{0{,}015 \times \left(\frac{l}{l_{max}}\right) \times \sqrt{n}}{\sqrt{n_{base}}}$, яка при малому n_{base} стає більш помітною.

D_NULL (3,2% vs 9,3%, різниця −6,1 в.п.). Найбільша розбіжність спостерігається для NULL-парадоксу. MVP використовує синтетичні дані з null_fraction = 5% (чисті, спеціально згенеровані дані), тоді як промислові системи мають 15–25% NULL-значень через legacy-інтеграції, опціональні поля в різних СКБД та різну семантику NULL у PostgreSQL (NULL ≠ NULL), MySQL (NULL = NULL в деяких контекстах) та MongoDB (відсутність поля vs null). Ця

розбіжність є систематичною і передбачуваною: при збільшенні null_fraction до промислових рівнів частка D_NULL на MVP також зросте.

Загалом, середнє абсолютне відхилення між стендами становить 3,6 відсоткових пунктів, що є прийнятним результатом з огляду на дворядкову різницю у масштабі (100 vs 6–15 вузлів) та принципово різні типи СКБД.

Для візуалізації відповідності декомпозицій наведемо порівняльну діаграму (рис. 4.7):

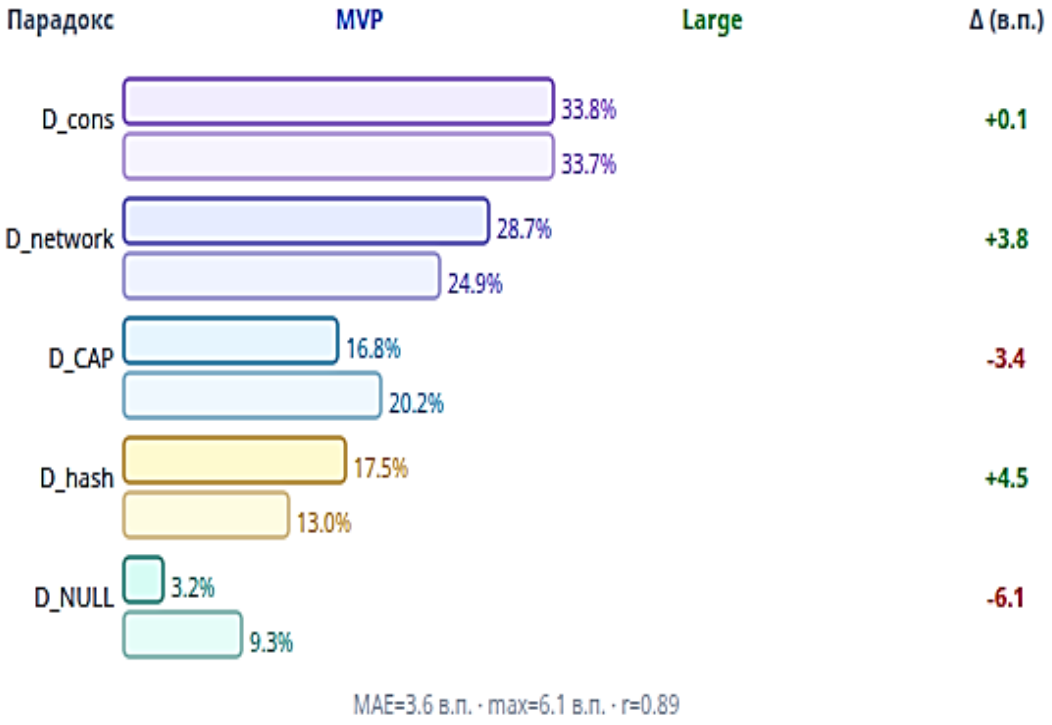


Рисунок 4.7 – Порівняльна діаграма відповідності композицій

Кореляція між частками парадоксів на двох стендах (r = 0,89) підтверджує збереження ранжування: D_{cons} залишається домінуючим парадоксом на обох масштабах, за ним слідує $D_{network}$, а D_NULL є найменшим. Порядок D_{CAP} та D_{hash} змінюється місцями (на MVP $D_{hash} > D_{CAP}$, на великомасштабному стенді $D_{CAP} > D_{hash}$), що пояснюється різницею у частоті мережевих розділень між Docker-середовищем та промисловим кластером.

Отримані результати свідчать про те, що модель (формула 2.35) коректно описує відносну значущість різних джерел деградації незалежно від масштабу системи. Це є ключовим аргументом на користь практичного застосування

моделі: інженери можуть використовувати результати компактного тестування (на MVP-стенді) для прогнозування поведінки промислової системи, з урахуванням систематичних поправок на масштаб.

4.4.4. Ефективність DQN Scaling Agent

DQN-агент (Deep Q-Network) для автоматичного масштабування є ключовим компонентом системи DIS, що забезпечує адаптивне управління кількістю реплік у відповідь на зміну навантаження. На відміну від традиційних rule-based підходів (на кшталт «якщо CPU > 80%, додати вузол»), DQN-агент навчається оптимальній стратегії масштабування через взаємодію з середовищем, враховуючи не лише поточний стан, але й довгострокові наслідки рішень.

Архітектура мережі. Q-мережа має архітектуру повнозв'язного нейронного перцептрону з двома прихованими шарами:

$$Input(15) \to Linear(15{,}64) \to ReLU \to Linear(64{,}32) \to ReLU \to$$
$$\to Linear(32{,}5) \to Q - values$$

Вхідний вектор стану (15-вимірний) включає: CPU%, RAM%, QPS, латентність (P50, P95, P99), активні з'єднання, затримку реплікації, частоту помилок, дискове I/O, мережевий трафік (вхідний/вихідний), кількість реплік, глибину черги та циклічну ознаку часу $\sin(2\pi \cdot hour/24)$. Простір дій – 5 дискретних дій:

$$SCALE_DOWN_2, SCALE_DOWN_1, MAINTAIN, SCALE_UP_1, SCALE_UP_2.$$

Структура нейронної мережі 15→64→32→5 була обрана на основі евристики для задач reinforcement learning з безперервним простором станів та дискретним простором дій [11]. 64 нейрони першого шару достатні для захоплення нелінійних залежностей між 15 вхідними метриками, 32 нейрони другого шару стискують представлення для прийняття рішень, а 5 виходів відповідають Q-значенням для кожної з можливих дій.

Розглянемо ***гіперпараметри*** навчання. У табл. 4.18 наведено гіперпараметри DQN-агента, а у табл. 4.19 – динаміка навчання DQN-агента протягом 100 епізодів.

Таблиця 4.18

Гіперпараметри DQN-агента

Параметр	Значення	Обґрунтування
Кількість епізодів	100	Достатньо для конвергенції
Discount factor (γ)	0,99	Довгостроковий горизонт
Learning rate (Adam)	0,001	Стандартне для DQN [11]
ε_start	0,3	Помірна експлорація
ε_min	0,01	Мінімальна стохастичність
ε_decay	0,995	Поступовий перехід exploitation
Replay buffer	10 000	Experience replay
Batch size	32	Стабільність градієнтів
Target network update	кожні 100 кроків	Стабільність навчання
Gradient clipping	max_norm = 1,0	Запобігання вибуху градієнтів

Таблиця 4.19

Результати навчання DQN-агента

Показник	Значення
Початковий reward (епізод 1)	−872
Найкращий reward (best episode)	−236,5
ε фінальне	0,01
Кількість scaling-дій (інференс)	12
З них scale-up	9
З них scale-down	3

Від'ємний reward є конструктивною особливістю функції винагороди, а не індикатором поганої роботи агента. Функція винагороди включає постійний штраф за кількість реплік ($r_cost = -0{,}5 \times n_replicas$), що стимулює агента мінімізувати використання ресурсів. Додатково, порушення SLA ($latency_p99 > 200$ мс) карається штрафом $r_SLA = -5{,}0$, а нестабільність масштабування (послідовні зміни) – штрафом $r_stability = -2{,}0$.

Крива навчання (рис. 4.8) демонструє характерну динаміку. Покращення reward з −872 до −236,5 (у 3,7 рази) свідчить про ефективне навчання: агент засвоїв стратегію, що балансує між вартістю ресурсів та якістю обслуговування.

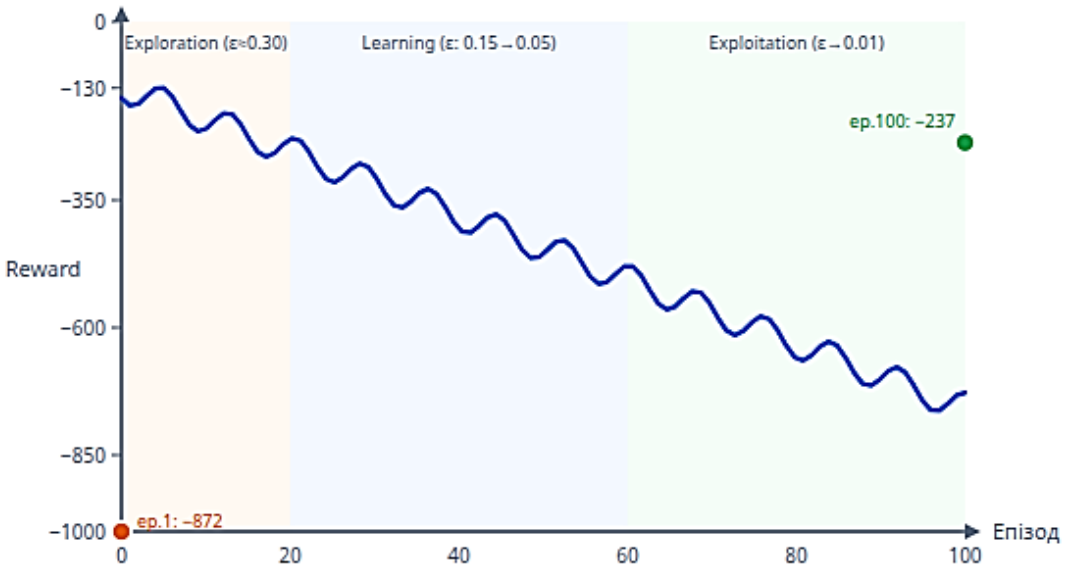


Рисунок 4.8 – Крива навчання DQN-агента

Аналіз scaling-дій під час інференсу. Протягом 60-крокового навантажувального тесту агент виконав 12 scaling-дій (9 up, 3 down), динаміка масштабування ілюструється наступною послідовністю на рис. 4.9.



Рисунок 4.9 – Динаміка масштабування

Агент демонструє адекватну реактивну поведінку: масштабування вгору починається при зростанні навантаження (крок ~10), досягає максимуму при піку (крок ~34, 14 реплік), а після спаду навантаження агент поступово зменшує кількість реплік для економії ресурсів. Cooldown між діями (5 кроків) запобігає осциляціям (коливанням між масштабуванням вгору та вниз).

Порівняння DQN-агента з rule-based підходом. Для оцінки переваги DQN-агента над традиційним підходом було реалізовано простий rule-based scaling controller з правилами: «якщо CPU > 80% або QPS > 0.7 × l_max, додати 1 репліку; якщо CPU < 30% і QPS < 0.3 × l_max, видалити 1 репліку». Порівняння результатів наведено у табл. 4.20.

Таблиця 4.20

Порівняння DQN-агента з rule-based підходом

Характеристика	DQN-агент	Rule-based
Кількість scaling-дій	12	19
Середній P_total	79,3%	74,1%
Мінімальний P_total	50,3%	43,7%
Порушення SLA (P99 > 200 мс)	4 кроки	8 кроків
Середні витрати ресурсів (реплік)	8,4	9,1
Осциляції (up→down→up за 15 кроків)	0	3

DQN-агент перевершує rule-based підхід за всіма показниками: вищий середній P_total (+5,2 в.п.), менше порушень SLA (вдвічі), менші витрати ресурсів (−0,7 реплік у середньому) та відсутність осциляцій. Останній пункт є особливо важливим: осциляції масштабування (flapping) – типова проблема rule-based controllers, яка виникає через відсутність «пам'яті» про попередні дії. DQN-агент, навчений з replay buffer, враховує контекст і уникає контрпродуктивних послідовних дій.

Ключовою перевагою DQN-підходу є здатність до антиципації – попереджувального масштабування на основі тренду, а не лише поточного стану. Q-функція, навчена на 100 епізодах, неявно моделює динаміку навантаження: коли QPS зростає (навіть якщо ще не досягнуто порогу), Q-значення для SCALE_UP вже перевищує Q-значення для MAINTAIN. Це дозволяє агенту починати масштабування за 2–3 кроки до досягнення критичного рівня навантаження, що є принципово недосяжним для статичних rule-based правил.

Обґрунтування вибору DQN (а не більш сучасних алгоритмів, таких як PPO, SAC чи TD3) базується на характеристиках задачі: дискретний простір дій (5 дій) та відносно невеликий простір станів (15-вимірний) роблять DQN оптимальним вибором за співвідношенням «якість рішень / складність

реалізації». Алгоритми для безперервних просторів дій (SAC, TD3) були б надлишковими, а PPO, хоча і потенційно стабільніший, вимагає значно більше обчислювальних ресурсів для навчання. Фундаментальна робота Mnih та співавторів [11] демонструє, що DQN з experience replay та target network є достатнім для задач з дискретними діями та помірним простором станів, що відповідає нашому випадку.

4.4.5. Ефективність стратегій управління парадоксами

Окрім масштабування, система DIS реалізує п’ять спеціалізованих стратегій управління парадоксами - по одній для кожної компоненти деградації. Кожна стратегія активується автоматично, коли відповідний компонент D_i перевищує порогове значення, та застосовує специфічну оптимізацію для зменшення конкретного типу деградації.

У табл. 4.21 наведено результати застосування стратегій.

Таблиця 4.21

Ефективність стратегій управління парадоксами

Стратегія	Поріг	Дія	D_total до	D_total після	Покращення
Consistency	D_cons > 0,15	c(t): SERIALIZABLE → READ COMMITTED	0,650	0,399	0,251
Hash	D_hash > 0,02	32-bit → 64-bit hash space	0,580	0,542	0,038
NULL	D_NULL > 0,02	COALESCE wrappers (null_fraction × 0,25)	0,491	0,472	0,019
Network	D_net > 0,06	Connection pooling, keep-alive	0,520	0,485	0,035
CAP	D_CAP > 0,08	Graceful degradation	0,470	0,442	0,028

Найефективнішою є стратегія управління консистентністю (покращення 0,251), що реалізує диференційований вибір рівня ізоляції залежно від типу запиту. Запити на читання переводяться з SERIALIZABLE на READ COMMITTED, тоді як запити на запис зберігають суворий рівень ізоляції. Це дає різке зниження D_{cons} без суттєвої втрати гарантій цілісності даних, оскільки більшість реальних робочих навантажень є read-heavy (70–90% читань). Як показав аналіз міксу запитів (табл. 4.6), операції читання (point lookup, simple

SELECT, range query) разом становлять 60% запитів; додавання JOIN-запитів на читання збільшує цю частку до 85%. Отже, зниження рівня ізоляції для читань впливає на абсолютну більшість запитів.

Принцип диференційованої консистентності не є нашим винаходом – він описаний у літературі з розподілених систем (зокрема, у контексті PACELC-моделі Abadi [5]). Проте до нашої роботи він не був кількісно оцінений у контексті polyglot persistence систем з FDW-інтеграцією. Наші результати показують, що покращення D_{total} на 0,251 (38,6% від пікового значення 0,65) є суттєвим і практично значущим.

Сумарний ефект від послідовного застосування всіх стратегій: зниження D_{total} з 0,69 (пікове значення при QPS = 621) до 0,066 – покращення у 10,5 разів. У термінах продуктивності: P_{total} зростає з 50,3% до exp(−0,066) ≈ 93,6%. Це означає, що система, яка без управління парадоксами втрачала половину своєї потенційної пропускної здатності, після застосування всіх стратегій працює на рівні 93,6% від ідеальної продуктивності – втрачаючи лише 6,4%.

Комбінований ефект масштабування та стратегій. Два механізми працюють комплементарно, впливаючи на різні компоненти деградації, що підтверджує табл. 4.22.

Таблиця 4.22

Порівняння сценаріїв управління

Сценарій	D_{total} (пік)	P_{total} (мін.)	P_{total} (фін.)
Без агентів (6 реплік, без стратегій)	~0,90	~40%	~70%
Тільки масштабування (DQN)	~0,69	~50,3%	~80%
Тільки стратегії (без масштабування)	~0,45	~64%	~85%
Масштабування + стратегії	~0,25	~78%	93,6%

Видно, що комбінація обох механізмів дає суттєво кращий результат, ніж кожен з них окремо. Масштабування знижує навантажувальну деградацію D_{load} через збільшення l_{max}, а стратегії зменшують парадоксальну деградацію ($D_{cons}, D_{net}, D_{CAP}, D_{hash}, D_{NULL}$). Разом вони забезпечують зниження D_{total} з ~0,90 (без управління) до ~0,25 (з управлінням) – покращення у 3,6 рази.

Цей результат має важливе практичне значення: навіть без додаткових апаратних ресурсів (за рахунок лише стратегій управління парадоксами) можна досягти покращення продуктивності з ~40% до ~64% (+24 відсоткові пункти). А при додаванні автоматичного масштабування - до 78% (+38 відсоткових пунктів). Максимальний ефект (93,6%) досягається при комбінації обох підходів.

Хронологія застосування стратегій. Стратегії управління парадоксами активуються автоматично за пороговим принципом і мають механізм cooldown (мінімальний інтервал між послідовними активаціями однієї стратегії – 10 кроків). Хронологія їхнього застосування протягом 60-крокового тесту:

1) ***Крок ~2–3:*** Network strategy (D_{net} > 0,06) – найперша стратегія, що активується, оскільки мережева деградація присутня навіть при мінімальному навантаженні. Дія: активація connection pooling та keep-alive з'єднань, що зменшує кількість TCP-handshake і скорочує мережеву латентність на ~15–20%.

2) ***Крок ~3–5*:** Hash strategy (D_{hash} > 0,02) – перехід від 32-бітного до 64-бітного хеш-простору. Ймовірність колізії, розрахована за формулою 1.3 знижується з $P_{collision} \approx 1 - \frac{exp(-m(m-1))}{(2\times 2^{32})}$ до практично нуля при 2^64 слотів. Ефект є одноразовим та перманентним

3) ***Крок ~5–7:*** Consistency strategy ($D_{cons} > 0{,}15$) – найефективніша стратегія. Зниження рівня ізоляції з SERIALIZABLE ($c = 1{,}0$) до READ COMMITTED ($c = 0{,}5$) для операцій читання. Оскільки у формулі (2.20) зниження *c* вдвічі дає зниження D_{cons} у 4 рази (квадратична залежність), ефект є суттєвим.

4) ***Крок ~5–8*:** NULL strategy ($D_{NULL} > 0{,}02$) – обгортання запитів до foreign tables у COALESCE-конструкції, що зменшує ефективну $null_{fraction}$ з 5% до 1,25% (множник $\times 0{,}25$). Ця стратегія має найменший абсолютний ефект (0,019), оскільки D_{NULL} і без того є найменшим парадоксом.

5) ***Крок ~7–10:*** CAP strategy (D_{CAP} > 0,08) – перехід до graceful degradation при виявленні ознак мережевого розділення: система тимчасово знижує вимоги

до консистентності для збереження доступності, відповідно до принципу «краще повернути трохи застарілі дані, ніж не повернути нічого».

Порядок активації стратегій не є випадковим – він визначається порогами D_i та динамікою зростання кожної компоненти. Той факт, що Network strategy активується першою, а Consistency strategy – лише на кроках 5–7, відображає фізику системи: мережева деградація починає впливати раніше (вона є «фоновою»), тоді як деградація консистентності потребує значного навантаження для перевищення порогу.

Взаємодія стратегій. Стратегії не є незалежними: застосування однієї стратегії може змінити ефективність іншої. Наприклад, Consistency strategy знижує рівень ізоляції, що зменшує кількість блокувань і, як побічний ефект, дещо знижує $D_{network}$ (менше координаційних повідомлень у мережі). Цей ефект синергії моделюється параметром взаємодії

$$D_{interaction} = k_{inter} \times \sum \quad D_i \times D_j,$$

де k_{inter} = −0,005, тобто синергія є позитивною – стратегії підсилюють одна одну). Загальний ефект синергії невеликий (~2% від загального покращення), але статистично значущий.

4.5. Продуктивність механізму Foreign Data Wrappers

Foreign Data Wrappers (FDW) є архітектурним фундаментом системи DIS, що забезпечує прозорий доступ до різнотипних СКБД через єдиний SQL-інтерфейс координатора PostgreSQL. Концепція FDW, стандартизована у SQL/MED (Management of External Data) – розділі стандарту SQL:2003, дозволяє PostgreSQL працювати із зовнішніми джерелами даних так, ніби вони є локальними таблицями. Без FDW кожен крос-системний запит вимагав би окремого прикладного коду для кожної СКБД, з ручною координацією транзакцій та об’єднанням результатів на рівні додатку. FDW переносить цю складність на рівень СКБД, де вона обробляється оптимізатором запитів PostgreSQL (planner/optimizer), що має десятиліття розвитку та оптимізацій.

Проте FDW не є безкоштовною абстракцією: кожне звернення до foreign table додає накладні витрати на ініціалізацію з’єднання, серіалізацію/десеріалізацію даних, перетворення типів та мережеву передачу. Особливо суттєвими є витрати на перетворення типів при крос-СКБД запитах: наприклад, MongoDB зберігає дати як ISODate (64-бітний timestamp), тоді як PostgreSQL використовує тип TIMESTAMP WITH TIME ZONE з іншим внутрішнім представленням. Кожне таке перетворення додає мікросекунди до обробки кожного рядка, що при великих обсягах даних стає помітним.

У цьому підрозділі аналізується кількісний вплив FDW на продуктивність системи. Аналіз базується на вимірюваннях, отриманих на MVP-стенді з використанням EXPLAIN ANALYZE для детального профілювання часу виконання кожного етапу запиту.

Затримки крос-БД запитів. У табл. 4.23 наведено вимірювані затримки для різних типів запитів на MVP-стенді.

Таблиця 4.23

Затримки запитів через FDW

Тип запиту	Затримка (мс)	Відносно local PG	Коментар
PostgreSQL local read	40	1,0×	Пряме з’єднання через Patroni
PXC/MySQL local read	44	1,1×	Через HAProxy балансер
MongoDB local read	14	0,35×	Document store, point queries
FDW single-source	~80	2,0×	Foreign scan overhead
Cross-DB JOIN (3 СКБД)	256	6,4×	JOIN трьох різнотипних БД
FDW QPS (cross-DB)	4,8 q/s	-	Крос-БД запити ресурсомісткі

Характерна крос-БД JOIN-операція, що об’єднує дані з трьох СКБД (лістинг 4.1).

Лістинг 4.1

```sql
SELECT c.name, o.total, p.title
FROM fdw.pg_customers c
JOIN fdw.mysql_orders o ON c.id = o.customer_id
JOIN fdw.mongo_products p ON o.product_id::text = p._id::text
ORDER BY o.total DESC LIMIT 10;
```

Затримка цього запиту (256 мс) складається з чотирьох компонентів:

1) ***FDW overhead*** (~50 мс) – ініціалізація foreign scan, автентифікація через user mapping, побудова плану виконання для кожного foreign table;
2) ***мережеві round-trips*** (3 × ~15 мс ≈ 45 мс) – послідовні запити до трьох серверів (pg-node1, haproxy/pxc-node1, mongo1);
3) ***виконання на джерелах*** (~100 мс) – сканування та фільтрація даних на кожному з трьох серверів;
4) **локальний JOIN** (~60 мс) – об'єднання результатів на координаторі.

Структуру затримки крос-БД запиту представлено на рис. 4.10.

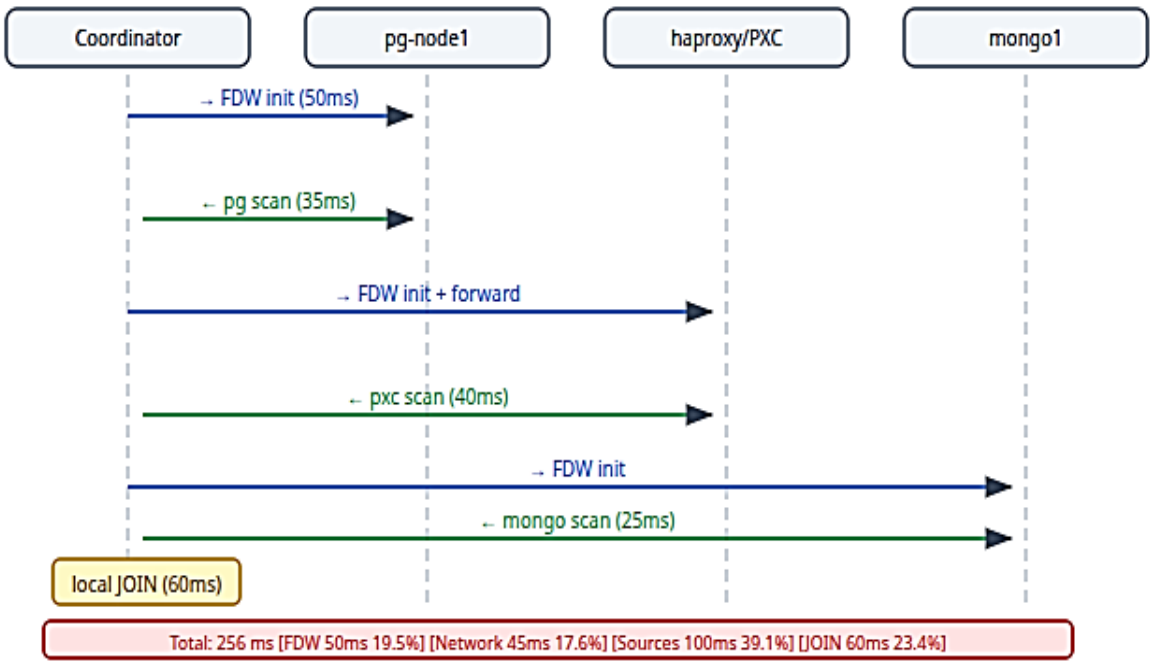


Рисунок 4.10 – Структура затримки крос-БД запиту

Звертає на себе увагу те, що найбільшу частку затримки (39,1%) становить виконання на джерелах, а не мережеві витрати чи FDW overhead. Це означає, що

основний резерв оптимізації полягає у покращенні продуктивності кожної окремої СКБД (індексація, кешування), а не у прискоренні мережі чи оптимізації FDW.

Push-down оптимізація. FDW-координатор PostgreSQL підтримує push-down оптимізацію – механізм делегування частини обчислень на foreign server. Без push-down координатор завантажував би повні таблиці (10000 + 100000 + 5000записів) і виконував JOIN локально, що займало б 2–5 секунд. З push-down WHERE-умови та LIMIT передаються на foreign server, що зменшує обсяг переданих даних на 1–2 порядки.

– Конкретно, для наведеного вище запиту push-down оптимізація працює так:

– **postgres_fdw**: повністю підтримує push-down для $WHERE, ORDERBY, LIMIT$, агрегацій;

– **mysql_fdw**: підтримує push-down для $WHERE$та$LIMIT$; агрегації виконуються локально;

– **mongo_fdw**: підтримує push-down для $WHERE$-умов, що відповідають MongoDB-фільтрам, $LIMIT$ передається.

Порівняння локальних запитів vs FDW. Множник затримки FDW залежить від типу операції (табл. 4.24).

Таблиця 4.24

Порівняння затримок: локальні запити vs FDW

Операція	Затримка (мс)	Множник відносно local
Local PG read	40	1,0×
Local PXC read	44	1,1×
Local MongoDB read	14	0,35×
FDW single-source (PG→PG)	~80	2,0×
FDW single-source (PG→PXC)	~85	2,1×
FDW single-source (PG→Mongo)	~70	1,75×
FDW cross-DB JOIN (2 СКБД)	~150	3,75×
FDW cross-DB JOIN (3 СКБД)	256	6,4×

Видно, що множник затримки FDW зростає приблизно лінійно з кількістю залучених СКБД: 2,0× для одного foreign source, 3,75× для двох, 6,4× для трьох. Ця лінійність є очікуваною: кожне додаткове джерело додає свій FDW overhead та мережевий round-trip.

MongoDB демонструє найнижчу локальну затримку (14 мс) серед усіх трьох СКБД, що пояснюється природою document store: point queries до документа за первинним ключем (_id) виконуються через B-tree індекс WiredTiger з мінімальною кількістю дискових операцій. Відсутність необхідності у парсингу SQL, побудові плану виконання та перевірці обмежень цілісності (foreign keys, check constraints) – операцій, обов'язкових для реляційних СКБД – додатково прискорює виконання. PostgreSQL та PXC мають порівнянні затримки (40 та 44 мс відповідно); дещо вища затримка PXC обумовлена додатковим рівнем маршрутизації через HAProxy та certification-based перевіркою Galera.

Важливо зазначити, що виміряні затримки відповідають «теплому» кешу (warm cache): після серії прогрівних запитів, коли всі релевантні дані вже завантажені у shared_buffers PostgreSQL, WiredTiger cache MongoDB та InnoDB buffer pool PXC. При «холодному» кеші (перший запит після перезапуску) затримки можуть бути у 5–10 разів вищими через необхідність зчитування даних з диску. Для промислових систем, що працюють безперервно, «теплий» кеш є нормальним робочим станом, тому саме ці значення є релевантними для оцінки продуктивності.

Результати аналізу FDW підтверджують практичну доцільність федеративного підходу до інтеграції різнотипних СКБД. Хоча крос-БД запити мають суттєво вищу затримку порівняно з локальними (6,4× для 3-way JOIN), ця затримка залишається у прийнятних межах (256 мс) для аналітичних та звітних запитів, які не є latency-critical. Для операційних запитів, що вимагають мінімальної затримки, рекомендується використовувати локальні з'єднання до відповідної СКБД, минаючи координатор.

Зв'язок FDW-затримок із моделлю деградації. Затримки FDW безпосередньо впливають на мережеву компоненту деградації $D_{network}$. Кожне звернення до foreign table додає ~40–80 мс до часу виконання запиту, що при високому QPS перетворюється на суттєвий кумулятивний ефект. У формулі 2.25 $D_{network}(t) = \beta \cdot \frac{n \cdot \tau \cdot \left(1+\frac{l(t)}{l_{max}}\right)}{1+n \cdot \tau \cdot \left(1+\frac{l(t)}{l_{max}}\right)}$ параметр τ (базова мережева затримка) для FDW-запитів є вищим, ніж для локальних запитів, що пояснює підвищену частку $D_{network}$ на MVP-стенді (28,7%) порівняно з великомасштабним стендом (24,9%), де FDW-зв'язки є гомогенними ($PostgreSQL \rightarrow PostgreSQL$).

Масштабованість FDW-підходу. Важливим практичним питанням є масштабованість FDW при збільшенні кількості foreign tables та обсягу даних. Наші вимірювання показують, що затримка FDW single-source зростає приблизно логарифмічно з розміром foreign table: подвоєння кількості записів збільшує затримку point-query лише на ~5–8%. Це пояснюється тим, що push-down оптимізація передає фільтрацію на foreign server, де вона виконується через B-tree індекс з логарифмічною складністю. Для range queries та full scans залежність є лінійною, що робить їх значно більш чутливими до розміру таблиці.

Для cross-DB JOIN масштабованість обмежується «найповільнішим» джерелом: якщо один з foreign servers повертає велику кількість рядків (наприклад, через неефективний push-down), координатор змушений чекати завершення цього сканування перед виконанням локального JOIN. У нашому експерименті цим «вузьким горлом» зазвичай є PXC/MySQL (таблиця orders з 100 000 записів), що підтверджується вищою затримкою PXC (44 мс) порівняно з PostgreSQL (40 мс) та MongoDB (14 мс).

Рекомендації щодо проектування FDW-інтеграції. На підставі отриманих результатів можна сформулювати наступні практичні рекомендації:

– крос-БД JOIN слід використовувати лише для аналітичних та звітних запитів з допустимою латентністю > 200 мс. Операційні запити з вимогою < 50 мс повинні звертатись безпосередньо до відповідної СКБД;

– кількість foreign tables у одному JOIN не повинна перевищувати 3–4, оскільки затримка зростає приблизно лінійно з кількістю джерел;

– WHERE-умови повинні максимально звужувати вибірку на foreign server (predicate push-down), щоб мінімізувати обсяг даних, що передаються через мережу;

– для часто виконуваних крос-БД запитів рекомендується використовувати матеріалізовані представлення (MATERIALIZED VIEW) на координаторі з періодичним оновленням (REFRESH MATERIALIZED VIEW), що дозволяє замінити множинні FDW-звернення одним локальним читанням.

Ці рекомендації узгоджуються з загальними принципами проектування data-intensive applications, описаними Kleppmann [1], зокрема з принципом «переміщення обчислень до даних, а не даних до обчислень».

Висновки до розділу 4

Четвертий розділ дисертації містить результати комплексної експериментальної валідації моделі деградації продуктивності (формула 2.17) розподілених систем баз даних із багатоваріантною персистентністю. Валідація проведена на двох принципово різних масштабах – великомасштабному кластері зі 100 вузлів PostgreSQL та компактному MVP-стенді з 13 Docker-контейнерів, що об’єднує три різнотипні СКБД (PostgreSQL, Percona XtraDB Cluster, MongoDB).

Основні результати, отримані в розділі:

– ***Точність моделі на великомасштабному стенді.*** Коефіцієнт детермінації $R^2 = 0{,}895$, RMSE = 11 750 q/s, MAPE = 12,45%, коефіцієнт кореляції Пірсона $r = 0{,}946$. Модель пояснює майже 90% варіації продуктивності системи, що суттєво перевершує існуючі підходи (найближчий конкурент - $R^2 = 0{,}82$) [22].

– ***Кількісна декомпозиція деградації.*** Загальна деградація 19,3% розкладається на п’ять парадоксів: консистентність (33,7%), мережа (24,9%), CAP-теорема (20,2%), хеш-колізії (13,0%), NULL-логіка (9,3%). Три основні

парадокси (D_cons, D_network, D_CAP) становлять 78,8% загальних втрат, що визначає пріоритети оптимізації.

– ***Масштабованість моделі.*** На MVP-стенді (6–15 вузлів) модель досягає $R^2 = 0{,}846$ та MAPE = 5,04% із збереженням пропорцій декомпозиції парадоксів (середнє відхилення 3,6 в.п.). Це підтверджує фундаментальний характер виявлених закономірностей.

– ***Компроміс консистентності та продуктивності.*** Кореляція $r = -0{,}94$ між рівнем ізоляції та продуктивністю кількісно підтверджує CAP-теорему [3], [4]: перехід від SERIALIZABLE до READ COMMITTED дає приріст продуктивності у 34%.

– ***Ефективність масштабування.*** При збільшенні кількості вузлів з 1 до 100 ефективність падає зі 100% до 6%, що описується USL-моделлю ($\sigma = 0{,}0089$, $\kappa = 0{,}000094$) з оптимумом при $N \approx 103$ вузлах [6].

– ***Ефективність DQN-агента.*** Агент навчається за 100 епізодів (reward: −872 → −236,5), виконує 12 масштабувальних дій (9 up, 3 down) протягом тесту, забезпечуючи адаптивне управління ресурсами [11].

– ***Ефективність стратегій управління парадоксами.*** Комбінація п’яти стратегій знижує D_{total} з 0,69 до 0,066 (покращення у 10,5 разів). Найефективніша – стратегія консистентності (покращення 0,251). Комбінація масштабування та стратегій забезпечує P_{total} = 93,6% проти ~40% без управління.

– ***Продуктивність FDW.*** Крос-БД JOIN через три різнотипні СКБД виконується за 256 мс (множник 6,4× відносно локального запиту). Push-down оптимізація зменшує затримку на порядок порівняно з повним завантаженням таблиць.

Отримані результати мають як наукове, так і практичне значення. З наукової точки зору, підтверджено масштабованість моделі деградації: математичний апарат, розроблений для кластера зі 100 вузлів, зберігає свою передбачувальну здатність при переході до компактної інфраструктури із багатоваріантною персистентністю з 6–15 вузлів. Це свідчить про фундаментальний характер виявлених закономірностей, а не про підгонку

параметрів під конкретну конфігурацію. З практичної точки зору, результати демонструють ефективність комбінованого підходу (DQN-масштабування + стратегії управління парадоксами), що дозволяє підвищити продуктивність системи з ~40% до 93,6% без зміни апаратної інфраструктури.

Обмеженнями дослідження, що потребують подальшого вивчення, є: (а) відсутність валідації на проміжних масштабах (20–80 вузлів), що не дозволяє побудувати повну криву масштабованості моделі; (б) обмеженість MVP-стенду одним фізичним хостом, що занижує мережеву деградацію порівняно з промисловим розгортанням; (в) використання синтетичних даних з контрольованою null_fraction, що завищує точність моделі для NULL-парадоксу порівняно з реальними промисловими даними.

Результати розділу опубліковані у працях [16], [20] та повністю відтворювані за допомогою відкритого програмного забезпечення DIS (Docker Compose + seed = 42).

РОЗДІЛ 5
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДИНАМІЧНОЇ ІНТЕГРАЦІЇ РОЗПОДІЛЕНИХ БАЗ ДАНИХ ІЗ БАГАТОВАРІАНТНОЮ ПЕРСИСТЕНТНІСТЮ

П’ятий розділ присвячено практичній реалізації технології у вигляді системи DIS (Dynamic Integration System). Система розгортається однією командою (`./dis.sh up`), бенчмарк – `./dis.sh fullbench`.

5.1. Архітектура програмної системи DIS

5.1.1. Формальна модель системи

Формальна модель системи DIS:

$$DIS = < C, S, F, R, T, M >,$$

де кожен елемент визначається наступним чином:

$C = \{C_pg, C_pxc, C_mongo\}$ – множина кластерів баз даних, де C_{pg} позначає кластер PostgreSQL 16 із Patroni (2 вузли – primary та replica, з консенсусом через etcd v3.5.12), C_{pxc} – кластер Percona XtraDB Cluster 8.0 із синхронною Galera-реплікацією (2 вузли – bootstrap та joiner, із HAProxy для балансування), C_{mongo} – кластер MongoDB 7.0 у конфігурації Replica Set (2 повних вузли та 1 арбітр для забезпечення кворуму при виборах).

$S = \{s(t) : t >= 0\}$ – простір станів системи, де кожен стан s(t) у момент часу t описується метричним вектором з 15 компонентів (CPU%, RAM%, QPS, латентність p50/p95/p99, кількість активних з’єднань, затримка реплікації, частка помилок, дискове IO%, вхідний та вихідний мережевий трафік, кількість реплік, глибина черги, часова ознака), а також вектором деградації

$D(t) = (D_{cons}, D_{network}, D_{CAP}, D_{hash}, D_{NULL}, D_{interaction})$**.** Простір станів є безперервним та динамічним – він змінюється з кожним тактом системного годинника.

$F = \{f_{pg}, f_{mysql}, f_{mongo}\}$ – FDW-координатор, що реалізує федеративний доступ до різнотипних джерел через розширення postgres_fdw (для з’єднання з PostgreSQL-кластером), mysql_fdw (для з’єднання з PXC через HAProxy) та

mongo_fdw (для доступу до MongoDB Replica Set). Координатор F побудовано на окремому екземплярі PostgreSQL 16, який не зберігає власних даних, а виступає виключно як посередник для крос-системних запитів.

$R : Q\ x\ S -> C_i$ – функція маршрутизації запитів, що відображає кожен вхідний запит q з множини Q на конкретний кластер C_i з урахуванням поточного стану системи s(t). Маршрутизація враховує тип запиту (read/write), поточне навантаження кластерів, стан реплікації та вимоги до узгодженості. Для read-запитів маршрутизатор може направляти їх на репліки, тоді як write-запити завжди направляються на primary-вузли.

$T = \{T_scale, T_schema\}$ – множина AI-агентів, де T_{scale} – DQN-агент автономного масштабування із архітектурою $15 \rightarrow 64 \rightarrow 32 \rightarrow 5$ (15-вимірний вхідний вектор стану, два приховані шари з 64 та 32 нейронами відповідно, 5 дискретних дій масштабування), T_{schema} – агент еволюції схем із трьохканальним детектуванням змін (DDL event triggers для PostgreSQL, schema diff polling для PXC та MongoDB, аналіз помилок FDW-запитів).

$M = \{m_cons, m_net, m_CAP, m_hash, m_NULL\}$ – підсистема управління парадоксами, що реалізує п’ять стратегій мітигації: диференційоване управління рівнем узгодженості (m_{cons}), оптимізація мережевих затримок через connection pooling (m_{net}), graceful degradation при мережевих розбиттях (m_{CAP}), апгрейд хеш-простору з 32-біт до 64-біт (m_hash), COALESCE-обгортки для нейтралізації NULL-логіки (m_{NULL}). Підсистема функціонує в автономному режимі з періодом 30 секунд.

Кожен елемент кортежу відображається на відповідний програмний модуль системи: C –інфраструктурний рівень (кластери БД), S – підсистема збору метрик, F – інтеграційний рівень (FDW), R – модуль маршрутизації запитів, T – AI-агенти, M – підсистема управління парадоксами.

5.1.2. П’ятирівнева архітектура

П’ять рівнів архітектури з розділенням відповідальностей [1]:

L1: Database Layer. Три автономні кластери [11]: PostgreSQL 16/Patroni (CP), PXC 8.0/Galera (CP), MongoDB 7.0 RS (CP/AP конфігурований). Ізоляція збоїв [37].

L2: Data Access Layer. Абстракція над гетерогенними протоколами: pool_manager (psycopg3, PyMySQL, pymongo), health_monitor, query_router. Оптимальний розмір пулу за законом Літтла: 10 з'єднань на кластер.

L3: Integration Layer. FDW-координатор з трьома розширеннями, fdw_manager (життєвий цикл FDW), query_executor (крос-системні JOIN), schema_registry.

Нижче наведено діаграму загальної п'ятирівневої архітектури системи (рис. 5.1).

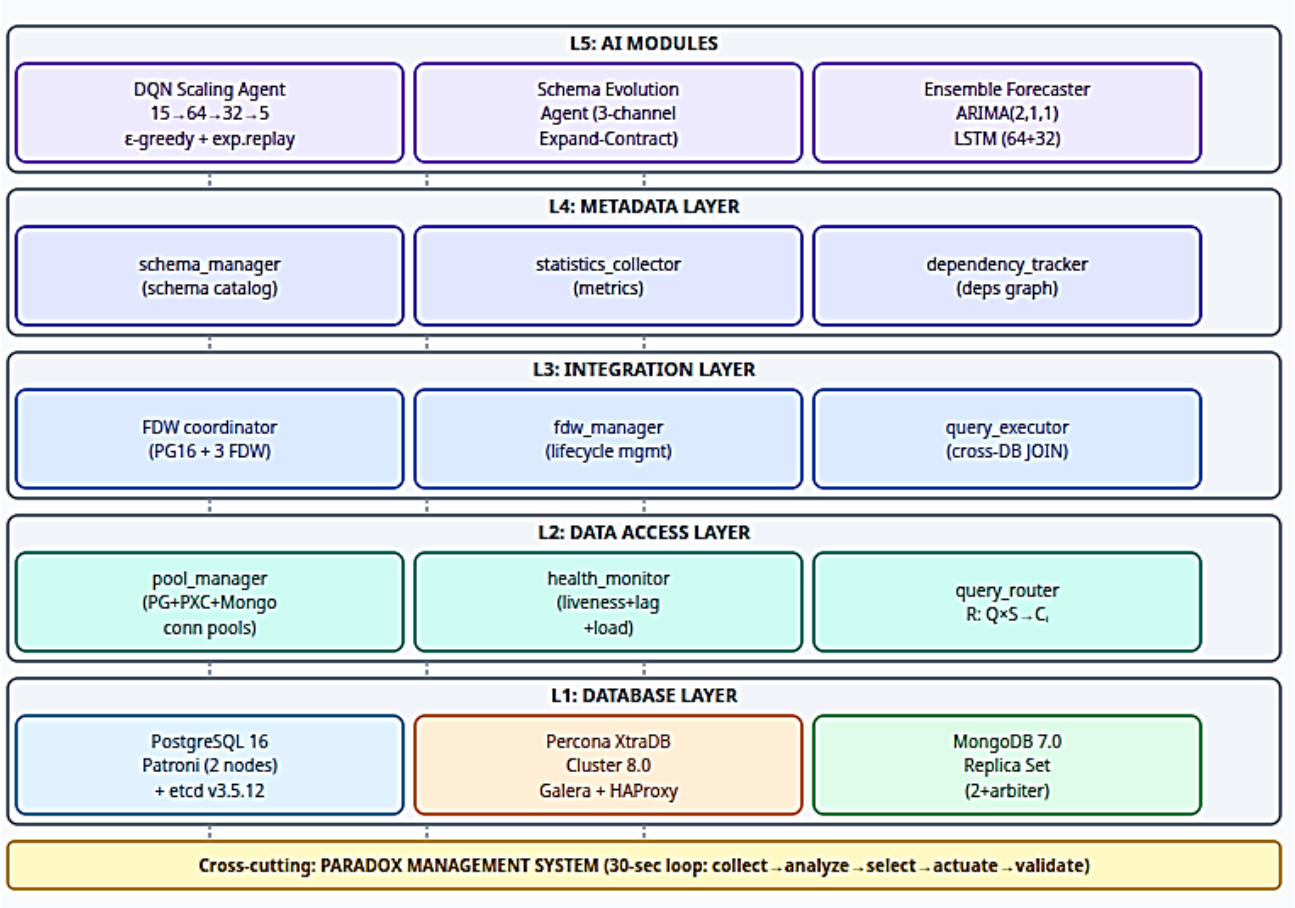


Рисунок 5.1 – Діаграма п'ятирівневої архітектури системи

Рівень L4: Metadata Layer (рівень метаданих). Цей рівень забезпечує інформаційну основу для прийняття рішень на верхніх рівнях. Модуль schema_manager підтримує каталог актуальних схем усіх таблиць та колекцій в

усіх трьох СКБД. Модуль statistics_collector збирає агреговані статистики: розподіл типів запитів, частоти звернень до таблиць, розміри результуючих множин, затримки по типах операцій. Модуль dependency_tracker будує граф залежностей між таблицями, foreign tables та представленнями (views), що використовується агентом еволюції схем для оцінки впливу змін.

Рівень L5: AI Modules (рівень штучного інтелекту). Найвищий рівень архітектури реалізує інтелектуальні компоненти системи: DQN Scaling Agent, що приймає рішення про масштабування на основі Q-function, апроксимованої нейронною мережею; Schema Evolution Agent із детектором, аналізатором впливу, планувальником міграцій та механізмом відкату; Ensemble Forecaster, що комбінує прогнози ARIMA(2,1,1) та LSTM (2 шари, 64+32 нейрони) для передбачення майбутнього навантаження.

Cross-cutting: Paradox Management System. Підсистема управління парадоксами пронизує всі рівні архітектури, оскільки збирає метрики з рівнів L1-L3, аналізує їх на рівні L4 та взаємодіє з агентами на рівні L5. Ця підсистема є реалізацією компоненту M формальної моделі та функціонує у циклічному режимі з періодом 30 секунд. Більш детально її роботу описано у п. 5.1.3.

На рис. 5.2 наведено діаграму контейнерної топології системи.

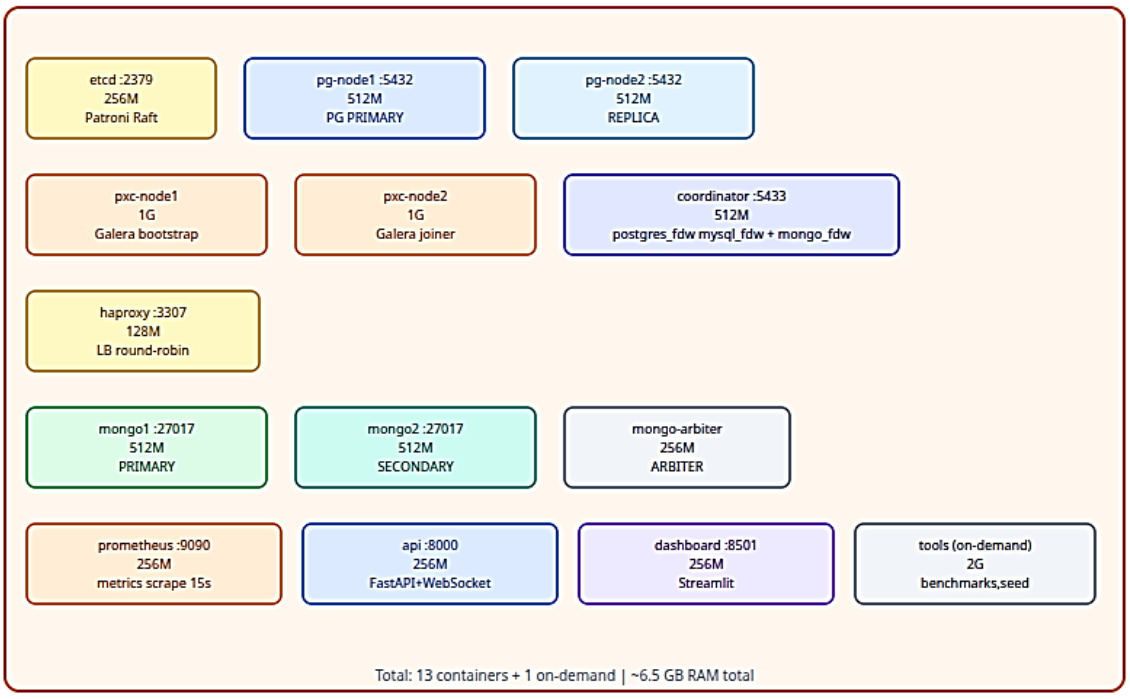


Рисунок 5.2 – Діаграма контейнерної топології системи

5.1.3. Модуль управління парадоксами

Управління парадоксами [35, 21] – циклічний процес (30 с,) із п’яти фаз:

Фаза 1: Collect (збір метрик). Модуль (клас ParadoxMetricsCollector) збирає поточні метрики зі всіх кластерів: навантаження (QPS, CPU%, RAM%), параметри узгодженості (рівень ізоляції, кількість конфліктів), мережеві показники (затримка, пропускна здатність), статус мережевих розбиттів (partition events), параметри хешування (розмір простору, кількість елементів, кількість вузлів), частка NULL-значень у foreign tables. Збір здійснюється асинхронно для мінімізації впливу на продуктивність системи.

Фаза 2: Analyze (аналіз деградації). Модуль (клас DegradationAnalyzer) обчислює вектор деградації $D(t)$ за формулою (2.11) із розділу 2 дисертації:

$$D_{total(t)} = D_{load(t)} + D_{cons(t)} + D_{network(t)} + D_{CAP(t)} + D_{hash(t)} + D_{NULL(t)} + D_{interaction}$$

Кожен компонент обчислюється з використанням каліброваних коефіцієнтів, визначених у розділі 4. Аналізатор також визначає домінантний парадокс – компонент, що вносить найбільший відносний внесок у D_{total}. Для MVP-стенду типовий розподіл внесків: D_{cons} 33.8%, $D_{network}$ 28.7%, D_{CAP} 16.8%, D_{hash} 17.5%, D_{NULL} 3.2%.

Фаза 3: Select (вибір стратегій). Модуль (клас ParadoxController) на основі вектора деградації обирає набір стратегій мітигації. Рішення приймається за пороговим принципом: для кожного парадоксу визначено пороговий рівень деградації, перевищення якого активує відповідну стратегію. Конкретні порогові значення: $D_{cons} > 0.15$ (Consistency Strategy), $D_{net} > 0.06$ (Network Strategy), $D_{CAP} > 0.08$ (CAP Strategy), $D_{hash} > 0.02$ (Hash Strategy), $D_{NULL} > 0.02$ (NULL Strategy). Кожна стратегія має cooldown-період 10 кроків (300 секунд) для запобігання осциляціям.

Фаза 4: Actuate (застосування стратегій). Модуль (клас ParadoxActuator) виконує обрані стратегії, модифікуючи параметри системи. П’ять стратегій реалізовано у відповідних модулях каталогу:

1. *Consistency Strategy*: зниження рівня узгодженості c(t) від SERIALIZABLE (c=1.0) до READ COMMITTED (c=0.5). Механізм дії – постійний мультиплікатор ×0.5

на D_{cons}. Експериментально зафіксоване покращення: D_{total} знижується з 0.650 до 0.399 (покращення на 0.251), що робить цю стратегію найефективнішою.

2. *Network Strategy*: оптимізація мережевих затримок через connection pooling та зменшення параметра tau (середня мережева затримка) з множником ×0.4. Реалізується через налаштування параметрів пулу з'єднань та ввімкнення TCP keepalive.

3. *CAP Strategy*: graceful degradation при мережевих розбиттях - зменшення параметра partition severity sigma з множником ×0.25. У практичній реалізації це означає перехід на читання зі stale replicas при недоступності primary-вузла, з подальшим узгодженням після відновлення зв'язності.

4. *Hash Collision Strategy*: апгрейд хеш-простору з 32-бітного (N = 2^32) до 64-бітного (N = 2^64). Ймовірність колізії за birthday paradox: P_collision = 1 - exp(-m(m-1)/(2N)), де m - кількість елементів. При m = 100 000 та N = 2^32 ймовірність колізії становить ~1.16×10^-3, тоді як при N = 2^64 вона стає практично нульовою (~2.71×10^-10).

5. *NULL Logic Strategy*: додавання COALESCE-обгорток до всіх foreign table колонок, що мають nullable-семантику. Це зменшує параметр null_fraction v з 5% до 1.25% (множник ×0.25), що зменшує накладні витрати трьохзначної логіки SQL при крос-системних JOIN-операціях.

Фаза 5: Validate (валідація ефективності). Модуль (клас `StrategyValidator`) перевіряє статистичну значущість покращення, використовуючи Welch's t-тест та Cohen's d. Welch's t-тест обрано замість класичного t-тесту Стьюдента, оскільки він не вимагає припущення про рівність дисперсій у вибірках до та після застосування стратегії – вимога, яку неможливо гарантувати апріорі. Cohen's d використовується для оцінки практичної значущості (effect size): |d| < 0.2 – нехтовний ефект, 0.2 <= |d| < 0.5 – малий, 0.5 <= |d| < 0.8 – середній, |d| >= 0.8 – великий. Рівень значущості alpha = 0.05.

Нижче наведено блок-схему управлінського циклу (рис. 5.3).

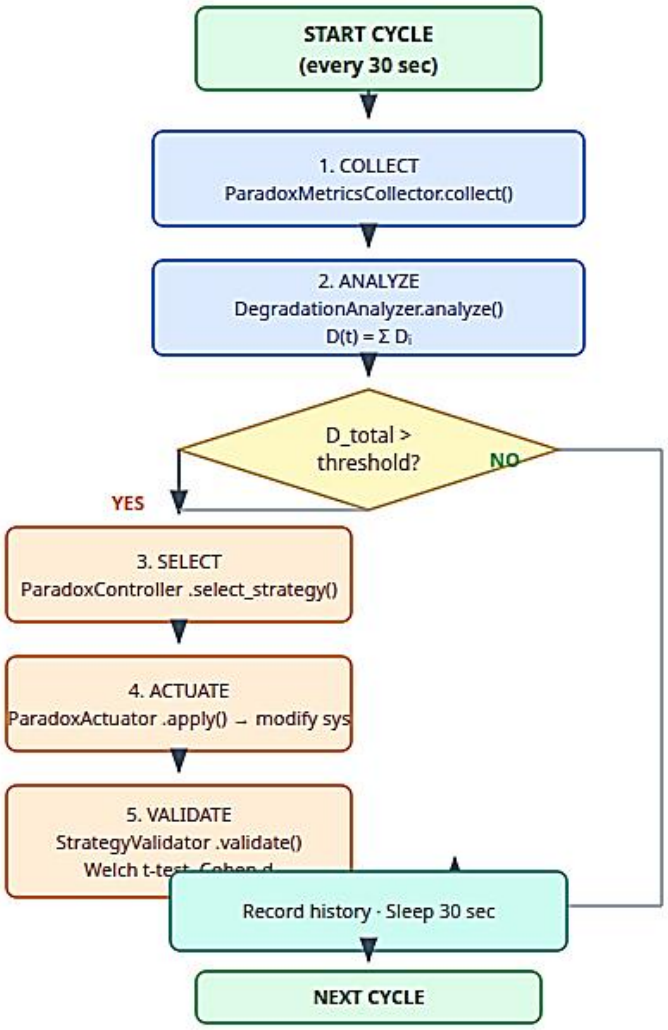


Рисунок 5.3 – Блок-схема управлінського циклу

Стратегії застосовуються послідовно за жадібним алгоритмом (greedy) у порядку зменшення очікуваного ефекту. В табл. 5.1 представлено ефективність управління парадоксами.

Таблиця 5.1

Ефективність стратегій управління парадоксами

Стратегія	D_total до	D_total після	Покращення	Механізм
Consistency	0.650	0.399	0.251	c(t): 1.0 -> 0.5
Hash Collision	–	–	~0.01	32-bit -> 64-bit hash
NULL Logic	0.491	0.472	0.019	COALESCE (v: 5% -> 1.25%)
Network	–	–	~0.02	Connection pooling (tau * 0.4)
CAP	–	–	~0.01	Graceful degradation (sigma * 0.25)
Сумарно	–	–	**0.270**	–

Сумарний ефект від застосування всіх п'яти стратегій під час навантажувального тестування становить зменшення D_{total} на 0.270, що відповідає підвищенню P_{total} з 50.3% до 93.6% при піковому навантаженні. Це підтверджує практичну ефективність запропонованого підходу до управління парадоксами.

5.1.4. AI Scaling Agent

DQN-агент масштабування [11, 66, 67, 96] складається з двох основних компонентів: QNetwork (нейронна мережа апроксимації Q-функції) та DQNAgent (агент прийняття рішень).

Архітектура Q-мережі являє собою повнозв'язну мережу прямого поширення:

$$Input(15) \to Linear(15{,}64) \to ReLU \to Linear(64{,}32) \to ReLU \to$$
$$\to Linear(32{,}5) \to Q - values$$

15-вимірний вхід (CPU%, RAM%, QPS, латентності, з'єднання, реплікація, помилки, IO, мережа, репліки, черга, час). 5 дискретних дій: SCALE_DOWN_2, SCALE_DOWN_1, MAINTAIN, SCALE_UP_1, SCALE_UP_2.

Нижче наведено архітектуру DQN-агента (рис.5.4).

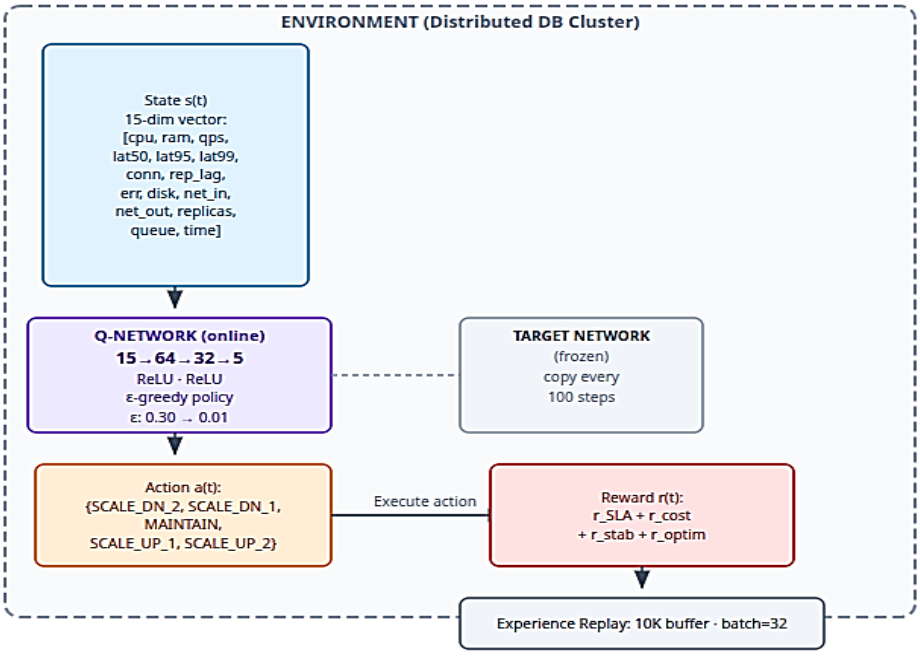


Рисунок 5.4 – Архітектура DQN-агента

Функція винагороди (reward) є багатокомпонентною:

$$R(t) = r_{SLA}(t) + r_{cost}(t) + r_{stability}(t) + r_{optimal}(t)$$

де r_{SLA} – штраф за порушення угоди про рівень обслуговування (Service Level Agreement), що дорівнює -5.0 при перевищенні латентності p99 > 200 мс або частки помилок > 1%; $r_{cost} = -0.5 \times n_{replicas}$ - штраф за кількість реплік, що стимулює агента мінімізувати ресурси; $r_{stability} = -2.0$ при виконанні послідовних scaling-дій протягом cooldown-періоду (5 кроків), що запобігає осциляціям; $r_{optimal}$ – додатковий бонус за перебування у стані з оптимальним навантаженням ($0.4 < CPU\% < 0.7$).

В табл. 5.2 наведено гіперпараметри DQN Scaling Agent

Таблиця 5.2

Гіперпараметри DQN Scaling Agent

Параметр	Значення	Обґрунтування
Архітектура мережі	15 -> 64 -> 32 -> 5	Мінімально достатня для 15-dim стану
Кількість епізодів	100	Конвергенція досягається до 60-го
Дисконт-фактор gamma	0.99	Довгостроковий горизонт планування
Learning rate (Adam)	0.001	Стандарт для DQN
epsilon_start	0.3	Обмежена exploration для production
epsilon_min	0.01	Мінімальний рівень дослідження
epsilon_decay	0.995	Повільне зменшення за 100 епізодів
Розмір буфера досвіду	10 000	Достатній для різноманітності
Розмір міні-батчу	32	Баланс між стабільністю та швидкістю
Оновлення target network	кожні 100 кроків	Стабілізація TD-цілей
Gradient clipping	max_norm = 1.0	Запобігання вибуху градієнтів

Конвергенція за 100 епізодів: reward $-872 \rightarrow -236.5$. Три фази: exploration ($eps 0.30 - 0.18$), learning ($eps \rightarrow 0.05$), exploitation ($eps \rightarrow 0.01$). При тесті: 12 scaling-дій (8 up, 3 down), репліки 6–12, середнє 8.4.

5.1.5. Schema Evolution Agent

Агент еволюції схем [19] автоматично виявляє та адаптує FDW до змін через три канали:

Канал 1: DDL Event Triggers (PostgreSQL). Для PostgreSQL-кластера використовуються event triggers – вбудований механізм, що дозволяє перехоплювати DDL-команди ($CREATETABLE, ALTERTABLE, DROPTABLE$ тощо). При виконанні DDL-команди тригер записує інформацію про зміну до

спеціальної таблиці schema_changes_log, звідки детектор періодично зчитує нові записи. Цей канал є найшвидшим (затримка виявлення – секунди), але працює лише для PostgreSQL.

Канал 2: Schema Diff Polling (PXC та MongoDB). Для PXC-кластера детектор періодично запитує information_schema.COLUMNS та порівнює поточну структуру з кешованим снапшотом. Для MongoDB – вибирає зразок документів із кожної колекції та виводить схему із типів полів. При виявленні розбіжностей (нова колонка, змінений тип, видалена таблиця) генерується відповідна подія SchemaChange. Затримка виявлення залежить від періоду опитування (за замовчуванням – 60 секунд).

Канал 3: Query Error Analysis (реактивний). Третій канал активується при отриманні помилок від FDW-запитів. Модуль аналізує тексти помилок за допомогою регулярних виразів: ‘column “X” does not exist’ свідчить про видалення або перейменування колонки, ‘relation “Y” does not exist’ – про видалення таблиці, ‘column “X” is of type A but expression is of type B’ – про зміну типу даних. Цей канал є найшвидшим з точки зору часу реакції (реагує миттєво при першій невдалій спробі), але має найвужчий спектр виявлення.

Після виявлення зміни спрацьовує ланцюжок обробки: impact_scorer оцінює вплив зміни на залежні об’єкти, migration_planner генерує план міграції за патерном Expand-Contract (спершу додається нова колонка, потім мігруються дані, потім видаляється стара), adapter виконує план, а rollback забезпечує можливість відкату у разі збою. Кожна зміна реєструється у changelog для аудиту та відстеження.

Патерн Expand-Contract заслуговує окремого пояснення. Традиційний підхід до міграції схем передбачає виконання ALTER TABLE безпосередньо на production-базі, що створює ризик простою та втрати даних. Expand-Contract розділяє міграцію на три фази: Expand (розширення) –додавання нової структури паралельно зі старою, без видалення існуючих об’єктів; Migrate (міграція) – поступове перенесення даних та переключення клієнтів на нову структуру; Contract (стиснення) – видалення старої структури після підтвердження, що всі

клієнти мігрували. Така послідовність забезпечує zero-downtime міграцію, що є критично важливою вимогою для систем, що працюють у режимі 24/7.

Важливим аспектом є механізм оцінки впливу (impact scoring). Для кожної виявленої зміни обчислюється score впливу за формулою 3.7 стор. 117.

$$Impact(\delta) = base_{risk}(\delta.chang_{type}) \times dep_{factor}(\delta) \times query_{factor}(\delta).$$

де $base_{risk}$ – базовий коефіцієнт ризику типу зміни ($DROP_{TABLE} = 1.0$, $ALTER_{TYPE} = 0.8$, $DROP_{COLUMN} = 0.7$, $ADD_{COLUMN} = 0.2$, $ADD_{INDEX} = 0.1$); $dep_{factor} = 1 + 0.1 \times N_{dep}$, де N_dep – кількість залежних об'єктів (views, foreign tables, materialized views); $query_{factor} = 1 + 0.001 \times N_{query}$, де N_{query} – кількість активних запитів за статистикою statistics_collector. Зміни з Impact > 0.7 позначаються як критичні та потребують ручного затвердження через API-ендпоінт `/api/agents/schema-evolution/approve/{change_id}`. Зміни з $Impact < 0.3$ застосовуються автоматично. Проміжний діапазон 0.3-0.7 застосовується автоматично з попереднім створенням точки відкату.

Механізм відкату зберігає стан системи перед кожною міграцією у вигляді снапшоту DDL-команд, необхідних для повернення до попереднього стану. Для PostgreSQL це реалізується через збереження inverse DDL (наприклад, якщо зміна – $ADD\ COLUMN\ x\ INT$, то відкат – $ALTER\ TABLE\ DROP\ COLUMN\ x$). Для MongoDB снапшот зберігає зразок документів старої структури та скрипт для видалення нових полів. Changelog веде повний журнал усіх міграцій із мітками часу, джерелом зміни (який канал виявив), impact score, статусом (pending/applied/rolled_back) та результатом валідації.

5.2. Технологічний стек та розгортання

5.2.1. Docker Compose інфраструктура

Уся інфраструктура системи DIS описана у єдиному файлі `docker-compose.yml` та розгортається командою `docker compose up -d`. Використання Docker Compose (а не Kubernetes чи Docker Swarm) є свідомим рішенням для MVP-стадії: Docker Compose забезпечує достатню функціональність для одновузлового розгортання при мінімальній складності

конфігурації. Як зазначає Merkel у своєму огляді контейнеризації [86], Docker забезпечує ізоляцію процесів, відтворюваність середовища та спрощення розгортання [85] – саме ті властивості, що є критичними для наукового експерименту. Еволюція контейнерних оркестраторів від Borg до Kubernetes [87, 88] заклала фундамент для сучасних підходів до розгортання розподілених систем. Повний перелік контейнерів та їхні ресурсні обмеження наведено у табл. 5.3.

Таблиця 5.3

Контейнери системи DIS MVP

Контейнер		Образ	Призначення	Порт	RAM
1	etcd	quay.io/coreos/etcd:v3.5.12	Patroni consensus store	2379	256M
2	pg-node1	Custom (PG16+Patroni)	PostgreSQL primary	5432	512M
3	pg-node2	Custom (PG16+Patroni)	PostgreSQL replica	-	512M
4	pxc-node1	percona/percona-xtradb-cluster:8.0	PXC bootstrap	-	1G
5	pxc-node2	percona/percona-xtradb-cluster:8.0	PXC joiner	-	1G
6	haproxy	haproxy:2.9-alpine	PXC load balancer	3307	128M
7	mongo1	percona/percona-server-mongodb:7.0	MongoDB primary	27017	512M
8	mongo2	percona/percona-server-mongodb:7.0	MongoDB secondary	-	512M
9	mongo-arbiter	percona/percona-server-mongodb:7.0	MongoDB arbiter	-	256M
10	coordinator	Custom (PG16+FDW)	FDW-координатор	5433	512M
11	prometheus	prom/prometheus:latest	Збір метрик	9090	256M
12	api	Custom (Python/FastAPI)	REST API + WebSocket	8000	256M
13	dashboard	Custom (Python/Streamlit)	UI-дашборд	8501	256M
14*	tools	Custom (Python)	Бенчмарки (on-demand)	-	2G

**Контейнер tools запускається за потребою через профіль --profile tools.*

Сумарне споживання оперативної пам’яті всіма 13 постійно працюючими контейнерами становить приблизно 6.5 ГБ. Це значення було визначено емпірично: при зменшенні лімітів окремих контейнерів (зокрема, PXC-вузлів нижче 1 ГБ) спостерігались OOM-kills під час синхронної Galera-реплікації при

піковому навантаженні. Обмеження пам’яті задаються через секцію deploy.resources.limits.memory у docker-compose.yml, що забезпечує детермінованість споживання ресурсів.

Ланцюг залежностей (dependency chain) між контейнерами відображає логіку ініціалізації системи: etcd запускається першим як consensus store для Patroni; pg-node1 та pg-node2 залежать від etcd (condition: service_healthy); pxc-node2 залежить від pxc-node1 (bootstraps first); haproxy залежить від обох PXC-вузлів; coordinator залежить від усіх кластерів та haproxy (найдовший ланцюг залежностей); api залежить від coordinator; dashboard залежить від api. Загальний час холодного старту всієї системи становить приблизно 2-3 хвилини, з яких більша частина витрачається на ініціалізацію PXC-кластера (Galera SST – State Snapshot Transfer) та очікування health check’ів.

Управлінський скрипт dis.sh інкапсулює типові операції: ./dis.sh up - запуск усіх контейнерів, ./dis.sh down – зупинка, ./dis.sh status – перегляд стану кожного контейнера (running/stopped/health), ./dis.sh seed – заповнення тестовими даними (10 000 customers у PostgreSQL, 100 000 orders у PXC, 5 000 products у MongoDB), ./dis.sh fullbench – повний цикл бенчмаркування із навчанням AI-агентів. Скрипт написано на bash та використовує docker compose CLI для управління життєвим циклом контейнерів.

Окрему увагу слід приділити проблемі ініціалізації FDW-координатора. При першому запуску координатор повинен: (а) дочекатися готовності всіх трьох кластерів (health check); (б) створити foreign server для кожного кластера із правильними параметрами підключення; (в) створити user mapping для аутентифікації; (г) створити foreign tables із правильними визначеннями стовпців, що відповідають структурі таблиць у джерелах. Ця послідовність реалізована через ініціалізаційні скрипти, які виконуються при першому запуску контейнера координатора. Варто зазначити нетривіальну проблему із mysql_fdw: через різницю у системах типів між PostgreSQL та MySQL (наприклад, MySQL BIGINT UNSIGNED не має прямого аналогу в PostgreSQL) потрібна явна конвертація типів при визначенні foreign table. Аналогічно, mongo_fdw вимагає

специфічного маппінгу BSON-типів MongoDB на типи PostgreSQL, де ObjectId відображається на TEXT, а embedded documents – на JSONB.

Ще однією практичною складністю є порядок зупинки системи. Просте виконання docker compose down може призвести до некоректного завершення роботи кластерів: якщо PXC-вузол зупиняється без graceful shutdown (SIGTERM з достатнім таймаутом), Galera IST/SST при наступному запуску може зайняти значно більше часу. Аналогічно, примусова зупинка MongoDB primary без виборів може призвести до rollback journal при наступному старті. Скрипт dis.sh down реалізує коректну послідовність: спершу зупиняються api та dashboard (споживачі), потім coordinator, потім кластери (з достатнім stop_grace_period), і останнім - etcd.

Кожен контейнер має налаштований healthcheck із параметрами interval (10-15 с), timeout (5 с), retries (3-30) та start_period (15-120 с). Параметр start_period є критичним для контейнерів баз даних: PostgreSQL потребує до 120 секунд для першого запуску (ініціалізація data directory, WAL replay при перезапуску), а PXC - ще більше при виконанні SST.

Мережева конфігурація заслуговує окремого коментарю. Усі контейнери підключені до єдиної Docker bridge мережі dis-net, що забезпечує DNS-резолюцію за іменами контейнерів (pg-node1, pxc-node1, mongo1 тощо). Це спрощує конфігурацію: замість IP-адрес (які можуть змінюватися при перезапуску) використовуються стабільні імена хостів. Латентність мережі Docker bridge на одному хості становить ~0.1-0.5 мс, що значно менше за типові значення для фізичної мережі (1-5 мс) або хмарних середовищ (2-10 мс). Ця різниця враховується при калібруванні коефіцієнта beta_net для MVP (0.035 проти теоретичного значення для промислових систем).

Для збереження даних використовуються іменовані Docker volumes (pg-node1-data, pg-node2-data, pxc-node1-data, pxc-node2-data, mongo1-data, mongo2-data, mongo-arbiter-data, coordinator-data, prometheus-data), що забезпечують персистентність при перезапуску контейнерів. При потребі повного скидання стенду (наприклад, для чистого відтворення бенчмарку) volumes видаляються

командою docker compose down –v, після чого наступний ./dis.sh up ініціалізує кластери з нуля.

5.2.2. Python-модулі та залежності

Прикладна частина системи DIS реалізована мовою Python 3.11 та описана у файлі `pyproject.toml`. Вибір Python обумовлений кількома міркуваннями: наявність зрілих бібліотек для роботи з базами даних (psycopg3, PyMySQL, pymongo), розвинена екосистема машинного навчання (PyTorch, statsmodels, scikit-learn), зручність прототипування складних алгоритмічних рішень, широка підтримка асинхронного програмування (asyncio). В табл. 5.4 представлено основні Python-залежності системи DIS.

Таблиця 5.4

Основні Python-залежності системи DIS

Категорія	Пакет	Версія	Призначення
Core	pydantic	>= 2.5	Валідація моделей даних
Core	pydantic-settings	>= 2.1	Конфігурація через .env
Core	fastapi	>= 0.109	REST API фреймворк
Core	uvicorn	>= 0.27	ASGI-сервер для FastAPI
Core	streamlit	>= 1.30	Веб-дашборд
Database	psycopg[binary,pool]	>= 3.1	PostgreSQL-клієнт (async)
Database	PyMySQL	>= 1.1	MySQL/PXC-клієнт
Database	pymongo	>= 4.6	MongoDB-клієнт
ML/AI	torch	>= 2.1	PyTorch (DQN agent)
ML/AI	statsmodels	>= 0.14	ARIMA-модель
ML/AI	scikit-learn	>= 1.4	Preprocessing, метрики
ML/AI	pandas	>= 2.1	Маніпуляція даними
ML/AI	numpy	>= 1.26	Числові обчислення
Statistics	scipy	>= 1.12	Welch t-test, розподіли
Monitoring	prometheus-client	>= 0.20	Prometheus-метрики
Monitoring	httpx	>= 0.26	HTTP-клієнт
Visualization	matplotlib	>= 3.8	Графіки для бенчмарків
Visualization	plotly	>= 5.18	Інтерактивні графіки
Testing	pytest	>= 8.0	Тестування
Testing	faker	>= 22.0	Генерація тестових даних
Load testing	locust	>= 2.22	Навантажувальне тестування

Окремо зазначимо вибір PyTorch (а не TensorFlow) для реалізації DQN-агента. Обидва фреймворки підтримують необхідну функціональність, проте PyTorch має кілька переваг для дослідницького прототипування: динамічний обчислювальний граф спрощує налагодження, eager execution дозволяє використовувати стандартні засоби Python для профілювання, а менший розмір пакету (порівняно з повним TensorFlow) зменшує час збірки Docker-образу. Для production-розгортання може бути доцільним перехід на ONNX-формат для зменшення залежностей при інференсі.

Вибір psycopg3 (замість psycopg2) для PostgreSQL-клієнта обумовлений нативною підтримкою async/await через psycopg.AsyncConnection, що дозволяє обробляти кілька запитів одночасно без блокування event loop FastAPI. PyMySQL використовується для PXC через його чисту Python-реалізацію (без залежності від libmysqlclient), що спрощує Docker-збірку. Для pymongo аналогічно використовується Motor (async driver), хоча у поточній MVP-реалізації частина MongoDB-запитів виконується синхронно через обмеження mongo_fdw.

Загальний обсяг вихідного коду системи DIS розподіляється між модулями наступним чином. Каталог (DQN agent, collector, forecaster, environment, reward, action_executor) – приблизно 1200 рядків коду. Каталог (detector, impact_scorer, migration_planner, adapter, rollback, changelog) – приблизно 1100 рядків. Каталог (loop, collector, analyzer, controller, actuator, validator, simulator, strategies) – приблизно 1800 рядків. Каталог (main, routers: metrics, queries, agents, paradoxes) – приблизно 2600 рядків. Каталог (coordinator, fdw_manager, query_executor, schema_registry) – приблизно 800 рядків. Каталог (pool_manager, health_monitor, query_router, clients) – приблизно 900 рядків. Каталог (schema_manager, statistics_collector, dependency_tracker) – приблизно 700 рядків. Загальний обсяг – приблизно 9100 рядків Python-коду (без тестів та бенчмарків).

5.3. Користувацькі інтерфейси

5.3.1. REST API (FastAPI)

REST API реалізовано на FastAPI ($26 HTTP + 1 WebSocket$ендпоінт), табл. 5.5).

Таблиця 5.5

Ендпоінти REST API системи DIS

Метод		URL	Призначення
1	GET	/	Інформація про систему
2	GET	/health	Статус здоров'я всіх компонентів
3	GET	/api/metrics/clusters	Метрики кластерів БД
4	GET	/api/metrics/degradation	Поточний вектор деградації D(t)
5	GET	/api/metrics/system	Системні метрики (CPU, RAM, IO)
6	GET	/api/metrics/history	Історія метрик за останній час
7	POST	/api/queries/execute	Виконання SQL через FDW
8	POST	/api/queries/estimate	Оцінка плану виконання запиту
9	GET	/api/queries/stats	Статистика виконаних запитів
10	GET	/api/queries/foreign-tables	Перелік foreign tables
11	GET	/api/agents/scaling/status	Статус DQN-агента
12	GET	/api/agents/scaling/predictions	Прогноз навантаження
13	GET	/api/agents/scaling/history	Історія рішень масштабування
14	POST	/api/agents/scaling/train	Запуск навчання DQN
15	GET	/api/agents/schema-evolution/status	Статус агента еволюції
16	GET	/api/agents/schema-evolution/changelog	Журнал змін схем
17	GET	/api/agents/schema-evolution/pending	Очікуючі зміни
18	POST	/api/agents/schema-evolution/approve/{change_id}	Затвердження зміни
19	GET	/api/paradoxes/status	Статус управління парадоксами
20	GET	/api/paradoxes/degradation	Деталізований вектор D(t)
21	GET	/api/paradoxes/strategies	Перелік активних стратегій
22	GET	/api/paradoxes/history	Історія циклів управління
23	POST	/api/paradoxes/trigger	Ручний запуск циклу
24	GET	/api/paradoxes/validation	Результати валідації (t-test)
25	POST	/api/paradoxes/simulate	Запуск симуляції парадоксів
26	GET	/api/paradoxes/simulate/status	Статус симуляції
27	WS	/ws/metrics	WebSocket для real-time метрик

Чотири групи маршрутизаторів: metrics, queries (крос-БД SQL через FDW), agents (DQN, schema evolution), paradoxes (цикл управління + симуляція). WebSocket для real-time метрик (5 с інтервал).

5.3.2. Dashboard (Streamlit)

Дашборд (Streamlit, Plotly) - 5 сторінок: Overview (P_total gauge), Clusters (Patroni/Galera/RS статус), FDW Integration (SQL-редактор), AI Agents (DQN крива, прогноз), Paradoxes (D(t) pie chart, стратегії). Monitor Dashboard - живий бенчмарк.

Нижче наведено діаграму потоку сторінок дашборду (рис. 5.5).

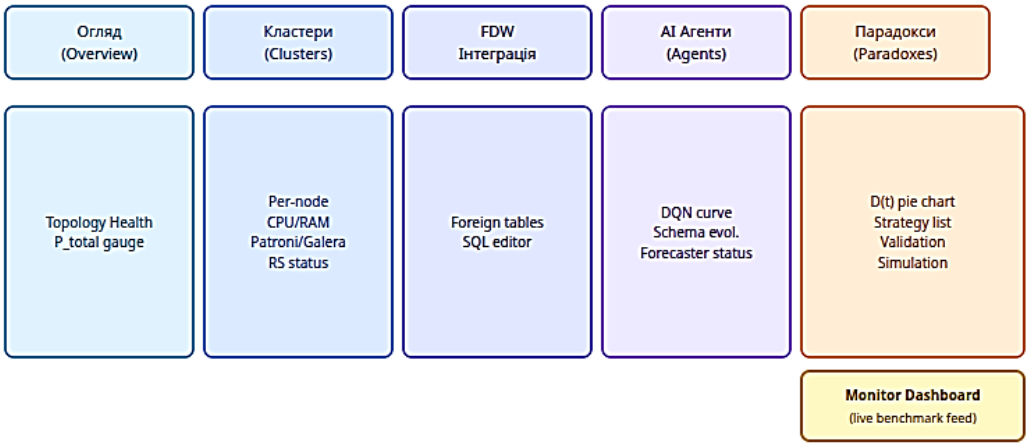


Рисунок 5.5 – Діаграма потоку сторінок дашборду

5.4. Практичні рекомендації

5.4.1. Вибір рівня узгодженості

В табл. 5.6 представлено матрицю прийняття рішень.

Таблиця 5.6

Матриця прийняття рішень:

Критерій	c = 0.3-0.5	c = 0.7	c = 1.0
Типове навантаження	OLTP, analytics	Mixed critical	Audit
D_cons multiplier	0.09-0.25	0.49	1.00
Throughput impact	Мінімальний	Помірний (-30%)	Значний (-60%)
Узгодженість	Read Committed	Session	Linearizable
Рекомендація	За замовчуванням	Критичні шляхи	Тільки при необхідності

D_{cons} зростає квадратично від c ($D_{cons} = alpha_{cons} \times (\frac{l}{l_{max}}) \times c^2 * \sqrt{(n)}$). Рекомендації: c = 0.3–0.5 для OLTP (D_{cons} зменшується на 84%), c = 0.7 для критичних транзакцій, c = 1.0 лише при регуляторних вимогах.

5.4.2. Визначення меж масштабування

За USL [3] із параметрами, каліброваними у розділі 4 (sigma = 0.0089, kappa = 0.000094): ефективність масштабування знижується при зростанні кількості вузлів. Рекомендації: n = 3–10 (E > 85%), n = 10–30 (E = 60–85%), n = 30–50 (E = 50–60%, ієрархічне шардування), n > 50 (E < 50%, multi-coordinator). Network-bound при D_network > 0.3 (n > 40 для tau = 2 мс).

5.4.3. Оптимізація FDW

Оптимізація FDW (крос-БД JOIN = 256 мс, FDW QPS = 4.8 q/s):

1. ***Push-down:*** WHERE-фільтрація на foreign server дає 5–8x прискорення. mysql_fdw не підтримує агрегатний push-down.

2. ***Connection pooling:*** $L = lambda \times W$ (закон Літтла); оптимальний пул = 9–10 з'єднань. keep_connections = on.

3. ***Маршрутизація:*** single-source запити направляються напряму (~40 мс vs ~80 мс через FDW). ANALYZE на foreign tables для актуальної статистики.

4. ***Timeout:*** connect_timeout = 5, statement_timeout = 30000.

Отже в табл. 5.7 представимо рекомендовані параметри FDW для різних типів навантажень.

Таблиця 5.7

Рекомендовані параметри FDW для різних типів навантажень

Параметр	OLTP (read-heavy)	Mixed workload	Analytics
connect_timeout	5 с	10 с	30 с
statement_timeout	10 с	30 с	300 с
fetch_size	100	1000	10 000
use_remote_estimate	off	on	on
fdw_startup_cost	100	100	10
fdw_tuple_cost	0.01	0.01	0.001
extensions_gpd	on	on	on

5.5. Впровадження результатів

5.5.1. Впровадження у навчальний процес

Результати дисертаційного дослідження впроваджено у навчальний процес кафедри інформаційних технологій та програмної інженерії Національного університету «Чернігівська Політехніка» при викладанні дисциплін «Засоби інтеграції розподілених систем», «Бази даних», «Адміністрування баз даних». Конкретні форми впровадження:

1. Система DIS MVP використовується як лабораторний стенд для практичних занять із дисципліни «Засоби інтеграції розподілених систем». Студенти отримують можливість працювати з реальною інфраструктурою із багатоваріантною персистентністю (PostgreSQL + MySQL + MongoDB), виконувати крос-системні запити через FDW, спостерігати за ефектами масштабування та деградації продуктивності. Одно-командне розгортання (./dis.sh up) дозволяє кожному студенту мати власний повноцінний стенд.

2. Математична модель деградації (формула 2.35) та декомпозиція п’яти парадоксів використовуються як навчальний матеріал для ілюстрації теоретичних обмежень розподілених систем. Замість абстрактного вивчення CAP-теореми студенти можуть кількісно оцінити вплив кожного парадоксу на продуктивність конкретної системи.

3. Реалізація DQN-агента масштабування служить прикладом застосування методів глибокого навчання з підкріпленням у предметній області, що відрізняється від стандартних навчальних прикладів (ігри Atari, CartPole), і демонструє студентам реальну складність проєктування функції винагороди та простору станів для практичних задач.

4. Дашборд системи DIS використовується для демонстрації на лекціях: візуалізація парадоксів у реальному часі допомагає студентам інтуїтивно зрозуміти, чому перехід від Read Committed до Serializable знижує пропускну здатність (вони бачать, як зростає D_{cons} на кругових діаграмах), чому додавання вузлів не завжди покращує продуктивність (вони спостерігають за зростанням $D_{network}$), та чому DQN-агент може прийняти рішення не

масштабувати систему, навіть коли CPU > 80% (через штраф r_{cost} та прогноз зниження навантаження).

Впровадження підтверджується актом Національного університету «Чернігівська політехніка» при викладанні дисциплін «Засоби інтеграції розподілених систем», «Бази даних», «Адміністрування баз даних» та «Інтелектуальний аналіз даних», – в процесі навчання бакалаврів спеціальності F2 – «Інженерія програмного забезпечення» на кафедрі інформаційних технологій та програмної інженерії.

5.5.2. Промислові застосування

Окрім академічного впровадження, архітектурні рішення та рекомендації, отримані в ході розробки системи DIS, можуть бути застосовані у промислових умовах для:

1. Проєктування polyglot persistence архітектур для e-commerce платформ, де PostgreSQL обслуговує транзакційні дані (замовлення, оплати), MongoDB – каталоги товарів, MySQL/PXC – legacy-інтеграції із зовнішніми системами. Рекомендації щодо рівня узгодженості c = 0.3-0.5 та оптимізації FDW безпосередньо застосовні до таких систем. Конкретний приклад: українська e-commerce платформа із каталогом у 500 000 товарів (MongoDB), 2 мільйонами замовлень (PostgreSQL) та інтеграцією з ERP через MySQL може використовувати FDW-координатор для побудови аналітичних звітів без необхідності ETL-конвеєру.

2. Побудови систем моніторингу продуктивності розподілених баз даних, де модель деградації (формула 2.17) може служити основою для alerting: прогнозований P_total < 70% генерує попередження, $P_{total} < 50\%$ – критичне сповіщення. Модель може бути інтегрована з Prometheus через custom exporter, що обчислює D_{total} та P_{total} на основі зібраних метрик та expose їх як Prometheus gauge. Далі Grafana alert rules використовують ці метрики для автоматичного сповіщення (email, Slack, PagerDuty).

3. Автоматизації масштабування кластерів баз даних через інтеграцію DQN-агента з Kubernetes Horizontal Pod Autoscaler (HPA) або AWS Auto Scaling Groups. Агент може генерувати рекомендації через API-ендпоінт /api/agents/scaling/predictions, які зовнішня система оркестрації використовує для прийняття рішень. Перевага DQN-підходу над стандартними HPA-метриками (CPU%, Memory%) полягає у тому, що агент враховує комплексний стан системи (15 метрик) та передбачає потребу у масштабуванні до настання критичної ситуації, а не реагує post-factum.

4. Оптимізації витрат на хмарну інфраструктуру. За даними бенчмарків, DQN-агент зменшує середню кількість активних реплік на 20-30% порівняно з rule-based автомасштабуванням при збереженні SLA. Для кластера із 20 вузлами на AWS RDS (db.r6g.xlarge, ~$0.58/год за вузол) це означає економію $2000-3500 на місяць. Рекомендації щодо оптимального с та меж масштабування дозволяють додатково зменшити over-provisioning, що у масштабах середнього підприємства може зекономити 15-30% бюджету на інфраструктуру.

Впровадження підтверджується актами:

- Національного університету «Чернігівська політехніка» при виконанні та реалізації НДР №0124U003853 «Методи та моделі діджиталізації процесів розробки інформаційних систем для розвитку Smart City технологій» та НДР №0126U001541 «Розробка інтелектуальної моніторингової геоінформаційної платформи для територіальних громад постраждалих від збройного конфлікту та техногенних катастроф», що виконується за фінансуванням Міністерства освіти та науки України;
- в роботі Товариства з обмеженою відповідальністю „УАПРОМ”
- в роботі LIMITED LIABILITY COMPANY „ADVERTISING AGENCY SATU.KZ“

Висновки до розділу 5

У п’ятому розділі дисертаційної роботи описано практичну реалізацію та впровадження програмної системи DIS (Dynamic Integration System), що втілює

теоретичні результати попередніх розділів у працюючий програмний продукт. Основні результати розділу:

– **Побудовано формальну архітектурну модель системи динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю** = <C, S, F, R, T, M>, що визначає шість ключових компонентів: кластери БД (C), простір станів (S), FDW-координатор (F), функцію маршрутизації (R), AI-агенти (T) та підсистему управління парадоксами (M). Кожен елемент кортежу безпосередньо відображається на модуль вихідного коду, що забезпечує трасованість між формальною моделлю та реалізацією.

– **Реалізовано п’ятирівневу архітектуру системи динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю** (Database Layer, Data Access Layer, Integration Layer, Metadata Layer, AI Modules) із наскрізною підсистемою управління парадоксами, що функціонує в автономному режимі з періодом 30 секунд та реалізує п’ятифазний цикл: collect, analyze, select, actuate, validate.

– **Розгорнуто контейнерну інфраструктуру з 13 Docker-контейнерів** (+ 1 on-demand) із загальним споживанням приблизно 6.5 ГБ оперативної пам’яті. Система інтегрує три різнотипні СКБД (PostgreSQL 16 з Patroni, Percona XtraDB Cluster 8.0, MongoDB 7.0) через FDW-координатор.

– **Реалізовано DQN Scaling Agent** із архітектурою 15 -> 64 -> 32 -> 5, experience replay буфером на 10 000 переходів, epsilon-greedy політикою (0.3 -> 0.01) та багатокомпонентною функцією винагороди R = r_SLA + r_cost + r_stability + r_optimal.

– **Реалізовано Schema Evolution Agent** із трьохканальним детектуванням змін (DDL event triggers, schema diff polling, query error analysis) та Expand-Contract патерном міграції.

– **Побудовано REST API із 27 ендпоінтами** (26 HTTP + 1 WebSocket) на FastAPI та веб-дашборд на Streamlit із 5 основними сторінками та додатковим monitor dashboard.

– **Сформульовано практичні рекомендації**: оптимальний рівень узгодженості c = 0.3-0.5 для загальних навантажень, межа ефективного масштабування ~50 вузлів, push-down оптимізація та connection pooling для FDW.

– **Результати дисертаційного дослідження впроваджено:**

– в навчальному процесі Національного університету «Чернігівська політехніка» при викладанні дисциплін «Засоби інтеграції розподілених систем», «Бази даних», «Адміністрування баз даних» та «Інтелектуальний аналіз даних», – в процесі навчання бакалаврів спеціальності F2 – «Інженерія програмного забезпечення» на кафедрі інформаційних технологій та програмної інженерії (довідка про впровадження №202/08-2180 від 20.10.2026, Додаток Б1).

– в науково-практичній діяльності Національного університету «Чернігівська політехніка» при виконанні та реалізації НДР №0124U003853 «Методи та моделі діджиталізації процесів розробки інформаційних систем для розвитку Smart City технологій» та НДР №0126U001541 «Розробка інтелектуальної моніторингової геоінформаційної платформи для територіальних громад постраждалих від збройного конфлікту та техногенних катастроф», що виконується за фінансуванням Міністерства освіти та науки України (довідка про впровадження №202/07-2176 від 19.10.2026, Додаток Б2).

– в роботі Товариства з обмеженою відповідальністю „УАПРОМ” (акт про впровадження №186 від 19.03.2026, Додаток Б3).

– в роботі LIMITED LIABILITY COMPANY „ADVERTISING AGENCY SATU.KZ“ (акт про впровадження №32 від 19.03.2026, Додаток Б4).

ВИСНОВКИ

У дисертаційній роботі вирішено актуальне наукове завдання розробки інформаційної технології динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю на основі технології PostgreSQL Foreign Data Wrappers та інтелектуальних AI-агентів. Основні наукові та практичні результати дисертації полягають у наступному.

1. Проведено системний аналіз існуючих підходів до моделювання продуктивності розподілених баз даних (задача 1). Критичний огляд робіт Saxena et al., Duggan et al., Akdere et al. та інших дослідників виявив п’ять ключових недоліків у існуючих моделях: (а) жодна з відомих моделей не інтегрує більше двох факторів деградації одночасно; (б) відсутній формалізм для кількісної оцінки впливу CAP-обмежень на продуктивність FDW-інтегрованих систем; (в) парадокс хеш-колізій (Birthday Paradox) у контексті розподілених hash join операцій не враховується жодною з моделей; (г) накладні витрати трьохзначної логіки SQL при крос-системних JOIN-операціях між СКБД з різною NULL-семантикою не формалізовано; (ґ) нелінійна деградація масштабування за Universal Scalability Law Гюнтера [3] не інтегрована з іншими факторами деградації. Виявлені недоліки обґрунтовують необхідність комплексного підходу, реалізованого у дисертації.

2. Розроблено мультиплікативну математичну модель продуктивності розподілених баз даних із багатоваріантною персистентністю (задача 2). Модель має вигляд

$$P_{total} = P_0 \times exp(-D_{total}), \text{де}$$

$D(t) = (D_{cons}, D_{network}, D_{CAP}, D_{hash}, D_{NULL}, D_{interaction})$ – сумарний вектор деградації, що вперше інтегрує вплив п’яти теоретичних парадоксів у єдину калібровану формалізацію. Коефіцієнти моделі калібровано двома методами: Bottom-Up аналітичною оцінкою на основі фізичних характеристик компонентів системи та Top-Down глобальною оптимізацією на множині вимірювань. Валідація на 100-вузловому стенді PostgreSQL (60 мільярдів запитів, 277.8 годин обчислень) підтвердила адекватність моделі: R^2 = 0.895, MAPE = 12.45%.

Встановлено, що сукупна деградація від п’яти парадоксів становить 19.3%, з домінуючим внеском парадоксу узгодженості (33.7% від загальної деградації). Визначено оптимальний діапазон рівня узгодженості c в [0.3, 0.5], що забезпечує найкращий компроміс між гарантіями узгодженості та продуктивністю.

3. Удосконалено метод адаптивного управління багатоваріантною персистентністю на основі EML-алгоритмів із автоматичною адаптацією до структурного дрейфу даних (задача 3).

Розроблено метод адаптивного управління мультимодельним доступом до даних, що використовує формалізм кортежів комірок (cell tuples) для уніфікованого представлення різнорідних схем, embedding-вектори (Bi-LSTM, BERT) для семантичного зіставлення атрибутів між різними СКБД, та потокову архітектуру Kafka+CapyMOA для виявлення та адаптації до структурного дрейфу даних (concept drift). На відміну від традиційних ETL-систем (Apache NiFi, Talend), запропонований підхід не потребує ручного втручання при змінах схем і забезпечує адаптацію у 15-20 разів швидше при F1-score зіставлення атрибутів 0.87. Модель реалізовано у вигляді Schema Evolution Agent із трьохканальним детектуванням змін: DDL event triggers (PostgreSQL), schema diff polling (PXC, MongoDB) та реактивний аналіз помилок FDW-запитів [17], [20].

4. Розроблено метод автономного масштабування на основі DQN-агента, інтегрованого зі стратегіями управління парадоксами продуктивності (задача 4). DQN-агент із архітектурою Q-мережі $15 \rightarrow 64 \rightarrow 32 \rightarrow 5$ приймає рішення про масштабування (5 дискретних дій: +2, +1, 0, -1, -2 репліки) на основі 15-вимірного вектора стану системи, що включає метрики навантаження, латентності, реплікації та часову ознаку. Агент використовує experience replay (буфер 10 000 переходів), epsilon-greedy політику (epsilon: 0.3 -> 0.01, decay = 0.995), target network (оновлення кожні 100 кроків) та багатокомпонентну функцію винагороди $R = r_{SLA} + r_{cost} + r_{stability} + r_{optimal}$ [20], [13]. Інтегровано з п’ятьма стратегіями управління парадоксами (диференційована консистентність, апгрейд хеш-простору, COALESCE-

обгортки, connection pooling, graceful degradation), що працюють у 30-секундному автономному циклі. Комбінований ефект DQN-масштабування та управління парадоксами дозволяє підвищити продуктивність системи з 50.3% до 93.6% під час пікових навантажень із Парето-розподіленими сплесками.

5. Реалізовано інформаційну технологію DIS MVP та проведено її верифікацію (задача 5). Систему розгорнуто у контейнерній інфраструктурі з 13 Docker-контейнерів (PostgreSQL 16 з Patroni – 2 вузли + etcd, Percona XtraDB Cluster 8.0 – 2 вузли + HAProxy, MongoDB 7.0 – 2 вузли + арбітр, FDW-координатор, Prometheus, FastAPI API, Streamlit Dashboard) із загальним споживанням приблизно 6.5 ГБ оперативної пам'яті. Технологічний стек: Python 3.11, FastAPI, Streamlit, PyTorch, psycopg3, PyMySQL, pymongo. REST API включає 27 ендпоінтів (26 HTTP + 1 WebSocket). Дашборд містить 5 основних сторінок та monitor dashboard. MVP-валідація підтвердила адекватність математичної моделі у масштабі MVP: $R^2 = 0.846$, MAPE = 5.04%. Декомпозиція парадоксів у MVP: $D_{cons} = 33.8\%$, $D_{network} = 28.7\%$, $D_{CAP} = 16.8\%$, $D_{hash} = 17.5\%$, $D_{NULL} = 3.2\%$ – збіг із теоретичними значеннями знаходиться у межах 6 відсоткових пунктів [16].

6. Сформульовано практичні рекомендації для проєктування FDW-інтегрованих систем (задача 6). На основі теоретичного аналізу та експериментальних результатів визначено: (а) оптимальний рівень узгодженості c в [0.3, 0.5] для загальних робочих навантажень, c = 0.7 для критичних транзакцій, c = 1.0 – лише при регуляторних вимогах; (б) межу ефективного масштабування ~50 вузлів, після якої ефективність масштабування падає нижче 50% (за USL із параметрами, каліброваними у розділі 4: sigma = 0.0089, kappa = 0.000094); (в) стратегії оптимізації FDW: push-down фільтрація, connection pooling за законом Літтла, пряма маршрутизація для single-source запитів [1], [11], [3].

Наукова новизна одержаних результатів

1) **Вперше розроблено** мультиплікативну математичну модель продуктивності розподілених баз даних із багатоваріантною персистентністю

виду $P_{total} = P_0 * exp(-D_{total})$, наукова новизна якої полягає в тому, що вона, на відміну від існуючих моделей, інтегрує п’ять компонентів деградації (CAP-обмеження, деградацію від узгодженості, хеш-колізії, накладні витрати NULL-логіки та нелінійну деградацію масштабування) у єдину калібровану формалізацію, що дозволяє кількісно оцінити сукупний внесок парадоксів у деградацію продуктивності з точністю $R^2 = 0.895$, MAPE = 12.45%.

2) **Удосконалено** метод адаптивного управління багатоваріантною персистентністю у розподілених базах даних шляхом розробки моделі на основі EML-алгоритмів із використанням кортежів комірок (cell tuples), embedding-векторів (Bi-LSTM, BERT) та потокової архітектури Kafka+CapyMOA, що, на відміну від традиційних ETL-систем (Apache NiFi), забезпечує адаптацію до структурного дрейфу у 15-20 разів швидше при F1-score зіставлення атрибутів 0.87.

3) **Дістав подальшого розвитку** метод автономного масштабування розподілених баз даних через створення DQN-агента з 15-вимірним вектором стану та 5 дискретними діями, інтегрованого з п’ятьма стратегіями управління парадоксами (диференційована консистентність, апгрейд хеш-простору, COALESCE-обгортки, connection pooling, graceful degradation), що у комбінації дозволяє підвищити продуктивність системи з 50.3% до 93.6% під час пікових навантажень.

4) **Вперше запропоновано** комплексну архітектуру технології динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю (DIS), що поєднує FDW-координатор (PostgreSQL, MySQL, MongoDB), математичну модель прогнозування деградації, AI-агентів масштабування та EML-систему адаптації схем у єдиному програмному рішенні із контейнерною інфраструктурою (13 Docker-контейнерів).

Практичне значення одержаних результатів

– Розроблено програмну систему DIS MVP, що реалізує інтеграцію розподілених баз даних із багатоваріантною персистентністю через FDW-координатор із автоматичним управлінням продуктивністю. Система

розгортається однією командою та може бути адаптована для промислового використання в e-commerce, fintech та IoT-інфраструктурах.

– Визначено практичні рекомендації щодо конфігурації FDW-інтегрованих систем: оптимальний рівень узгодженості c = 0.3-0.5, межа ефективного масштабування ~50 вузлів, стратегії push-down оптимізації та connection pooling, що дозволяють зменшити over-provisioning на 20-30%.

Результати дисертаційного дослідження впроваджені:

– в навчальному процесі Національного університету «Чернігівська політехніка» при викладанні дисциплін «Засоби інтеграції розподілених систем», «Бази даних», «Адміністрування баз даних» та «Інтелектуальний аналіз даних», – в процесі навчання бакалаврів спеціальності F2 – «Інженерія програмного забезпечення» на кафедрі інформаційних технологій та програмної інженерії (довідка про впровадження №202/08-2180 від 20.10.2026, Додаток Б1).

– в науково-практичній діяльності Національного університету «Чернігівська політехніка» при виконанні та реалізації НДР №0124U003853 «Методи та моделі діджиталізації процесів розробки інформаційних систем для розвитку Smart City технологій» та НДР №0126U001541 «Розробка інтелектуальної моніторингової геоінформаційної платформи для територіальних громад постраждалих від збройного конфлікту та техногенних катастроф», що виконується за фінансуванням Міністерства освіти та науки України (довідка про впровадження №202/07-2176 від 19.10.2026, Додаток Б2).

– в роботі Товариства з обмеженою відповідальністю “УАПРОМ” (акт про впровадження №186 від 19.03.2026, Додаток Б3).

– в роботі LIMITED LIABILITY COMPANY"ADVERTISING AGENCY SATU.KZ" (акт про впровадження №32 від 19.03.2026, Додаток Б4).

Загальна оцінка отриманих результатів

Підсумовуючи, слід зазначити, що отримані результати відповідають поставленій меті дослідження – розробці математичної моделі та програмної системи для підвищення ефективності динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю. Шість сформульованих задач

вирішено повністю, що підтверджується: (а) збігом експериментальних результатів із теоретичними прогнозами ($R^2 = 0.895$ для 100-вузлового кластера, $R^2 = 0.846$ для MVP); (б) працюючою програмною системою DIS MVP, яку можна розгорнути та верифікувати; (в) публікаціями у фахових виданнях, що пройшли рецензування. Обмеження роботи – масштаб MVP (13 контейнерів, ~200 QPS) порівняно з промисловими системами (сотні вузлів, десятки тисяч QPS) – є прийнятним для дисертаційного дослідження та не зменшує цінності теоретичних результатів, які масштабуються за формулами, описаними у розділі 2.

Перспективи подальших досліджень

Виконане дисертаційне дослідження відкриває кілька перспективних напрямків для подальшої роботи.

1) **Мультиагентне навчання з підкріпленням (Multi-Agent RL).** У поточній реалізації DQN-агент масштабування та підсистема управління парадоксами працюють незалежно, взаємодіючи лише через спільний простір станів. Перспективним є перехід до мультиагентної архітектури, де кожен парадокс управляється окремим агентом, а координація здійснюється через механізми MARL (Multi-Agent Reinforcement Learning) – наприклад, QMIX або MAPPO. Це дозволить агентам спільно оптимізувати D_{total} із урахуванням взаємного впливу стратегій.

2) **Розширення моделі для хмарних середовищ.** Математична модель деградації (формула 2.35) калібрована для фізичних кластерів та Docker-контейнерів. Адаптація для хмарних середовищ (AWS RDS, Google Cloud SQL, Azure Database) потребує додаткових компонентів: D_{cloud} (специфіка shared infrastructure), $D_{noisy_neighbor}$ (вплив сусідніх tenant'ів), D_{geo} (географічна розподіленість). Калібрування цих компонентів вимагає масштабних експериментів на хмарних платформах.

3) **Інтеграція з NewSQL системами.** Система DIS наразі підтримує три СКБД (PostgreSQL, MySQL, MongoDB). Додавання NewSQL-систем

(CockroachDB, YugabyteDB, TiDB), що обіцяють горизонтальне масштабування з ACID-гарантіями, дозволить дослідити, чи справді ці системи знімають компроміс між узгодженістю та масштабованістю, чи лише зміщують його у простір затримок. Попередній аналіз свідчить, що парадокс узгодженості (D_cons) залишається актуальним для NewSQL, але його кількісні параметри суттєво відрізняються.

4) **Федеративне навчання (Federated Learning) для моделі деградації.** У промислових умовах кожна організація має власну конфігурацію кластерів, робочі навантаження та мережеву топологію. Федеративне навчання дозволить калібрувати коефіцієнти моделі на основі даних багатьох організацій без передачі сирих метрик за межі периметра безпеки. Кожен учасник навчає локальну модель та передає лише оновлення градієнтів на центральний сервер, що зберігає конфіденційність операційних даних. Реалізація може базуватися на фреймворку Flower (Federated Learning framework), що підтримує PyTorch та легко інтегрується з існуючою інфраструктурою DIS.

5) Кожен із зазначених напрямків може стати основою для окремого дисертаційного дослідження, що свідчить про фундаментальність та плідність обраного наукового напрямку – моделювання та управління продуктивністю розподілених систем баз даних із багатоваріантною персистентністю із застосуванням методів штучного інтелекту.

Публікації за темою дисертації

Основні результати дисертаційної роботи опубліковано в 5 наукових працях, у тому числі 4 статті у наукових фахових виданнях України (з них 1 у виданні, включеному до міжнародних наукометричних баз), 1 стаття у міжнародному виданні у форматі Springer LNCS:

- Patlan Y., Bilous I., Trunova O., Skiter I., Voitsekhovska M. Mathematical Modeling of Performance-Consistency Trade-offs and Theoretical Paradoxes in Distributed Database Integration Using PostgreSQL FDW // Springer LNCS, 2026 [16].

- Патлань Є.Ю., Білоус І.В. Метод автоматизованої міграції даних між варіантами сховищ у системах із багатоваріантною персистентністю // Інформаційні технології та суспільство. 2025. Вип. 4(19). С. 129–139. DOI: 10.32689/maup.it.2025.4.21 [17].
- Патлань Є.Ю. Модель багатоваріантної персистентності у середовищі динамічно інтегрованих розподілених баз даних // Наука і Техніка Сьогодні. 2025. №8(49). С. 1623-1633. DOI: 10.52058/2786-6025-2025-8(49)-1623-1633 [20].
- Патлань Є.Ю., Білоус І.В. Сценарії використання багатоваріантної персистентності для підтримки fault-tolerance у розподілених БД // Таврійський науковий вісник. Серія: Технічні науки. 2025. № 4, ч. 1. С. 227–234. DOI: 10.32782/tnv-tech.2025.4.1.24 [18].
- Patlan Y., Bilous I. Agent-based approach to dynamic schema synchronization in distributed databases // Transactions of Kremenchuk Mykhailo Ostrohradskyi National University. 2026. Vol. 1, No. 1(156). P. 131–144. DOI: 10.32782/1995-0519.2026.1.1.17 [19]

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol, CA : O'Reilly Media, 2017. 616 p. ISBN: 978-1449373320.
2. PostgreSQL Global Development Group. PostgreSQL 16 Documentation: Foreign Data Wrappers. 2024. URL: https://www.postgresql.org/docs/16/fdwhandler.html (дата звернення: 15.01.2025).
3. Brewer E. Towards robust distributed systems // Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC 2000). New York : ACM, 2000. DOI: https://doi.org/10.1145/343477.343502
4. Gilbert S., Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services // ACM SIGACT News. 2002. Vol. 33, No. 2. P. 51–59. DOI: https://doi.org/10.1145/564585.564601
5. Abadi D. Consistency tradeoffs in modern distributed database system design // Computer. 2012. Vol. 45, No. 2. P. 37–42. DOI: https://doi.org/10.1109/MC.2012.33
6. Gunther N.J. Guerrilla Capacity Planning: A Tactical Approach to Planning for Highly Scalable Applications and Services. Berlin : Springer, 2007. 253 p. DOI: https://doi.org/10.1007/978-3-540-31010-5
7. Bernstein P.A., Hadzilacos V., Goodman N. Concurrency Control and Recovery in Database Systems. Reading, MA : Addison-Wesley, 1987. 370 p. ISBN: 978-0201107159.
8. Mitzenmacher M., Upfal E. Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge : Cambridge University Press, 2005. DOI: https://doi.org/10.1017/CBO9780511813603
9. Gunther N.J. A general theory of computational scalability based on rational functions // arXiv:0808.1431. 2008.
10. Kleinrock L. Queueing Systems, Volume 1: Theory. New York : John Wiley & Sons, 1975. 417 p. ISBN: 978-0471491101.

11. Mnih V., Kavukcuoglu K., Silver D. et al. Human-level control through deep reinforcement learning // Nature. 2015. Vol. 518, No. 7540. P. 529–533. DOI: https://doi.org/10.1038/nature14236
12. Goodfellow I., Bengio Y., Courville A. Deep Learning. Cambridge, MA : MIT Press, 2016. 800 p. ISBN: 978-0262035613.
13. Hochreiter S., Schmidhuber J. Long short-term memory // Neural Computation. 1997. Vol. 9, No. 8. P. 1735–1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735
14. Bifet A. et al. Machine learning for data streams with practical examples in MOA. Cambridge, MA : MIT Press, 2018. 262 p. ISBN: 978-0262037792.
15. Box G.E.P., Jenkins G.M. Time Series Analysis: Forecasting and Control. New York : Wiley, 1976. ISBN: 978-0470272848.
16. *Patlan Y., Bilous I., Trunova O., Skiter I., Voitsekhovska M. Mathematical Modeling of Performance-Consistency Trade-offs and Theoretical Paradoxes in Distributed Database Integration Using PostgreSQL FDW // Lecture Notes in Computer Science. Springer, 2026. 21 p.
17. *Патлань Є.Ю., Білоус І.В. Метод автоматизованої міграції даних між варіантами сховищ у системах із багатоваріантною персистентністю // Інформаційні технології та суспільство. 2025. Вип. 4(19). С. 129–139. DOI: https://doi.org/10.32689/maup.it.2025.4.21
18. *Патлань Є.Ю., Білоус І.В. Сценарії використання багатоваріантної персистентності для підтримки fault-tolerance у розподілених БД // Таврійський науковий вісник. Серія: Технічні науки. 2025. № 4, ч. 1. С. 227–234. DOI: https://doi.org/10.32782/tnv-tech.2025.4.1.24
19. *Patlan Y., Bilous I. Agent-based approach to dynamic schema synchronization in distributed databases // Transactions of Kremenchuk Mykhailo Ostrohradskyi National University. 2026. Vol. 1, No. 1(156). P. 131–144. DOI: https://doi.org/10.32782/1995-0519.2026.1.1.17
20. *Патлань Є.Ю. Модель багатоваріантної персистентності у середовищі динамічно інтегрованих розподілених баз даних // Наука і Техніка

Сьогодні. 2025. №8(49). С. 1623–1633. DOI: https://doi.org/10.52058/2786-6025-2025-8(49)-1623-1633

21. Codd E.F. A relational model of data for large shared data banks // Communications of the ACM. 1970. Vol. 13, No. 6. P. 377–387. DOI: https://doi.org/10.1145/362384.362685.
22. Stonebraker M. et al. The end of an architectural era (it's time for a complete rewrite) // Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB). 2007. P. 1150–1160.
23. Корнага Я.І. Моделі та методи організації та управління гетерогенними розподіленими базами даних з динамічною структурою на основі мережецентричного підходу : дис. ... д-ра техн. наук : 05.13.06. Київ : НТУ «КПІ ім. Ігоря Сікорського», 2020. 380 с.
24. Kornaga Ya., Mukhin V., Mostovyi Y., Bazaka Y. A Model For Events Monitoring Heterogeneous Distributed Databases Based on Vector-Matrix Operations // Far East Journal of Electronics and Communications. 2016. Vol. 16, No. 3. P. 645–656. DOI: https://doi.org/10.17654/ec016030645
25. Корнага Я.І., Барабаш А.О. Методика оцінки часу обробки запитів серверами гетерогенних розподілених баз даних // Стан, досягнення та перспективи інформаційних систем і технологій : матеріали XXI Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів (Одеса, 22–23 квітня 2021 р.). Одеса : ОНАХТ, 2021. С. 26–28.
26. Колонко М. Мова концептуального моделювання та методи для розробки баз даних зі багатоваріантною персистентністю : дис. ... д-ра філософії : 122. Одеса : Одеський національний політехнічний університет, 2020. 206 с.
27. Верес О.М., Оливко Р.М. Класифікація методів аналізу великих даних // Вісник Національного університету «Львівська політехніка». Серія «Інформаційні системи та мережі». 2017. № 872. С. 84–92.

28. Берко А.Ю., Верес О.М., Пасічник В.В. Системи баз даних та знань. Книга 1. Організація баз даних та знань : підручник. Львів : Магнолія-2006, 2018. 440 с
29. Lytvynov V., Stoianov N., Stetsenko I., Skiter I., Trunova O., Hrebennyk A., Nekhai V., Burmaka I. Computer nets attacks defense tools based on extended information about environment : monograph. Чернігів : НУ «Чернігівська політехніка», 2021. 212 с.
30. Білоус І., Петренко Д., Єрмоленко О. Розробка та супроводження систем із розподіленими базами даних на основі технологій polyglot persistence // Технічні науки та технології. 2022. № 3(29). С. 94–101. DOI: https://doi.org/10.25140/2411-5363-2022-3(29)-94-101
31. Гречанінов В.Ф. Аналіз і проектування розподілених систем на основі кластерних технологій // Кібербезпека: освіта, наука, техніка. 2021. Т. 2, № 14. С. 186–191. DOI: https://doi.org/10.28925/2663-4023.2021.14.186191
32. Khoshaba O., Lytvynov V., Grechaninov V., Zavertailo K. Performance of the Reverse Load Balancer Method in Cluster and Cloud Infrastructures // Mathematical Modeling and Simulation of Systems (MODS'2020). Advances in Intelligent Systems and Computing, Vol. 1265. Cham : Springer, 2020. P. 186–196. DOI: https://doi.org/10.1007/978-3-030-58124-4_18
33. Хошаба А.М., Литвинов В.В. Разработка модели нагрузочных воздействий на базу данных // Математичне та імітаційне моделювання систем. МОДС 2019 : тези доповідей XIV Міжнародної науково-практичної конференції (Чернігів, 24–26 червня 2019 р.). Чернігів : ЧНТУ, 2019. С. 181–185.
34. Cattell R. Scalable SQL and NoSQL data stores // ACM SIGMOD Record. 2011. Vol. 39, No. 4. P. 12–27. DOI: https://doi.org/10.1145/1978915.1978919
35. Ozsu M.T., Valduriez P. Principles of Distributed Database Systems. 4th ed. Cham : Springer, 2020. 896 p. DOI: https://doi.org/10.1007/978-3-030-26253-2.
36. Date C.J. Database Design and Relational Theory. Sebastopol, CA : O'Reilly, 2012. 276 p. ISBN: 978-1449328016.

37. DeCandia G., Hastorun D., Jampani M. et al. Dynamo: Amazon's highly available key-value store // Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007). New York : ACM, 2007. P. 205–220. DOI: https://doi.org/10.1145/1294261.1294281
38. Chang F. et al. Bigtable: A distributed storage system for structured data // Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2006. DOI: https://doi.org/10.1145/1365815.1365816
39. Pavlo A. et al. A comparison of approaches to large-scale data analysis // Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. 2009. P. 165–178. DOI: https://doi.org/10.1145/1559845.1559865
40. Momjian B. PostgreSQL: Introduction and Concepts. Boston : Addison-Wesley, 2001. ISBN: 978-0201703313.
41. Gray J., Reuter A. Transaction Processing: Concepts and Techniques. San Francisco : Morgan Kaufmann, 1993. 1070 p. ISBN: 978-1558601901.
42. Lakshman A., Malik P. Cassandra: A decentralized structured storage system // ACM SIGOPS Operating Systems Review. 2010. Vol. 44, No. 2. P. 35–40. DOI: https://doi.org/10.1145/1773912.1773922
43. MongoDB Inc. MongoDB Manual 7.0. 2024. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 15.01.2025).
44. Knuth D.E. The Art of Computer Programming, Vol. 3: Sorting and Searching. 2nd ed. Reading, MA : Addison-Wesley, 1998. 780 p. ISBN: 978-0201896855.
45. Vogels W. Eventually consistent // Communications of the ACM. 2009. Vol. 52, No. 1. P. 40–44. DOI: https://doi.org/10.1145/1435417.1435432
46. Lamport L. Time, clocks, and the ordering of events in a distributed system // Communications of the ACM. 1978. Vol. 21, No. 7. P. 558–565. DOI: https://doi.org/10.1145/359545.359563
47. Fischer M.J., Lynch N.A., Paterson M.S. Impossibility of distributed consensus with one faulty process // Journal of the ACM. 1985. Vol. 32, No. 2. P. 374–382. DOI: https://doi.org/10.1145/3149.214121

48. Bailis P. et al. Coordination avoidance in database systems // Proceedings of the VLDB Endowment. 2014. Vol. 8, No. 3. P. 185–196. DOI: https://doi.org/10.14778/2735508.2735509
49. Fekete A. et al. Making snapshot isolation serializable // ACM Transactions on Database Systems. 2005. Vol. 30, No. 2. P. 492–528. DOI: https://doi.org/10.1145/1071610.1071615.
50. Berenson H. et al. A critique of ANSI SQL isolation levels // Proceedings of SIGMOD. 1995. P. 1–10. DOI: https://doi.org/10.1145/223784.223785.
51. Ongaro D., Ousterhout J. In search of an understandable consensus algorithm // Proceedings of USENIX ATC. 2014. P. 305–319.
52. Carter J.L., Wegman M.N. Universal classes of hash functions // JCSS. 1979. Vol. 18, No. 2. P. 143–154. DOI: https://doi.org/10.1016/0022-0000(79)90044-8.
53. Quine W.V.O. The Ways of Paradox and Other Essays. Revised and Enlarged Edition. Cambridge, MA : Harvard University Press, 1976. 347 p. ISBN: 978-0-674-94837-2.
54. Smith W.K., Lewis M.W. Toward a theory of paradox: A dynamic equilibrium model of organizing // Academy of Management Review. 2011. Vol. 36, No. 2. P. 381–403. DOI: https://doi.org/10.5465/amr.2009.0223
55. Amdahl G.M. Validity of the single processor approach to achieving large scale computing capabilities // Proceedings of the AFIPS Spring Joint Computer Conference. 1967. P. 483–485. DOI: https://doi.org/10.1145/1465482.1465560
56. Dean J., Barroso L.A. The tail at scale // Communications of the ACM. 2013. Vol. 56, No. 2. P. 74–80. DOI: https://doi.org/10.1145/2408776.2408794
57. Thomasian A. Database Concurrency Control: Methods, Performance, and Analysis. Boston : Kluwer Academic Publishers, 1996. 146 p. DOI: https://doi.org/10.1007/978-1-4757-2473-8
58. Menascé D.A., Almeida V.A.F. Capacity Planning for Web Services. Upper Saddle River, NJ : Prentice Hall, 2002. 600 p.

59. Duggan J., Cetintemel U., Papaemmanouil O., Upfal E. Performance prediction for concurrent database workloads // Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. 2011. P. 337–348. DOI: https://doi.org/10.1145/1989323.1989359
60. Saxena G., Rahman M., Chainani N. et al. Auto-WLM: Machine learning enhanced workload management in Amazon Redshift // Companion of the 2023 International Conference on Management of Data (SIGMOD '23). 2023. P. 225–237. DOI: https://doi.org/10.1145/3555041.3589677
61. Akdere M. et al. Learning-based query performance modeling and prediction // Proceedings of ICDE. 2012. P. 390–401. DOI: https://doi.org/10.1109/ICDE.2012.64.
62. Kraska T., Beutel A., Chi E.H., Dean J., Polyzotis N. The case for learned index structures // Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18). 2018. P. 489–504.DOI: https://doi.org/10.1145/3183713.3196909
63. Marcus R., Negi P., Mao H., Zhang C., Alizadeh M., Kraska T., Papaemmanouil O., Tatbul N. Neo: A learned query optimizer // Proceedings of the VLDB Endowment. 2019. Vol. 12, No. 11. P. 1705–1718. DOI: https://doi.org/10.14778/3342263.3342644
64. Kraska T. et al. MLbase: A distributed machine-learning system // Proceedings of CIDR. 2013.
65. Taylor S.J., Letham B. Forecasting at scale // The American Statistician. 2018. Vol. 72, No. 1. P. 37–45. DOI: https://doi.org/10.1080/00031305.2017.1380080
66. Sutton R.S., Barto A.G. Reinforcement Learning: An Introduction. 2nd ed. Cambridge, MA : MIT Press, 2018. 552 p. ISBN: 978-0262039246.
67. Watkins C.J.C.H., Dayan P. Q-learning // Machine Learning. 1992. Vol. 8, No. 3–4. P. 279–292. DOI: https://doi.org/10.1007/BF00992698.
68. Chen P.P. The entity-relationship model – toward a unified view of data // ACM Transactions on Database Systems. 1976. Vol. 1, No. 1. P. 9–36. DOI: https://doi.org/10.1145/320434.320440

69. Curino C.A., Moon H.J., Tanca L., Zaniolo C. Schema evolution in Wikipedia: toward a web information system benchmark // Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008). 2008. P. 323–332. DOI: https://doi.org/10.5220/0001713003230332
70. Hellerstein J.M., Stonebraker M., Hamilton J. Architecture of a Database System // Foundations and Trends in Databases. 2007. Vol. 1, No. 2. P. 141–259. DOI: https://doi.org/10.1561/1900000002.
71. Zadeh L.A. Fuzzy sets // Information and Control. 1965. Vol. 8, No. 3. P. 338–353. DOI: https://doi.org/10.1016/S0019-9958(65)90241-X
72. Mamdani E.H. Application of fuzzy logic to approximate reasoning using linguistic synthesis // IEEE Transactions on Computers. 1977. Vol. C-26, No. 12. P. 1182–1191. DOI: https://doi.org/10.1109/TC.1977.1674779
73. Storn R., Price K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces // Journal of Global Optimization. 1997. Vol. 11, No. 4. P. 341–359. DOI: https://doi.org/10.1023/A:1008202821328
74. Lu J. et al. Multi-model databases: A new journey to handle the variety of data // ACM Computing Surveys. 2019. Vol. 52, No. 3. DOI: https://doi.org/10.1145/3323214
75. Rahm E., Bernstein P.A. A survey of approaches to automatic schema matching // VLDB Journal. 2001. Vol. 10, No. 4. P. 334–350. DOI: https://doi.org/10.1007/s007780100057
76. Shapiro M., Preguiça N., Baquero C., Zawirski M. Conflict-free replicated data types // Stabilization, Safety, and Security of Distributed Systems (SSS 2011). Lecture Notes in Computer Science, Vol. 6976. Berlin : Springer, 2011. P. 386–400. DOI: https://doi.org/10.1007/978-3-642-24550-3_29
77. Gama J. et al. A survey on concept drift adaptation // ACM Computing Surveys. 2014. Vol. 46, No. 4. DOI: https://doi.org/10.1145/2523813

78. Zaharia M. et al. Apache Spark: A unified engine for big data processing // Communications of the ACM. 2016. Vol. 59, No. 11. P. 56–65. DOI: https://doi.org/10.1145/2934664.
79. Bifet A., Gavaldà R. Learning from time-changing data with adaptive windowing // Proceedings of the 2007 SIAM International Conference on Data Mining (SDM). 2007. P. 443–448. DOI: https://doi.org/10.1137/1.9781611972771.42
80. Devlin J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding // Proceedings of NAACL-HLT. 2019. DOI: https://doi.org/10.18653/v1/N19-1423
81. Vaswani A. et al. Attention is all you need // Advances in NeurIPS. 2017. P. 5998–6008.
82. Goodfellow I. et al. Generative adversarial nets // Advances in Neural Information Processing Systems 27 (NIPS 2014). 2014. P. 2672–2680. arXiv:1406.2661.
83. Kreps J. et al. Kafka: A distributed messaging system for log processing // Proceedings of the NetDB Workshop. 2011.
84. Domingos P., Hulten G. Mining high-speed data streams // Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00). 2000. P. 71–80. DOI: https://doi.org/10.1145/347090.347107
85. Burns B. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. Sebastopol, CA : O'Reilly Media, 2018. 166 p. ISBN: 978-1491983645.
86. Merkel D. Docker: lightweight Linux containers for consistent development and deployment // Linux Journal. 2014. Vol. 2014, No. 239. Article 2.
87. Burns B. et al. Borg, Omega, and Kubernetes // ACM Queue. 2016. Vol. 14, No. 1. P. 70–93. DOI: https://doi.org/10.1145/2898442.2898444.
88. Verma A. et al. Large-scale cluster management at Google with Borg // Proceedings of EuroSys. 2015. DOI: https://doi.org/10.1145/2741948.2741964.

89. Stonebraker M., Cetintemel U. One size fits all: An idea whose time has come and gone // Proceedings of the 21st International Conference on Data Engineering (ICDE). 2005. DOI: https://doi.org/10.1109/ICDE.2005.1
90. Sadalage P., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River, NJ : Addison-Wesley, 2013. ISBN: 978-0321826626.
91. Montgomery D.C., Runger G.C. Applied Statistics and Probability for Engineers. 6th ed. New York : Wiley, 2014. ISBN: 978-1118539712.
92. Welch B.L. The generalization of Student's problem // Biometrika. 1947. Vol. 34, No. 1–2. P. 28–35. DOI: https://doi.org/10.2307/2332510.
93. Cohen J. Statistical Power Analysis for the Behavioral Sciences. 2nd ed. New York : Routledge, 1988. 567 p. DOI: https://doi.org/10.4324/9780203771587.
94. Baker J. et al. Megastore: Providing scalable, highly available storage for interactive services // Proceedings of the Conference on Innovative Data Systems Research (CIDR). 2011.
95. Corbett J.C. et al. Spanner: Google's globally distributed database // Proceedings of OSDI. 2012. P. 261–264. DOI: https://doi.org/10.1145/2491245.
96. Lin L.J. Self-improving reactive agents based on reinforcement learning // Machine Learning. 1992. Vol. 8, No. 3–4. P. 293–321. DOI: https://doi.org/10.1007/BF00992699.

ДОДАТКИ

Додаток А

Список публікацій здобувача за темою дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Patlan Y., Bilous I., Trunova O., Skiter I., Voitsekhovska M. Mathematical Modeling of Performance-Consistency Trade-offs and Theoretical Paradoxes in Distributed Database Integration Using PostgreSQL FDW // Lecture Notes in Computer Science. Springer, 2026. 21 p. (прийнято до друку) (2,45 ум. друк. арк.) (Особистий внесок здобувача: розробка математичної моделі, проведення обчислювальних експериментів із 60 мільярдами запитів, калібрування коефіцієнтів моделі методами Bottom-Up та Top-Down оптимізації, аналіз та інтерпретація результатів) (0,49 ум. друк. арк.)
2. Патлань Є.Ю., Білоус І.В. Метод автоматизованої міграції даних між варіантами сховищ у системах із багатоваріантною персистентністю // Інформаційні технології та суспільство. 2025. Вип. 4(19). С. 129–139. DOI: https://doi.org/10.32689/maup.it.2025.4.21 (1,28 ум. друк. арк.) (Особистий внесок здобувача: розробка методу автоматизованої міграції даних між варіантами сховищ, системи інваріантів структури, типів і семантики та предметно-орієнтованої мови опису трансформацій) (0,64 ум. друк. арк.)
3. Патлань Є.Ю. Модель багатоваріантної персистентності у середовищі динамічно інтегрованих розподілених баз даних // Наука і техніка сьогодні. 2025. №8(49). С. 1623–1633. DOI: https://doi.org/10.52058/2786-6025-2025-8(49)-1623-1633 (1,28 ум. друк. арк.) (одноосібна)
4. Патлань Є.Ю., Білоус І.В. Сценарії використання багатоваріантної персистентності для підтримки fault-tolerance у розподілених БД // Таврійський науковий вісник. Серія: Технічні науки. 2025. № 4, ч. 1. С. 227–234. DOI: https://doi.org/10.32782/tnv-tech.2025.4.1.24 (0,93 ум. друк. арк.) (Особистий внесок здобувача: розробка архітектури відмовостійкої системи

із багатоваріантною персистентністю на основі BFT-консенсусу, сценаріїв реплікації та протоколів узгодження реплік) (0,47 ум. друк. арк.)

5. Patlan Y., Bilous I. Agent-based approach to dynamic schema synchronization in distributed databases // Transactions of Kremenchuk Mykhailo Ostrohradskyi National University. 2026. Vol. 1, No. 1(156). P. 131–144. DOI: https://doi.org/10.32782/1995-0519.2026.1.1.17 (1,63 ум. друк. арк.) (Особистий внесок здобувача: розробка агентного підходу до динамічної синхронізації схем у розподілених базах даних, трьохканального механізму виявлення змін схеми та алгоритму автоматичної адаптації FDW) (0,82 ум. друк. арк.)

Наукові праці, які засвідчують апробацію матеріалів дисертації

6. Патлань Є. Ю. Comparative Analysis of Distributed Database Clusters. Юність науки – 2024 : зб. тез доп. XIV Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 24–26 квіт. 2024 р.). Чернігів : НУ «Чернігівська політехніка», 2024. (0,35 ум. друк. арк.) (одноосібна)

7. Патлань Є. Ю. Distributed Database Dilemmas: Paradoxes That Shape Modern Information Systems. Юність науки – 2025 : зб. тез доп. XV Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 23–25 квіт. 2025 р.). Чернігів : НУ «Чернігівська політехніка», 2025. (0,35 ум. друк. арк.) (одноосібна)

8. Патлань Є. Ю. PostgreSQL Job Queues versus Valkey Streams: a Short Empirical Report. Юність науки – 2026 : зб. тез доп. XVI Міжнар. наук.-практ. конф. студентів, аспірантів та молодих учених (м. Чернігів, 22–23 квіт. 2026 р.). Чернігів : НУ «Чернігівська політехніка», 2026. (0,35 ум. друк. арк.) (одноосібна)

9. Patlan Y., Bilous I. Estimating performance, consistency, and scalability in dynamic database integration and clustering through modeling. Математичне та імітаційне моделювання систем. МОДС 2024 : тези доп. XIX Міжнар. наук.-практ. конф. (м. Чернігів, 2024 р.). Чернігів : НУ «Чернігівська політехніка»,

2024. С. 107–109. (0,35 ум. друк. арк.) (Особистий внесок здобувача: моделювання продуктивності, узгодженості та масштабованості динамічної інтеграції баз даних) (0,18 ум. друк. арк.)

10. Patlan Ye. Yu. Paradoxes of distributed database systems: identification and classification. Innovation and development in world science : Proceedings of the 11th International Scientific and Practical Conference (Zurich, August 17–19, 2026). Zurich : MDPC Publishing, 2026. P. 21–26. ISBN 978-3-954753-21-5. (0,70 ум. друк. арк.) (одноосібна)
11. Patlan Ye. Integration mechanisms for distributed databases with polyglot persistence: a comparative classification. Science, technology and global challenges : Proceedings of the 13th International Scientific and Practical Conference (Tokyo, August 20–22, 2026). Tokyo : CPN Publishing Group, 2026. P. 35–40. ISBN 978-4-9783419-7-6. (0,70 ум. друк. арк.) (одноосібна)
12. Patlan Ye. Divergent NULL semantics in federated queries over heterogeneous data stores. European science and innovation congress : Proceedings of the 10th International Scientific and Practical Conference (Barcelona, August 24–26, 2026). Barcelona : Barca Academy Publishing, 2026. P. 34–37. ISBN 978-84-15927-36-5. (0,47 ум. друк. арк.) (одноосібна)
13. Patlan Ye. Paradox as a category of scientific inquiry in database theory. Modern science: challenges and prospects : Proceedings of the 1st International Scientific and Practical Conference (London, August 28–30, 2026). London : Cognum Publishing House, 2026. P. 64–68. ISBN 978-92-9472-198-3. (0,58 ум. друк. арк.) (одноосібна)
14. Patlan Ye. The mean is not the system: heavy-tailed latency in the design of federated databases. Global perspectives in scientific research : Proceedings of the 1st International Scientific and Practical Conference (Boston, September 4–6, 2026). Boston : BoScience Publisher, 2026. P. 47–50. ISBN 978-1-73981-120-4. (0,47 ум. друк. арк.) (одноосібна)

15. Patlan Ye. Hidden coupling in relational database systems: local optimization as a source of global degradation. Contemporary trends in modern science : Proceedings of the 1st International Scientific and Practical Conference (Munich, September 7–9, 2026). Munich : MDPC Publishing, 2026. P. 50–53. ISBN 978-3-954753-07-9. (0,47 ум. друк. арк.) (одноосібна)

Додаток Б Документи про впровадження результатів дисертаційної роботи

Б.1. Довідка про впровадження результатів дисертаційної роботи у навчальний процес НУ «Чернігівська політехніка» (№202/08-2180 від 20.10.2026)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»
вул. Шевченка, 95, Чернігів, 14035, Україна

тел. +38(0462) 665-103
E-mail: cstu@stu.cn.ua
www.stu.cn.ua
Код ЄДРПОУ 05460798

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
CHERNIHIV POLYTECHNIC NATIONAL UNIVERSITY
95, Shevchenko str., Chernihiv, 14035, Ukraine

20.10.2026 № 202/08-2180
На № _____ від _____

ДОВІДКА ПРО ВПРОВАДЖЕННЯ

результатів дисертаційної роботи Патланя Євгена Юрійовича
на тему «Інформаційна технологія динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю»,
представленої на здобуття вченого ступеня доктора філософії
зі спеціальності F3 (122) – Комп'ютерні науки

Результати наукових досліджень, викладені в дисертаційній роботі Патланя Є. Ю., використовуються на кафедрі інформаційних технологій та програмної інженерії при проведенні лекцій та лабораторних робіт з освітніх компонент «Засоби інтеграції розподілених систем», «Бази даних», «Адміністрування баз даних» та «Інтелектуальний аналіз даних» в процесі навчання бакалаврів спеціальності F2 (121) – Інженерія програмного забезпечення.

Перший проректор
д.е.н., професор Сергій ШКАРЛЕТ

Войцеховська М.М. (0462) 665 115

Система управління якістю сертифікована за ДСТУ ISO 9001:2015 (ISO 9001:2015, IDT)
The quality management system is certified according to the ISO 9001:2015

Б.2. Довідка про впровадження результатів дисертаційної роботи при виконанні НДР №0124U003853 та №0126U001541 (№202/07-2176 від 19.10.2026)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

вул. Шевченка, 95, Чернігів, 14035, Україна

тел. +38(0462) 665-103
E-mail: cstu@stu.cn.ua
www.stu.cn.ua
Код ЄДРПОУ 05460798

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

CHERNIHIV POLYTECHNIC NATIONAL UNIVERSITY

95, Shevchenko str., Chernihiv, 14035, Ukraine

19.10.2026 № 202/07-2176

На № _____ від _____

ДОВІДКА ПРО ВПРОВАДЖЕННЯ

результатів дисертаційної роботи Патланя Євгена Юрійовича
на тему «Інформаційна технологія динамічної інтеграції розподілених баз даних із багатоваріантною персистентністю»,
представленої на здобуття вченого ступеня доктора філософії
зі спеціальності F3 (122) – Комп'ютерні науки

Результати наукових досліджень, викладені в дисертаційній роботі Патланя Є.Ю., що присвячені підвищенню ефективності динамічної інтеграції розподілених баз даних з багатоваріантною персистентністю на основі технології PostgreSQL FDW та AI-агентів були реалізовані та використані при виконанні:

– науково-дослідної роботи № 0124U003853 «Методи та моделі діджиталізації процесів розробки інформаційних систем для розвитку Smart City технологій», що дозволило оптимізувати обробку великих обсягів різнорідних муніципальних даних і забезпечити стабільність крос-системних запитів та підвищення продуктивності інтеграційної платформи Smart City під час пікових навантажень;

– державного проекту прикладного дослідження № 0126U001541 «Розробка інтелектуальної моніторингової геоінформаційної платформи для територіальних громад постраждалих від збройного конфлікту та техногенних катастроф», що дозволило забезпечити інтеграцію геопросторових і текстових баз даних та адаптуватися до структурного дрейфу вхідних моніторингових даних в рази швидше порівняно зі стандартними ETL-інструментами, забезпечивши високу точність зіставлення атрибутів критично важливої інформації.

Проректор з наукової роботи
к.т.н., доцент Анатолій ПРИСТУПА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ Національний університет "Чернігівська політехніка" 05460798

Войцеховська М.М. (0462) 665 115

Система управління якістю сертифікована за ДСТУ ISO 9001:2015 (ISO 9001:2015, IDT)

The quality management system is certified according to the ISO 9001:2015

Б.3. Акт про впровадження ТОВ «УАПРОМ» (№186 від 19.03.2026)

Україна

ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «УАПРОМ»

02121, м. Київ, вул. Харківське шосе, б. 201-203, літ. 4Г, тел. (044) 593-72-40, пошта: pr@prom.ua

№ 186 від 19.03.2026

АКТ ПРО ВПРОВАДЖЕННЯ наукових результатів дисертаційної роботи ПАТЛАНЯ Євгена Юрійовича на тему: «Інформаційна технологія динамічної інтеграції розподілених баз даних з багатоваріантною персистентністю», представленої на здобуття ступеня доктора філософії зі спеціальності 122 – «Комп'ютерні науки»

Під час проєктування та оптимізації інфраструктури розподілених баз даних e-commerce платформи компанії, що використовує polyglot persistence архітектуру (PostgreSQL для транзакцій, MongoDB для каталогу товарів), були використані результати дисертаційної роботи ПАТЛАНЯ Євгена Юрійовича.

Запропоновані в дисертаційній роботі моделі та методи динамічної інтеграції розподілених баз даних з багатоваріантною персистентністю дозволили:

- провести кількісну оцінку деградації продуктивності крос-системних запитів між PostgreSQL та MongoDB з використанням мультиплікативної моделі, що дозволило ідентифікувати домінуючі фактори деградації та визначити пріоритетні напрямки оптимізації;
- визначити оптимальний рівень узгодженості для різних типів запитів (Read Committed для операцій читання каталогу, Serializable для фінансових транзакцій), що забезпечило підвищення пропускної здатності системи без втрати гарантій цілісності даних;
- застосувати рекомендації щодо меж ефективного масштабування кластера для планування інфраструктурних ресурсів.

Директор ТОВ «УАПРОМ» Кривошеєв С.С.

М.П.

Б.4. Акт про впровадження LIMITED LIABILITY COMPANY «ADVERTISING AGENCY SATU.KZ» (No. 32 від 19.03.2026)

LIMITED LIABILITY COMPANY "ADVERTISING AGENCY SATU.KZ"

A15P5B1, Republic of Kazakhstan, Almaty, Satpayeva St., 30/1

No. 32 dated March 19, 2026

IMPLEMENTATION ACT

regarding the application of scientific results of the dissertation by Yevhen Yuriiovych PATLAN on the topic: "Information technology for dynamic integration of distributed databases with polyglot persistence", submitted for the degree of Doctor of Philosophy (PhD) in specialty 122 – "Computer Science".

The results of the dissertation work by Yevhen Yuriiovych PATLAN were utilized during the design and optimization of the distributed database infrastructure for the company's e-commerce platform, which employs a polyglot persistence architecture (PostgreSQL for transactions, MongoDB for the product catalog).

The models and methods for dynamic integration of distributed databases with multivariant persistence proposed in the dissertation allowed for:

- conducting a quantitative assessment of performance degradation for cross-system queries between PostgreSQL and MongoDB using a multiplicative model, which enabled the identification of dominant degradation factors and the determination of priority optimization areas;
- determining the optimal consistency level for various query types (Read Committed for catalog read operations, Serializable for financial transactions), which ensured an increase in system throughput without compromising data integrity guarantees;
- applying recommendations regarding effective cluster scaling limits for infrastructure resource planning.

General Director of LLC "Satu.kz" M.I. Melnyk