ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Оптимізація алгоритмів попередньої підготовки Big Data в задачах візуалізації бізнес-процесів»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Руслан ШВЕЦЬ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Руслан ШВЕЦЬ

Керівник: Володимир САДОВЕНКО
канд. фіз-мат наук.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Швецю Руслану Петровичу

1. Тема кваліфікаційної роботи: «Оптимізація алгоритмів попередньої підготовки Big Data в задачах візуалізації бізнес-процесів»

керівник кваліфікаційної роботи Володимир САДОВЕНКО, канд. фіз.-мат. наук

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література щодо методів попередньої підготовки даних у системах Big Data, характеристики великих об’ємів даних, методи очищення, фільтрації, нормалізації та агрегації, вимоги до формату і частоти оновлення даних та вимоги до масштабованості.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз методів попередньої підготовки Big Data для задач бізнес-аналітики та візуалізації.
2. Розробка оптимізованого алгоритму попередньої підготовки Big Data для підвищення продуктивності систем візуалізації бізнес-процесів.
3. Оцінювання ефективності алгоритмів обробки великих масивів даних.
4. Реалізація програмного модуля (CHUB): ETL/ELT, коннектори, модулі інтеграції з Azure, схеми трансформації даних.
5. Проведення експериментальної валідації та порівняльного аналізу ефективності (пропускна здатність, затримки, узгодження даних).
6. Висновки, рекомендації щодо впровадження та подальших досліджень.

5. Перелік ілюстративного матеріалу: *презентація*

1. Порівняльна характеристика існуючих методів вирішення задачі.
2. Математична модель зваженого агрегату.
3. Архітектури обробки та управління даними Big Data.
4. Результати моделювання системи попередньої обробки даних.
5. Таблиці порівняльних показників (пропускна здатність, вартість розробки, вартість володіння)
6. Демонстрація роботи модуля на прикладі (скріншоти, вихідний код)

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10.2025-04.11.2025	
2	Вивчення матеріалів щодо попередньої обробки даних	05.11.2025-09.11.2025	
3	Дослідження методів підготовки даних	10.11.2025-14.11.2025	

4	Визначення навантажувального обчислення для проведення вимірювання ефективності оптимізації	15.11.2025-17.11.2025	
5	Реалізація програмного модуля (CHUB): модулі трансформації, узгодженості та інтеграції	18.11.2025-23.11.2025	
6	Проведення експериментальної валідації та порівняльного аналізу ефективності	24.11.2025-26.11.2025	
7	Оформлення роботи: вступ, висновки, реферат	27.11.2025-29.11.2025	
8	Розробка демонстраційних матеріалів	30.11.2025-19.12.2025	

Здобувач вищої освіти _____ Руслан ШВЕЦЬ
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 77 стор., 9 табл., 17 рис., 20 джерел.

Мета роботи - оптимізація алгоритмів попередньої підготовки Big Data в задачах візуалізації бізнес-процесів.

Об'єкт дослідження - процеси попередньої підготовки даних у системах зберігання та обробки Big Data.

Предмет дослідження - алгоритми, методи та підходи попередньої підготовки Big Data, що застосовуються для візуалізації бізнес-процесів.

У роботі використано методи аналізу та порівняння алгоритмів попередньої обробки Big Data, зокрема методи трансформації, нормалізації та зменшення розмірності даних. Розробку та програмну реалізацію моделі зваженого агрегата виконано з використанням алгоритмічних та експериментальних методів. Оцінювання ефективності системи здійснювалося шляхом навантажувального тестування з урахуванням впливу апаратного середовища. Для підвищення продуктивності застосовано оптимізацію запитів, розділення даних, кешування та зміну форматів зберігання. Аналіз результатів проведено за метриками пропускної здатності, затримки та узгодженості даних.

Проведено аналіз сучасних методів попередньої обробки даних у системах Big Data, зокрема підходів до трансформації, нормалізації та зменшення розмірності даних. Розглянуто алгоритми оптимізації запитів, методи розділення даних і кешування, а також підходи до вибору форматів зберігання з урахуванням їх впливу на пропускну здатність, затримку та узгодженість даних. На основі проведеного аналізу обґрунтовано доцільність використання моделі зваженого агрегата для підвищення продуктивності обробки даних.

Реалізовано програмний модуль CHUB для відтворення процесів ETL/ELT, модулі коннекторів, модулі інтеграції з середовищем та схеми трансформації даних і бібліотеки схем баз даних.

Проведено експерименти для валідації реалізованих методів в режимі навантажувального тестування з метою збору фактичних показників продуктивності системи Big Data. Результати демонструють значне зменшення використання ресурсів апаратної частини середовища та суттєвий приріст продуктивності системи на прикладі визначення зваженого агрегату для деяких випадків оптимізації. У випадку кешування система показала збільшення продуктивності у 2600 разів - від QPS 0.085 на неоптимізованому наборі даних до QPS 1028 за рахунок зменшення обсягу вибірки і фактичних даних.

Результати дослідження підтверджують ефективність запропонованого підходу та перспективність його впровадження для підвищення продуктивності системи Big Data. Подальші дослідження можуть бути спрямовані на вдосконалення методів попередньої обробки даних.

КЛЮЧОВІ СЛОВА: ПОПЕРЕДНЯ ПІДГОТОВКА ДАНИХ, BIG DATA, ОПТИМІЗАЦІЯ ЗАПИТІВ, ЗВАЖЕНА АГРЕГАЦІЯ, NOSQL, БАЗИ ДАНИХ, ПРОПУСКНА ЗДАТНІСТЬ, ЗАТРИМКА, УЗГОДЖЕНІСТЬ ДАНИХ.

ABSTRACT

Text part of the master's qualification work: 77 pages, 17 pictures, 9 tables, _20_ sources.

The purpose of the work is to optimize Big Data preprocessing algorithms for business process visualization tasks.

Object of research - is the data preprocessing processes in Big Data storage and processing systems.

Subject of research - is the algorithms, methods, and approaches to Big Data preprocessing applied to business process visualization.

Summary of the work: The work employs methods of analysis and comparison of Big Data preprocessing algorithms, including data transformation, normalization, and dimensionality reduction techniques. The development and software implementation of the weighted aggregate model were carried out using algorithmic and experimental methods. The system performance was evaluated through load testing, taking into account the impact of the hardware environment. To improve performance, query optimization, data partitioning, caching, and storage format substitution were applied. The analysis of results was conducted using throughput, latency, and data consistency metrics.

An analysis of modern data preprocessing methods in Big Data systems was performed, focusing on approaches to data transformation, normalization, and dimensionality reduction. Query optimization algorithms, data partitioning and caching methods, as well as approaches to selecting storage formats with respect to their impact on throughput, latency, and data consistency, were examined. Based on this analysis, the feasibility of using a weighted aggregate model to enhance data processing performance was substantiated.

A software module named CHUB was implemented to reproduce ETL/ELT processes, including connector modules, environment integration modules, data transformation schemes, and database schema libraries.

Experiments were conducted to validate the implemented methods under load testing conditions in order to collect actual performance metrics of the Big Data system. The results demonstrate a significant reduction in hardware resource utilization and a substantial increase in system performance, illustrated by the weighted aggregate computation in selected optimization cases. In the case of caching, the system achieved a 2600-fold performance improvement, increasing from 0.085 QPS on an unoptimized dataset to 1028 QPS due to a reduction in query scope and processed data volume.

The research results confirm the effectiveness of the proposed approach and its potential for improving the performance of Big Data systems. Further research may focus on enhancing data preprocessing methods.

KEYWORDS: DATA PREPROCESSING, BIG DATA, QUERY OPTIMIZATION, WEIGHTED AGGREGATION, NOSQL, DATABASES, THROUGHPUT, LATENCY, DATA CONSISTENCY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД - база даних; структурована сукупність даних, що зберігаються в електронній формі.

СУБД - система управління базами даних; спеціалізоване програмне забезпечення для створення, адміністрування та спільного використання баз даних.

ОЗУ - оперативний запам’ятовуючий пристрій (RAM, Random Access Memory)

ACID (Atomicity, Consistency, Isolation, Durability) - сукупність властивостей (атомарність, узгодженість, ізольованість, довговічність), що забезпечують надійність обробки транзакцій у реляційних СУБД.

BASE (Basically Available, Soft state, Eventual consistency) - модель проектування розподілених систем, яка віддає перевагу доступності та гнучкості стану даних перед негайною узгодженістю (принцип «узгодженості в кінцевому результаті»).

CAP (Consistency, Availability, Partition Tolerance) - теорема про неможливість одночасного забезпечення узгодженості, доступності та стійкості до розподілу в розподілених системах.

CPU (Central Processing Unit) - центральний процесор, основний апаратний компонент обчислювальної системи, що виконує інструкції програмного коду.

CRM (Customer Relationship Management) - система управління взаємовідносинами з клієнтами; прикладне програмне забезпечення для автоматизації стратегій взаємодії із замовниками.

ELT (Extract, Load, Transform) - архітектурний підхід до інтеграції даних, за якого трансформація виконується безпосередньо у цільовому сховищі після завантаження сирих даних.

ERP (Enterprise Resource Planning) - система планування ресурсів підприємства; комплексне програмне забезпечення для управління ключовими бізнес-процесами організації.

ETL (Extract, Transform, Load) - процес вилучення даних із зовнішніх джерел, їхнього перетворення на проміжному шарі та подальшого завантаження у цільове сховище.

IO (Input/Output) - операції введення та виведення даних, що визначають взаємодію процесора з периферійними пристроями або накопичувачами.

IoT (Internet of Things) - інтернет речей; мережа фізичних об'єктів, оснащених сенсорами та технологіями для обміну даними у реальному часі.

KPI (Key Performance Indicator) - ключовий показник ефективності; кількісний індикатор, що використовується для оцінки рівня досягнення стратегічних або оперативних цілей.

PCA (Principal Component Analysis) - метод головних компонент; алгоритм зниження розмірності даних шляхом виділення найбільш значущих характеристик.

RAM (Random Access Memory) - енергозалежна оперативна пам’ять із довільним доступом, що використовується для тимчасового зберігання даних під час обчислень.

SQL (Structured Query Language) - декларативна мова програмування, призначена для управління даними у реляційних СУБД та виконання аналітичних запитів.

t-SNE (t-distributed Stochastic Neighbor Embedding) - алгоритм нелінійного зниження розмірності, який використовується переважно для візуалізації високовимірних структур даних.

UMAP (Uniform Manifold Approximation and Projection) - сучасний метод зниження розмірності, що базується на теорії ріманової геометрії та забезпечує високу швидкість обробки великих масивів даних.

VSM (Value Stream Mapping) - карта потоку створення цінності; візуальний інструмент аналізу процесів для ідентифікації та усунення втрат.

ВСТУП

Актуальність теми дослідження зумовлена стрімким розвитком інформаційних технологій та зростання обсягів даних, що генеруються сучасними інформаційними системами, зумовили широке поширення концепції Big Data у різних галузях, зокрема в аналізі та візуалізації бізнес-процесів. Ефективне використання великих масивів даних дозволяє підвищити обґрунтованість управлінських рішень, оптимізувати внутрішні процеси організацій та забезпечити конкурентні переваги. Водночас значні обсяги, різнорідність і швидкість надходження даних суттєво ускладнюють їх обробку та подальшу аналітику.

Зростання об‘ємів даних у світі за даними компанії IDC

Рік	Кількість даних
2003	5 екзабайтів (1 ЕБ = 1 млрд ГБ)
2008	0,18 зетабайтів (1ЗБ = 1024 ЕБ)
2015	більше ніж 6,5 зетабайтів
2020	40-44 зетабайтів
2025 (прогноз)	400-440 зетабайтів

Одним із ключових етапів роботи з Big Data є попередня підготовка даних, яка включає очищення, трансформацію, нормалізацію та зменшення розмірності. Саме на цьому етапі формується якість вхідних даних для аналічних алгоритмів і систем візуалізації.

Окрім затримок з доступом до вхідних даних, недостатньо оптимізовані алгоритми попередньої обробки даних різної структури з різних джерел

призводять до зростання затримок опрацювання даних, зниження пропускної здатності системи та неефективного використання апаратних ресурсів, що є критичним для задач аналізу бізнес-процесів у реальному або наближеному до реального часі.

Особливої актуальності набуває оптимізація алгоритмів попередньої підготовки даних у контексті побудови агрегованих показників, які широко використовуються для візуалізації та аналізу бізнес-процесів. Застосування моделі зваженого агрегата дозволяє отримувати більш інформативні результати, однак водночас підвищує обчислювальне навантаження на систему Big Data. Це зумовлює необхідність дослідження та впровадження ефективних методів оптимізації, зокрема оптимізації запитів, розділення даних, кешування та вибору раціональних форматів зберігання.

Метою магістерської кваліфікаційної роботи є оптимізація алгоритмів попередньої підготовки Big Data в задачах візуалізації бізнес-процесів.

Для досягнення поставленої мети в роботі необхідно розв’язати такі основні завдання:

- проаналізувати предметну галузь та сучасні підходи до попередньої обробки даних у системах Big Data;
- дослідити алгоритми трансформації, нормалізації та зменшення розмірності даних;
- розробити та впровадити модель зваженого агрегата в системі Big Data, як найбільш жадібний алгоритм серед інших форм перетворення даних;
- реалізувати та оцінити методи оптимізації обробки даних, зокрема оптимізацію запитів, розділення даних, кешування та зміну форматів зберігання;
- провести експериментальні дослідження з використанням навантажувального тестування;

- виконати аналіз результатів за метриками пропускної здатності, затримки та узгодженості даних і надати практичні рекомендації.

Об’єктом дослідження є процеси попередньої підготовки даних у системах зберігання та обробки Big Data.

Предметом дослідження є алгоритми, методи та підходи попередньої підготовки Big Data, що застосовуються для візуалізації бізнес-процесів.

Наукова новизна полягає:

- у теоретичному обґрунтуванні та практичній реалізації комплексного підходу до попередньої обробки надвеликих масивів даних, які забезпечують підвищення продуктивності систем візуалізації бізнес-процесів у порівнянні з поширеними підходами до організації обробки та зберігання великих обсягів даних;
- на відміну від існуючих підходів, які фокусуються на горизонтальному масштабуванні сховищ, запропонований метод враховує взаємозв’язок між структурою даних та архітектурою обчислювального вузла. Це забезпечує вищу пропускну здатність системи без експоненціального зростання витрат на апаратне забезпечення.

Практичне значення роботи полягає у можливості використання отриманих результатів для підвищення продуктивності систем Big Data, що застосовуються для аналізу та візуалізації бізнес-процесів, а також у впровадженні розроблених методів у реальні інформаційні системи.

Апробація результатів магістерської роботи. Основні положення та результати дослідження доповідалися та обговорювалися на науково-практичних конференціях та семінарах з питань інформаційних технологій в освіті.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Концепція великих даних сформувалася наприкінці XX століття, коли організації почали створювати та накопичувати надзвичайно великі масиви інформації. На початковому етапі ці дані здебільшого мали структурований характер і зберігалися в реляційних базах даних. Однак із розвитком Інтернету, цифрових технологій та соціальних мереж обсяги неструктурованих даних - зокрема текстових, графічних, відео та сенсорних - почали стрімко зростати. У результаті традиційні підходи до обробки інформації виявилися недостатніми й потребували модернізації для ефективної роботи з такими потоками даних.

Великі мережеві дані (Big community records) - це масиви даних значного обсягу, що генеруються або накопичуються в різноманітних мережах, таких як Internet, комунікаційні, транспортні, цитаційні, біологічні мережі тощо. Ці набори даних мають настільки складну структуру, що традиційних методів обчислення виявляється недостатньо, і для отримання з них цінної інформації необхідні вдосконалені методи аналітики.

П’ять ключових характеристик великих даних (**концепція 5V**) - швидкість (velocity), обсяг (volume), цінність (value), достовірність (veracity) та різноманітність (variety) - повною мірою притаманні й великим мережевим даним [1, с. 42].

1.1 Інженерія Big Data

Життєвий цикл інженерії даних можна поділити на п’ять етапів [2, с. 5]. Кожен етап має притаманні йому процеси, технології та вимоги. У сукупності ці п’ять етапів утворюють повний життєвий цикл даних у сучасних системах інженерії даних і допомагають перетворювати необроблені дані на цінні бізнес-інсайти.

- **Генерація даних**. На цьому етапі дані створюються або надходять із різних джерел. Це можуть бути транзакції в бізнес-системах, дії користувачів у веб- або мобільних застосунках, дані з IoT-пристроїв, логи-подій [3, с. 128], сторонні API або зовнішні відкриті набори даних. Важливою характеристикою цього етапу є різноманіття форматів, швидкість появи даних та їхня якість.
- **Зберігання даних**. Згенеровані дані необхідно надійно зберігати. Для цього використовують різні типи сховищ: реляційні бази даних, data lake, data warehouse, об’єктні сховища в хмарі тощо. На цьому етапі приймаються рішення щодо структури зберігання, масштабованості, вартості, безпеки та доступності даних.
- **Поглинання даних** - це процес перенесення даних із джерел у сховище. Вона може відбуватися в пакетному (batch) режимі або в режимі реального часу (streaming). На цьому етапі важливо забезпечити надійність доставки даних, обробку помилок, контроль дублікатів та мінімальні затримки.
- **Трансформація даних**. Після завантаження дані часто потребують очищення, нормалізації, агрегації та збагачення. На цьому етапі «сирі» дані перетворюються на аналітично корисні набори, зручні для подальшого використання. Тут застосовуються бізнес-правила, логіка обчислень та підготовка даних для конкретних сценаріїв - аналітики, звітності або машинного навчання.
- **Надання даних**. Останній етап - це надання підготовлених даних кінцевим споживачам. Ними можуть бути BI-інструменти, дашборди, аналітики, data scientists або прикладні сервіси. На цьому етапі важливі продуктивність запитів, актуальність даних, контроль доступу та зручність використання.

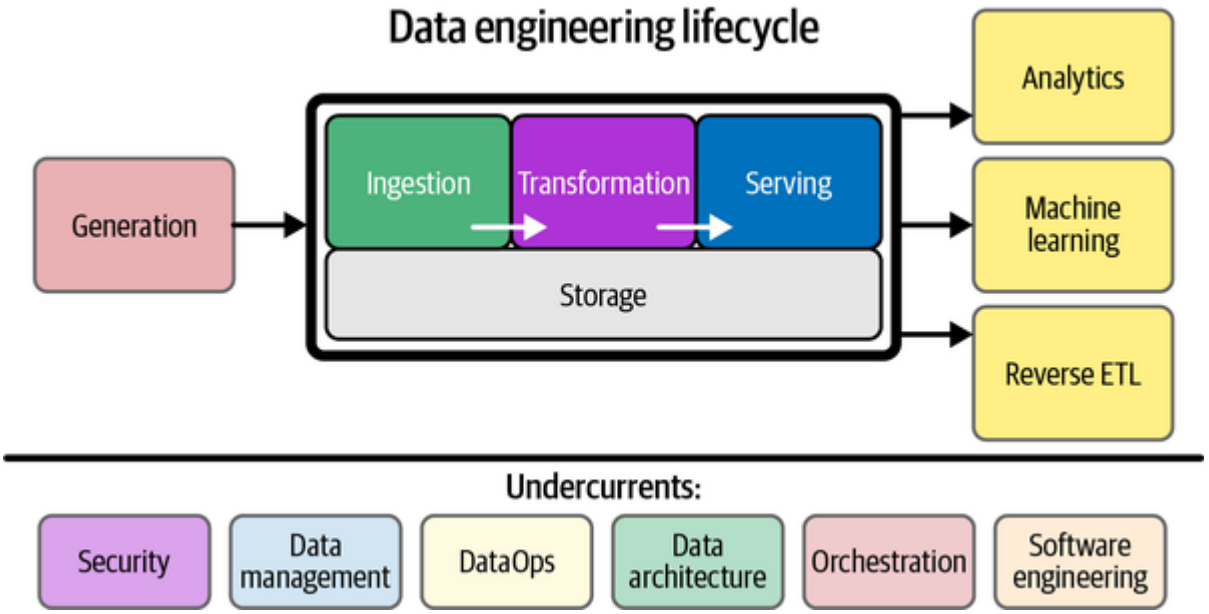


Рис 1.1 Життєвий цикл інженерії даних

Дані можуть бути обмеженими або необмеженими. Обмежені дані характеризуються наявністю визначеного кінця, тоді як необмежені дані генеруються безперервно та потенційно є нескінченними. Прикладом обмежених даних можуть слугувати показники продажів товарів за попередній рік. Натомість необмежені дані це, наприклад, інформація з дорожніх сенсорів, які в реальному часі фіксують кількість транспортних засобів і їхню швидкість руху на автомагістралях [2, с 236].

Розрізнення обмежених і необмежених даних має принципове значення під час проєктування конвеєрів обробки даних. У випадку обмежених даних дослідник або інженер має змогу працювати з повним набором інформації, аналізувати його цілісно, виконувати запити, розміщувати дані в проміжному середовищі обробки та застосовувати інструменти валідації для визначення допустимих діапазонів значень, характеристик розподілу або інших показників якості даних.

ETL-процес є одним із ключових етапів управління даними, що забезпечує інтеграцію інформації з різнорідних джерел, її очищення, трансформацію та подальше завантаження до сховища даних. На початковій

стадії структуровані і неструктуровані дані, отримані з різних джерел, експортуються до зони обробки [4]. У межах цього етапу здійснюється виправлення недоліків, які могли виникнути під час формування або введення даних, зокрема помилок форматування, орфографічних неточностей і відсутніх значень.

Після завершення етапу очищення виконується трансформація даних, що передбачає приведення інформації до уніфікованої логічної та фізичної схеми, розробленої відповідно до вимог сховища даних. Оброблені та повністю підготовлені дані переносяться із зони обробки до довготривалого сховища, де стають доступними для подальшого використання в аналітичних задачах та процесах підтримки прийняття рішень.

Аналогічні за змістом процеси характерні і для підходу ELT, однак його принципова відмінність полягає в тому, що дані спочатку завантажуються безпосередньо до сховища даних, а етап їх трансформації виконується вже після завантаження з використанням обчислювальних ресурсів сховища.

1.2 Моделі агрегування та узагальнення інформації

Моделі агрегування та узагальнення інформації - це підходи до зведення детальних даних у компактні, зрозуміліші форми (узагальнення) та об'єднання схожих даних (агрегування) для аналізу та прийняття рішень. Вони є ключовими у створенні інформаційних систем для аналітики, дозволяючи виявляти закономірності та зменшувати складність представлення даних.

Таблиця 1.1

Порівняння агрегування та узагальнення

	Агрегування	Узагальнення
Мета	Стиснення даних	Виявлення закономірностей
Рівень	Дані	Модель
Інтерпретованість	Висока	Від середньої до низької

Узагальнення - це процес побудови абстрактної моделі, що описує загальні закономірності даних, формує концептуальну модель предметної області для стратегічного прогнозування. Типи узагальнення:

Моделі зменшення розмірності (зокрема Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), Uniform Manifold Approximation and Projection (UMAP) та автоенкодери) призначені для перетворення високовимірних даних у простір меншої розмірності зі збереженням найбільш інформативних характеристик. Такі методи широко застосовуються для візуалізації складних багатовимірних структур, зменшення рівня шуму в даних та підвищення обчислювальної ефективності подальших моделей машинного навчання [5, с. 65-105][6, с. 79–95].

Кластеризація - це метод машинного навчання для групування схожих об’єктів у гомогенні кластери на основі метрик близькості. Алгоритми (k-means, DBSCAN, ієрархічні методи) виявляють приховані структури в даних без попередньої розмітки, забезпечуючи компактне узагальнення інформації [7, с. 205-233].

Моделі правил і шаблонів виявляють стійкі логічні зв’язки та повторювані сценарії в масивах даних. Алгоритми асоціативного пошуку (Apriori, FP-Growth) ідентифікують приховані залежності між подіями

(наприклад, супутні товари у кошику), а дерева рішень (Decision Trees) формують ієрархію правил для класифікації та прогнозування [8, с. 146-200].

Регресійні та ймовірнісні моделі встановлюють математичні залежності між змінними для прогнозування значень або оцінки ймовірності подій. Лінійна та логістична регресії (включно з GLM) моделюють вплив факторів на цільовий показник, а Байєсівські методи адаптують прогнози на основі нових даних [9, с. 219-248].

Інтелектуальне та семантичне узагальнення базується на використанні нейромережевих архітектур для перетворення даних у векторні представлення (embeddings), що відображають їхній контекстуальний зміст. Моделі забезпечують глибоку абстракцію через багаторівневу обробку, трансформуючи неструктуровані масиви у змістовні знаннєві структури для когнітивного аналізу та автоматичного реферування [10, с. 69-96].

Агрегування - це процес перетворення атомарних даних у компактні показники, що класифікується за напрямами: статистичне (обчислення середніх, медіан, дисперсії), категоріальне (групування за ознаками), ієрархічне (OLAP-операції консолідації та деталізації) та вагове (врахування значущості елементів), семантичне агрегування і використання віконних функцій [11, с. 404-423].

Дескриптивно-статистичне агрегування з визначенням середнього, медіани, мінімумів і максимумів, дисперсії, стандартного відхилення, квантилі, тощо, застосовується для аналізу часових рядів, у бізнес-аналітиці для виведення KPI, інших метрик [12, с. 25-60].

Категоріальне групування базується на розподілі множини даних на непересічні підмножини за дискретними ознаками. Модель реалізується через оператори SQL (GROUP BY, CUBE, ROLLUP) та подібні функції в Spark [13, с. 190-192], що дозволяють обчислювати агрегатні показники для кожного сегмента та його ієрархічних підсумків.

Вагове агрегування застосовується у випадках, коли вхідні елементи мають неоднаковий вплив на кінцевий обчислюваний показник. Суть методу полягає у присвоєнні індивідуального вагового коефіцієнта кожному елементу даних перед їхнім об'єднанням [14, с. 120-134].

Ієрархічне та багатовимірне агрегування дозволяє досліджувати дані на різних рівнях абстракції, основною сферою застосування є побудова інтерактивних дашбордів, яке об’єднує гетерогенну інформацію за змістом, перетворюючи сирі масиви на структуровані метрики для аналізу(прим. HETree (Hierarchical Exploration Tree) [15, с. 3-20]).

Слід також враховувати часовий аспект під час роботи з потоковими даними. Для обмежених наборів даних можна одноразово обчислити статистичні характеристики, такі як середнє або максимальне значення певного атрибута. У випадку необмежених даних ці показники необхідно постійно обчислювати в міру надходження нових елементів у межах конвеєра обробки даних.

Для необмежених даних характерним є потоковий режим надходження, за якого наперед невідомо, якими будуть наступні елементи даних. Водночас це не виключає можливості їх валідації, оскільки для таких даних зазвичай існують відомі обмеження.

Це підводить до ключового механізму перетворення необмежених потоків у керовані обмежені набори що робить можливим їх аналітичне опрацювання, контроль якості та використання в системах підтримки прийняття рішень.

Часове (віконне) агрегування перетворює безперервні потоки даних на дискретні набори шляхом аналізу в межах визначених часових інтервалів. *Фіксовані вікна* розділяють потік на відрізки однакової тривалості, що не перекриваються (наприклад, щогодинні звіти). *Ковзні вікна* дозволяють інтервалам перекриватися, зміщуючись із заданим кроком для аналізу динамічних показників, тоді як *сесійні вікна* групують дані на основі періодів активності користувача, розділених паузами [16, с. 145–178].

1.3 Методи візуалізації бізнес-процесів

Бізнес-процеси генерують багатовимірні потоки даних (фінансові, часові, логістичні) з гетерогенних джерел, таких як ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) та IoT (Internet of Things). Складність інтеграції цих розрізнених масивів робить попередню обробку критичним етапом підготовки інформації. Якість візуалізації та глибина аналітичних висновків безпосередньо залежать від ефективності агрегування та очищення даних на стадії їх підготовки до візуального представлення [17, с. 45-72].

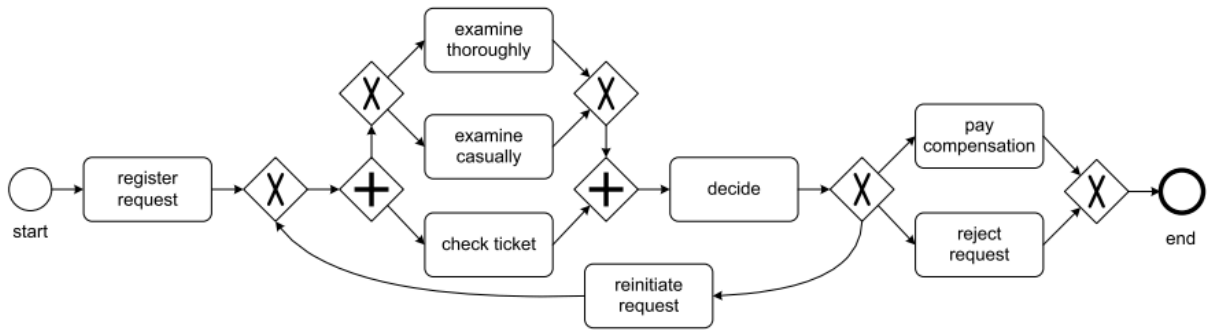


Рис 1.2 Процес, змодельований в термінах BPMN

Методи візуалізації бізнес-процесів спрямовані на перетворення складних багатовимірних даних у наочні моделі для операційного та стратегічного управління:

Діаграми потоків та нотації (BPMN, EPC): Графічне відображення послідовності кроків, подій та логічних розгалужень процесу для аналізу його структури. BPMN (Business Process Model and Notation) - система умовних позначень (нотація) для моделювання бізнес-процесів (Рис 1.2). EPC (Event-driven Process Chain) - тип блок-схеми, що використовується для моделювання бізнес-процесів, орієнтований на події

Інтерактивні панелі керування (BI-панелі): Використання віджетів (графіків, гістограм, теплокарт) для моніторингу KPI у реальному часі на основі агрегованих даних з ERP та CRM на панелі керування (Рисунок 1.3).

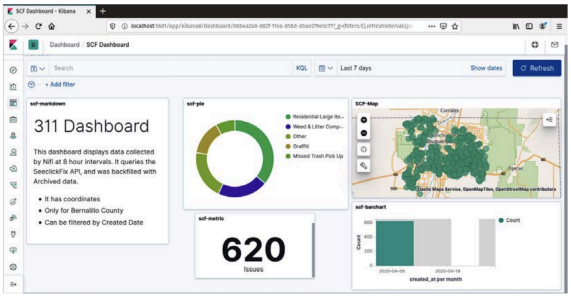


Рис. 1.3 Панель керування з фільтрацією у системі Kibana.

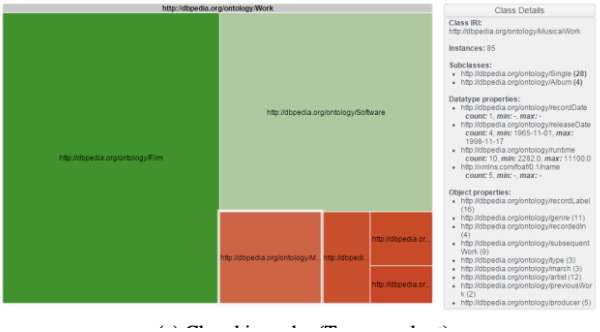


Рис. 1.4 Ієрархія класів (Treemap Chart)

Праця Бланша та Леколіне, присвячена масштабованим деревоподібним картам (Zoomable Treemaps), демонструє інтеграцію різноманітних технік взаємодії, де деревоподібна структура виступає багатомасштабним навігаційним простором (Рисунок 1.4). Це дослідження є показовим з погляду модернізації традиційних методів візуалізації шляхом впровадження інструментів агрегації даних та багаторівневої навігації в ієрархічних структурах [18, с. 1248-1252][19].

Карти потоку створення цінності (VSM): візуалізація матеріальних та інформаційних потоків для ідентифікації втрат і оптимізації часових витрат.

Графи зв'язків: автоматична побудова фактичних моделей процесів на основі лог-файлів систем для виявлення реальних відхилень від регламентів.

3D-візуалізація та цифрові двійники: Створення об'ємних моделей логістичних або виробничих процесів для імітаційного моделювання сценаріїв «що, якщо».

Перетворення багатовимірних даних у зрозумілі аналітичні моделі є основним результатом використання методів візуалізації бізнес-процесів. Вони допомагають виявити приховані закономірності та оптимізувати операційну діяльність. Ефективність цих методів критично залежить від якості попередньої

обробки та агрегування даних, що дозволяє уникнути надмірної деталізації та зосередитись на ключових показниках.

Вибір конкретного інструменту візуалізації залежить від поставленої аналітичної задачі та типу даних. Наприклад, для оперативного моніторингу використовуються одні підходи, тоді як для стратегічного планування та імітаційного моделювання - інші [7, с. 29-32]. Сучасні тенденції вказують на інтеграцію різних методів для забезпечення комплексного розуміння бізнес-середовища.

Інтерактивні панелі керування можна створювати в різних аналітичних платформах і BI-інструментах, таких як Tableau, Power BI, Google Data Studio, Grafana, Qlik Sense, Looker та інші. Вони дозволяють інтегрувати дані з різних джерел - баз даних, CSV-файлів, API або хмарних сервісів - і перетворювати їх на інтерактивні графіки, таблиці та карти. Такі дашборди допомагають відстежувати ключові показники, аналізувати тенденції, приймати рішення на основі даних і забезпечують візуальне представлення складної інформації для команд і керівництва.

Наприклад, створення панелей керування в Kibana [2, с. 139-151] дозволяє візуалізувати дані з Elasticsearch у зрозумілих графіках, таблицях та картах. Інтерактивний інтерфейс Kibana дозволяє користувачам фільтрувати дані, налаштовувати візуалізації та швидко отримувати аналітичні висновки.

Якість візуалізації безпосередньо впливає на швидкість та точність прийняття управлінських рішень. Правильно підібраний та реалізований метод візуалізації дозволяє не лише констатувати поточний стан справ, а й прогнозувати майбутні тренди, виявляти вузькі місця та мінімізувати потенційні ризики у складних бізнес-системах.

2 АНАЛІЗ АЛГОРИТМІВ ПОПЕРЕДНЬОЇ ОБРОБКИ ДАНИХ ТА ВВЕДЕННЯ МОДЕЛІ ЗВАЖЕНОГО АГРЕГАТА

У цьому розділі розглядаються методи попередньої підготовки даних, що базуються на адаптації алгоритмів трансформації даних, таких як нормалізація, згладжування, усереднення, агрегації, інтегрованих з механізмом тимчасового зберігання. Метою такого підходу є підвищення продуктивності системи шляхом поєднання двох різносторонніх процесів: трансформації та накопичення даних. Це дозволяє ефективніше обробляти великі обсяги інформації та теоретично покращує продуктивність системи, забезпечуючи більш точну та своєчасну аналітику.

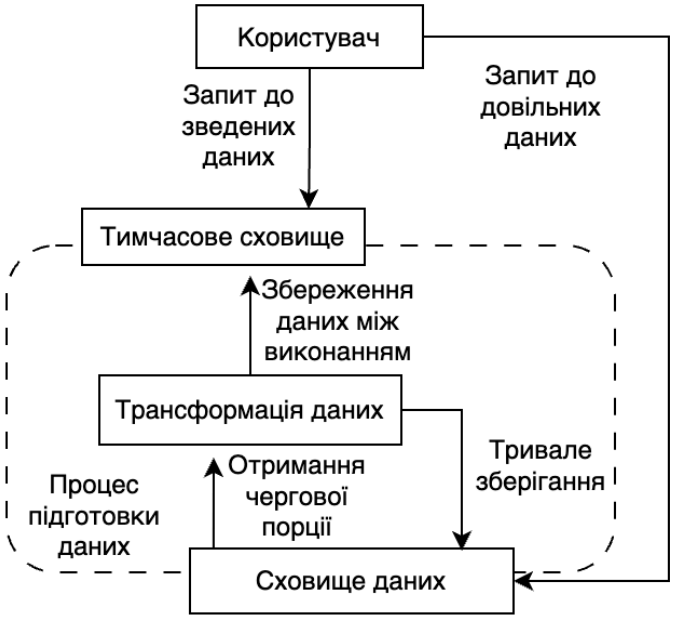


Рис. 2.1 Процеси трансформації і накопичення даних

Архітектура трансформації даних (Рисунок 2.1) базується на взаємодії модуля обробки зі складним ієрархічним середовищем збереження. Запит користувача спрямовується до консолідованих даних, формування яких забезпечує модуль трансформації. Логіка керування потоками передбачає звернення до архівного сегмента у випадках, коли шукані дані відсутні у

тимчасовому буфері. Ключову роль відіграє підсистема довготривалого зберігання, яка гарантує цілісність даних після завершення сеансу роботи застосунку.

При цьому реалізовано диференційований підхід: частина обробленої інформації зберігається у тимчасовому сховищі для оперативного доступу, тоді як інша - спрямовується до глобального репозиторію для накопичення та тривалого зберігання. Таким чином, тимчасове сховище забезпечує швидкий доступ до даних під час виконання програми, а довготривале сховище виступає ресурсом для накопичення та підтримки історичних даних.

2.1 Призначення та основні етапи попередньої обробки даних

Призначення попередньої обробки даних необхідне для забезпечення якості, надійності та придатності вихідного набору даних для подальшого аналізу або візуалізації чи безпосереднього використання в аналітичних чи прогнозних моделях та як вхідні дані для систем прийняття управлінських рішень. Необроблені дані можуть містити шум, пропущені значення, аномалії та невідповідні формати, що можуть негативно вплинути на точність і ефективність аналітичних зрізів.

Виявлення та виправлення помилок, суперечностей та неточностей в даних відбуваються на етапі **очищення даних**.

Вирішення проблем узгодженості схем, ідентифікації дублюючих сутностей між джерелами та забезпечення єдиного формату для даних із різних джерел, об'єднанням їх до єдиного, узгодженого набору даних відбувається на етапі **інтеграції даних**.

На етапі **трансформації даних**, дані приводяться до формату найбільш придатного для конкретного методу аналізу або моделювання, виконуючи вимоги нормалізації чи масштабування діапазону числових значень або

стандартизації із середнім значенням та стандартним відхиленням, або виконуючи агрегацію даних як операцію узагальнення показників за вимірами.

Виконуючи **зменшення розмірності даних** досягається зменшення обсягу даних без втрати критично важливої інформації для підвищення ефективності обчислень та зменшення шуму за допомогою вибірки ознак шляхом вибору найбільш релевантних атрибутів, видобуванням ознак - через створення нових менш об’ємних ознак із існуючих за допомогою таких методів як Метод головних компонент (PCA) [20, с. 243-249].

Використання **дискретизації та генерація ознак** полягає у перетворенні безперервних числових даних на дискретні категорії або створення абсолютно нових, більш інформативних ознак із існуючих [21, с. 65-112].

2.1.1 Аналіз методів трансформації і нормалізації даних

В результаті обробки, дані можуть бути заповнені середніми/медіанами/модою або видалені при наявності значної кількості пропусків. Також серед дій над даними є виявлення та обробка викидів у даних, які можуть бути пов’язані з помилками вимірювання або рідкісними але реальними подіями для прийняття рішення стосовно трансформації, корекції чи видалення (приміром, ідентифікація повторюваних записів, які можуть спотворювати результати аналізу).

В свою чергу, для коригування показників повторюваних подій використовуються методи згладжування шумів і випадкових коливань: зважене ковзне середнє, експоненціальне згладжування, поліноміальна регресія, кубічні сплайни, та ін. Також виконуються дії зі зменшення кількості записів (sampling) шляхом використання репрезентативної підмножини даних замість усього масиву.

Трансформація даних охоплює різноманітні методи, що змінюють дані, не змінюючи їхньої фундаментальної природи, для відповідності вимогам аналізу. Найчастіше використовуються наступні методи:

- **Кодування категоріальних змінних:** перетворення текстових категорій на числові значення;
- **Агрегація даних:** об'єднання кількох фрагментів даних у зведену форму
- **Генерація ознак**: створення нових атрибутів на основі існуючих даних для покращення продуктивності моделі
- **Згладжування та дискретизація:** перетворення безперервних числових даних на дискретні категорії або діапазони для спрощення аналізу
- **Логарифмічна трансформація:** застосування математичних функцій до даних для зменшення перекосу від викидів та наближення до нормального розподілу.

Нормалізація є важливою підмножиною трансформації, особливо в статистичному аналізі та машинному навчанні, де вона забезпечує рівномірний внесок усіх ознак у модель, запобігаючи домінуванню ознак із великими діапазонами.

- **Мін-Макс масштабування**: масштабує значення ознаки до певного діапазону. Це корисно для алгоритмів, чутливих до масштабу, до недоліків якого слід віднести складність обробки викидів;
- **Z-оцінка масштабування**: перетворює дані так, щоб вони мали середнє значення 0 і стандартне відхилення 1. Цей метод ефективніший для обробки викидів і корисний для алгоритмів, які передбачають нормальний розподіл (наприклад, лінійна регресія, градієнтний спуск).

- **L1 та L2 нормалізація**: використовуються для масштабування вектора ознак так, щоб його L1 або L2 норма (сума абсолютних або квадратних значень) дорівнювала 1. Це часто застосовується в нейронних мережах та алгоритмах кластеризації.

2.1.2 Огляд методів зменшення розмірності

Методи зменшення розмірності - це сукупність технік, спрямованих на скорочення кількості випадкових змінних (ознак) шляхом отримання набору головних (незалежних) змінних, зберігаючи при цьому якомога більше корисної інформації з вихідного набору даних. Ці методи допомагають зменшити обчислювальне навантаження, усунути надлишковості та покращити візуалізацію даних.

Класичним, лінійним методом зменшення розмірності є Метод головних компонент через створення невеликої кількості головних компонент, які пояснюють максимальну дисперсію (варіативність) вихідного набору даних. Це дозволяє узагальнити інформацію, що міститься в десятках чи сотнях змінних, до 2-5 ключових компонентів.

Загалом методи можна поділити на дві категорії: відбір ознак та видобування/проекція ознак.

Методи відбору ознак обирають підмножину оригінальних ознак, видаляючи нерелевантні або надлишкові не створюючи нових змінних.

Методи-фільтри оцінюють релевантність ознак, використовуючи статистичні показники незалежно від моделі машинного навчання, до яких належать: *дисперсійний аналіз*, що видаляє ознаки з низькою дисперсією; *хі-квадрат*, який використовується для категорійних даних для перевірки залежності між ознакою та цільовою змінною; *кореляційний аналіз* для видалення ознак, які сильно корелюють між собою або слабко корелюють із цільовою змінною.

Методи-обгортки використовують конкретний алгоритм машинного навчання для оцінки ефективності різних підмножин ознак. Вони є обчислювально дорожчими, але часто дають кращі результати. До таких методів слід віднести *покроковий відбір* як послідовне додавання або видалення ознак для знаходження оптимальної комбінації; *рекурсивне виключення ознак* багаторазове тренування моделі та видалення найменш важливих ознак на кожній ітерації.

2.1.3 Вибір комплексу алгоритмів попередньої обробки даних

Оптимізація попередньої обробки Big Data спрямована на зменшення обчислювальної складності та витрат пам'яті і процесорного часу. Серед типових задач (фільтрація, нормалізація, очищення) знаходження показників агрегації за вимірами об’єктивно є найскладнішою з ряду причин.

Обчислювальна складність. На відміну від простої агрегації (наприклад, SUM або COUNT), де кожен елемент обробляється за одну ітерацію, **зважений агрегат** вимагає зберігання та зіставлення векторів ваг для кожного виміру. Виконання додаткових операцій множення перед підсумовуванням, що подвоює кількість операцій з плаваючою комою та, в загальному випадку, може потребувати неодноразового звернення до визначених агрегатів з попередніх ітерацій.

Зважені агрегати часто потребують обчислення знаменника (суми ваг) паралельно з чисельником. Це потребує додаткового використання пам’яті, оскільки система має підтримувати два накопичувальних буфери замість одного для кожної групи.

У розподілених системах, агрегації зваженого показника, що потребують групування даних за ключами, значення, що обчислюються, супроводжуються відповідними вагами. Якщо ваги динамічні або залежать від контексту всієї вибірки, це унеможливлює використання локальних комбайнерів (в MapReduce

моделі), що призводить до критичного перевантаження мережі. У надвеликих наборах даних виміри часто є розрідженими. При **зваженій агрегації** необхідно обробляти перетини множин "дані-ваги". Якщо структура ваг не збігається зі структурою даних, обчислювальна складність зростає експоненціально через необхідність виконання додаткових аналітичних операцій перед агрегацією.

Модель зваженого агрегата є потужним інструментом для агрегації великої кількості окремих записів або показників в одну комплексну метрику. Замість того, щоб аналізувати кожен окремий параметр ізольовано, модель дозволяє консолідувати їх, використовуючи ваги, які відображають відносну важливість кожного критерію. У задачах, що включають численні джерела даних або критерії оцінки, агрегація допомагає структурувати цю складність і перетворити її на єдиний, зрозумілий показник.

Гнучкість моделі зваженого агрегування дає змогу застосовувати **ітераційний підхід** щодо визначення оптимальних значень вагових коефіцієнтів, що забезпечує адаптивність та підвищує точність моделі. Початкові ваги можуть бути визначені на основі експертних оцінок або попереднього аналізу. В рамках ітераційного процесу ці ваги можуть переглядатися та коригуватися. У міру надходження нових даних або зміни умов завдання модель може бути уточнена. Застосування ітераційних процедур дає змогу мінімізувати помилку шляхом застосування методів машинного навчання або оптимізаційних алгоритмів, що забезпечує стійкість та обґрунтованість фінальної моделі.

Завдяки агрегації записів, яка забезпечує комплексність оцінки, та можливості використання ітераційного підходу для уточнення параметрів, модель зваженого агрегата є високоефективним та гнучким рішенням для поставленої задачі. Вона дозволяє створити прозору, вимірювану та об'єктивну систему оцінювання.

Поки звичайна агрегація демонструє лінійну масштабованість, **зважена агрегація** стає обмежувальним фактором через високу вартість серіалізації та

відсутність можливості повністю локалізувати обчислення на окремих вузлах кластера.

2.2 Теоретичні основи моделі зваженого агрегата

Теоретичною основою моделі зваженого агрегату є принцип лінійної адитивної моделі, яка виходить з припущення, що загальна оцінка об'єкта є сумою значень окремих його компонентів, помножених на відповідні коефіцієнти важливості (вагові коефіцієнти).

Математично це представлено загальною формулою

$$\sum_{i=0}^{n} f(x_i, w_i), \tag{2.1}$$

де x_i - значення i-го критерію;

w_i - вага критерію;

n - кількість показників для розрахунку.

Одним із ключових етапів у застосуванні даної моделі є визначення або експертне призначення ваг, що має забезпечувати об'єктивне відображення важливості або впливу кожного критерію на кінцевий результат. Цей процес є визначальним для успішного застосування моделі, оскільки саме ваги відіграють критичну роль у формуванні точних і адекватних висновків.

Модель зваженої агрегації є фундаментальним інструментом для структурованого та раціонального аналізу інформації. Вона забезпечує ефективне узагальнення даних, перетворюючи сирі, часто неоднорідні дані на обґрунтовані та інтерпретовані показники.

Завдяки використанню вагових коефіцієнтів модель дозволяє зменшувати вплив менш значущих параметрів та і виділяти ключові фактори, що критично

важливо для прийняття обґрунтованих рішень у таких сферах, як бізнес, наукові дослідження та державне управління. Таким чином, модель забезпечує підвищення точності аналізу та підтримує високий рівень гнучкості й адаптивності до завдань дослідження або прийняття рішень.

Якщо у досліджуваній множині показники для різних вимірів не є однорідними, без нормалізації вони матимуть непропорційно великий вплив на математичну модель через відмінності в абсолютних значеннях. Нормалізація усуває ці відмінності в масштабах:

$$\widehat{x} = \frac{x}{\sum_{i=0}^{n} x_i}, \tag{2.2}$$

де x - показник, який підлягає нормалізації;

n - кількість елементів.

В результаті чого скалярне значення s зваженого показника набуває наступної форми:

$$s = f(x, w) = \widehat{x} * w, \tag{2.3}$$

де $\widehat{x}$ - нормалізований показник;

w - ваговий коефіцієнт.

Тоді зважений максимум визначається за формулою:

$$WMax(x, w) = \max_{i}(s_i), \tag{2.4}$$

де s_i - скалярне значення i-го елементу.

2.2.1 Методи визначення вагових коефіцієнтів

Визначення вагових коефіцієнтів w_i (формула 2.1) є критично важливим етапом при побудові моделі зваженого агрегату, оскільки саме ваги визначають відносну важливість кожного показника. Існує кілька основних груп методів їх визначення: **експертні та суб'єктивні методи, об'єктивні методи, комбіновані методи**.

Експертні та суб'єктивні методи спираються на думку фахівців, експертів або осіб, які приймають рішення, і є найбільш поширеними, особливо коли об'єктивних історичних даних недостатньо. Представлені **методом прямого присвоєння**, де кожному критерію присвоюється числове значення, сума балів зазвичай нормалізується до 1 або 100% і вважається найпростішим, проте потенційно найменш точним методом.

В **методі попарних порівнянь** експерти послідовно порівнюють кожну пару критеріїв між собою за шкалою важливості (наприклад, "Критерій А втричі важливіший за Критерій Б"). На основі матриці порівнянь розраховуються власні вектори, які формують остаточний набір вагових коефіцієнтів. Метод забезпечує високу узгодженість суджень і є дуже надійним.

Ще одним методом є метод ранжування, де експерти просто впорядковують критерії від найбільш до найменш важливого. Ваги згодом можуть бути розраховані на основі присвоєного рангу за певними формулами (наприклад, Rank Sum або Rank Reciprocal methods).

Об'єктивні (або дата-орієнтовані) методи визначення ваг використовують математичний та статистичний аналіз для отримання ваг без впливу людської суб'єктивності. Вони ідеально підходять для ситуацій, де набір даних містить чітку цільову змінну, що дозволяє алгоритмам "вчитися" на фактичних результатах.

Приклади таких методів включають:

Регресійний аналіз: Використовує статистичні моделі для визначення того, як різні вхідні ознаки впливають на цільову зміну, призначаючи відповідні ваги, що мінімізують помилку прогнозування.

Метод аналізу головних компонент (PCA): Хоча він в основному використовується для зменшення розмірності, він визначає ваги (навантаження) для створення нових, некорельованих компонентів на основі дисперсії даних.

Ентропійний метод: Визначає ваги на основі міри невизначеності або дисперсії даних: чим більше дисперсія даних (ентропія), тим більша вага присвоюється відповідному критерію.

3 РОЗРОБКА, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ BIG DATA ДЛЯ МОДЕЛІ ЗВАЖЕНОГО АГРЕГАТА

3.1 Постановка експерименту

Метою даного розділу є комплексне дослідження та визначення оптимального алгоритму попередньої підготовки даних, що засноване на прикладі обчислення зваженого агрегату. Оскільки попередня обробка часто є найбільш ресурсомістким етапом (Data Bottleneck), особлива увага приділяється оцінці апаратного забезпечення та його здатності ефективно масштабувати обчислення.

У межах розділу аналізується критичний вплив архітектури процесора, кількості обчислювальних ядер та механізмів паралелізму на швидкість виконання користувацьких запитів. Отримані результати дозволять обґрунтувати вибір апаратної та програмної конфігурації для подальшого високонавантаженого тестування та глибинного аналізу великих масивів даних.

Розглянемо інформаційну систему з наступними характеристиками, які визначають її архітектуру та логічну структуру, функціональні можливості, принципи збору, обробки, зберігання й передачі даних. Система забезпечує накопичення, актуалізацію та аналіз значних обсягів інформації, при цьому враховуються такі параметри, як середній розмір одного рядка даних, загальна кількість записів у базі даних, швидкість доступу до інформації та вимоги до її цілісності й безпеки.

Основні кількісні та якісні характеристики інформаційної системи наведено в таблиці 3.1. База даних системи складається зі 100 взаємопов’язаних таблиць, що відображають предметну область та забезпечують нормалізоване зберігання інформації. Загальний обсяг даних становить близько 1 ТБ, що

свідчить про необхідність використання ефективних механізмів зберігання та оптимізації доступу до даних.

Середній розмір одного рядка даних складає приблизно 1 КБ, а загальна кількість записів у базі даних сягає близько 1 мільярда рядків. Такі показники зумовлюють підвищені вимоги до продуктивності системи управління базами даних, індексації, паралельної обробки запитів та використання кешування.

Інформаційна система підтримує близько 50 функціональних запитів, основними з яких є агрегатні запити, операції фільтрації та групування даних. Зазначені типи запитів орієнтовані на аналітичну обробку інформації та формування зведених показників, що використовується для прийняття управлінських рішень.

З огляду на обсяг і характер даних, у системі передбачено забезпечення цілісності, конфіденційності та доступності інформації, а також можливість масштабування у разі зростання кількості користувачів і даних. Взаємодія з користувачами та іншими інформаційними системами здійснюється за допомогою стандартизованих інтерфейсів і протоколів обміну даними, що забезпечує інтеграцію системи в єдине інформаційне середовище.

Таблиця 3.1

Характеристики інформаційної системи

Параметр	**Значення**
Кількість таблиць	100
Загальний обсяг даних	1ТБ
Кількість функціональних запитів	50
Середній розмір рядка	1КБ
Кількість рядків	≈ 1 млрд
Тип запитів	агрегати, фільтрація, групування

3.1.1 Апаратне середовище та його вплив

Основна мета дослідження полягає в оцінці алгоритмічної ефективності методів і засобів обробки даних, однак апаратне середовище, у якому здійснюється виконання програмної реалізації системи, має суттєвий вплив на її фактичну продуктивність. Обчислювальні ресурси, характеристики підсистеми пам’яті та засобів зберігання даних, а також параметри мережевої взаємодії можуть як обмежувати, так і підсилювати ефективність обраних алгоритмів.

Апаратне середовище є одним із визначальних чинників, які необхідно враховувати під час проведення розрахунків і моделювання системи обробки даних, оскільки саме воно визначає можливості паралельної обробки, швидкість доступу до даних і затримки під час виконання операцій введення-виведення. Ігнорування апаратних характеристик може призвести до отримання результатів, що не відповідають реальним умовам експлуатації системи.

У межах дослідження доцільно враховувати такі ключові аспекти апаратного середовища: обчислювальну потужність процесорів, кількість і тип ядер, обсяг та швидкодію оперативної пам’яті, характеристики кеш-пам’яті, продуктивність підсистеми зберігання даних, а також пропускну здатність і затримки мережевих з’єднань. Сукупний вплив зазначених параметрів безпосередньо визначає час виконання алгоритмів, масштабованість системи та її здатність обробляти великі обсяги даних у задані часові межі.

Нижче розглянуто основні аспекти апаратного середовища, які мають бути враховані під час аналізу та подальшого моделювання системи обробки даних з метою отримання об’єктивної оцінки її продуктивності та ефективності.

Апаратне середовище інформаційної системи зазвичай представлене сукупністю обчислювальних і комунікаційних ресурсів, що безпосередньо впливають на ефективність виконання алгоритмів обробки даних. Ключовими

компонентами такого середовища є центральний процесор і кількість його ядер, що визначають можливість паралельного виконання окремих частин програми та рівень підтримки багатопотокової обробки. Наявність кількох обчислювальних ядер дозволяє ефективно розподіляти навантаження між потоками виконання та зменшувати загальний час обробки великих обсягів даних.

Важливу роль відіграє обсяг оперативної пам’яті, виділеної для виконання програмної реалізації алгоритмів обробки даних. Достатній обсяг оперативної пам’яті забезпечує можливість зберігання проміжних результатів, кешування часто використовуваних даних та зменшення кількості звернень до зовнішніх носіїв, що позитивно впливає на загальну продуктивність системи.

Для тривалого зберігання даних між етапами їх перетворення використовується підсистема зберігання, що характеризується обсягом і технологією жорсткого диска або твердотільного накопичувача. Тип носія даних, його пропускна здатність та затримки доступу визначають швидкість операцій введення-виведення, що є критичним чинником під час роботи з великими масивами даних і виконання агрегатних або аналітичних запитів.

Окремим елементом апаратного середовища є мережевий інтерфейс, який забезпечує взаємодію системи з зовнішніми постачальниками даних та/або розподіленими системами зберігання. Його пропускна здатність і рівень затримок безпосередньо впливають на час отримання та передачі даних, особливо у випадках інтеграції з віддаленими сервісами або хмарними платформами.

Алгоритм обробки даних, за наявності відповідних апаратних ресурсів, повинен бути адаптований до паралельної обробки та налаштований на одночасне перетворення декількох записів. Такий підхід дозволяє підвищити пропускну здатність системи та ефективніше використовувати доступні обчислювальні ресурси.

Для оцінки навантаження на інформаційну систему використовується модель навантаження, параметри якої наведено в Таблиця 3.1. Передбачається, що одночасно з системою можуть працювати до 100 користувачів, кожен з яких має доступ у середньому до 50 функцій системи. Кожна функція, у свою чергу, генерує близько 5 запитів, а середній час однієї користувацької сесії становить 300 секунд. Зазначені параметри дозволяють оцінити інтенсивність запитів, загальне навантаження на обчислювальні та мережеві ресурси, а також визначити вимоги до масштабованості та продуктивності системи в умовах реальної експлуатації.

Таблиця 3.2

Модель навантаження

Параметр	Значення
Кількість одночасних користувачів	100
F, Кількість функцій в системі	50
Кількість запитів на одну функцію	5
T, Середній час сесії	300 sec

Кількість запитів в секунду розраховується за формулою:

$$QPS = \frac{U \times Q_f}{T}, \tag{3.1}$$

де U - кількість одночасних користувачів;

Q_f - кількість запитів на одну функцію;

T - середній час сесії.

Для моделі навантаження (Таблиця 3.2), кількість запитів складає:

$$QPS = \frac{200 * 5}{300} = 3.33 \text{ запитів/с}$$

З урахуванням піків навантаження, з коефіцієнтом х5:

$$QPS_{peak} = QPS * 5 \approx 17 \text{ запитів}/c$$

Далі переходимо до розрахунку характеристик процесора (CPU):

$$CPU_{query} = C_{parse} + C_{plan} + C_{scan} + C_{join}, \quad (3.2)$$

де

$C_{parse} + C_{plan} = 0.2$ мс

$C_{scan} = 1$ мс

$C_{join} = 4$ мс

Середній запит:

$$CPU_{query} = 4.2 \text{ мс} = 0.0042 \text{ с}$$

CPU завантаження з урахуванням пікових навантажень:

$$CPU = QPS_{peak} \times CPU_{query} \quad (3.3)$$

$$CPU = 17 * 0.0042 = 0.0714 \text{ CPU-сек/сек}$$

Визначимо мінімальну кількість ядер для середовища в якому будуть виконуватись запити, резервуючи не менше 30% на управління процессами, перемикання контексту, системними та програмними перериваннями операційної системи:

$$Cores = \frac{0.0714\ CPU-\text{сек/сек}}{(100\% - 30\%)} \approx 0.1$$

Припустимо, що 90% часу буде використовуватись на сканування таблиць та створення об’єднань. Маючи коефіцієнт складності виконання K_{bd} подібних операцій:

$$K_{bd} = \frac{\text{середній розмір таблиці}}{\text{розмір індексів}} \quad (3.4)$$

Визначимо кількість ядер з розрахунком на 1ТБ даних і 20% індексів:

$$K_{db} = 5$$

$$Cores = 0.1 \text{ x } K_{db} \text{ x } 10 = 5,$$

де x10 - резерв для 10-кратного використання CPU для задач агрегації та узгодження даних при паралельному виконанню запитів.

Ефективність багатопоточності залежить від структури задачі, використання ресурсів і накладних витрат, проте, теоретичний розрахунок і узагальнені практичні результати говорять про показник - в найкращому випадку 30% від загального показника швидкості роботи системи.

Мінімальна конфігурація для 1ТБ даних і 100 користувачів: **5 CPUs**.

Рекомендовано: **8-16 CPUs**.

Наступним кроком розраховуємо оперативну пам’ять (RAM):

$$RAM = BufferPool + QueryMemory + IndexCache + OS \quad (3.5)$$

де $BufferPool$ = 10%-30% від розміру БД.

Для початкового об’єму даних 1 ТБ:

$$BufferPool = 1 \text{ ТБ } * 20\% = 200 \text{ ГБ}$$

Припустимо, що розмір одного запиту займає приблизно 50 МБ, для $Q_{concurrent}$ = 50 одночасних виконань:

$$RAM_{query} = Q_{concurrent} * 50 \text{ МБ} = 2.5 \text{ ГБ}$$

Індекси БД займають 20% від загального розміру, виділяється 50% на кешування, таким чином,

$$RAM_{index} = 1 \text{ ТБ} * 20\% * 50\% = 100 \text{ ГБ}$$

Для потреб операційної системи і кеша файлової системи виділяється:

$$RAM_{OS} = 16 \text{ ГБ}$$

Таким чином, за формулою (3.5) визначимо необхідний розмір оперативної памяті:

$$RAM = 200 + 100 + 2.5 + 16 \approx 320 \text{ ГБ}$$

Мінімальний об’єм оперативної пам’яті: **320 ГБ**.

Рекомендовано: **512 ГБ**.

Звідси, апаратне забезпечення, що задовольняє заплановане навантаження на систему виглядає наступним чином:

Таблиця 3.3

Розраховане апаратне забезпечення

CPU	ОЗУ,	Швидкість передачі даних,
6 ядер, 3.2ГГц	384 ГБ	500 МБ/с

3.1.2 Базова пропускна здатність інформаційної системи

За відсутності індексів або механізмів денормалізації виконання запиту здійснюється за найменш ефективним сценарієм обробки даних. На першому

етапі система управління базами даних змушена виконувати повне сканування таблиці значень (full table scan) з метою відбору записів, що відповідають умовам запиту. Такий підхід призводить до значних витрат часу та ресурсів, особливо у випадку великих таблиць з мільйонами або мільярдами рядків.

Після відбору необхідних записів виконується операція з'єднання (JOIN) з таблицею ваг, яка використовується для зіставлення відповідних значень і застосування вагових коефіцієнтів до вибраних даних. За відсутності оптимізуючих структур зберігання або попередньо обчислених зв'язків операція JOIN також потребує значних обчислювальних ресурсів, оскільки виконується над великими наборами даних і може супроводжуватися додатковими операціями сортування або хешування.

На завершальному етапі обробки здійснюються агрегаційні операції, такі як обчислення сум, середніх значень або інших зведених показників, що формують підсумкові результати запиту. Виконання агрегацій над великим обсягом проміжних даних додатково збільшує навантаження на процесор і підсистему пам'яті, а також впливає на загальний час виконання запиту.

У сукупності повне сканування таблиць, ресурсоємні операції з'єднання та подальша агрегація призводять до лінійного або квазілінійного зростання часу виконання запиту залежно від обсягу даних. Це обґрунтовує доцільність використання індексів, денормалізованих структур або попередньо агрегованих даних для підвищення продуктивності системи обробки даних у реальних умовах експлуатації.

Припустимо, що система повторно використовує приблизно третину даних, збережених у результаті попередніх виконань реалізованих в ній функцій, а решта даних формується шляхом генерації нових значень на основі раніше отриманих результатів, що дає змогу оцінити обсяг обчислень і, в цілому, знайти шляхи щодо збільшення загальної ефективності повторного використання інформації.

Технічні характеристики апаратної частини довготривалого зберігання даних істотно впливають на доступність інформації для її повторного використання та подальшої обробки. Під час роботи з великими масивами даних ключову роль відіграють параметри підсистеми зберігання, зокрема тип накопичувача, швидкість читання та запису, час доступу до даних і пропускна здатність інтерфейсу підключення. Саме ці характеристики визначають загальний час отримання даних із довготривалого сховища та безпосередньо впливають на продуктивність системи в сценаріях повторного використання інформації.

Тактова частота центрального процесора та кількість його ядер, які забезпечують виконання програмних інструкцій у паралельному режимі, у більшості випадків значно перевищують можливості накопичувачів щодо зчитування даних у довільному порядку. У результаті обчислювальні ресурси процесора використовуються ефективно лише після того, як дані вже завантажені з довготривалого сховища в оперативну пам’ять. Тому параметри процесора в такому контексті переважно впливають на швидкість перетворення та обробки даних, але не визначають ефективність їх первинного доступу.

З огляду на це, під час аналізу доступності повторно використовуваних даних процесорні характеристики можуть бути знехтувані на даному етапі дослідження, оскільки вони не є вузьким місцем системи. Основним обмежувальним фактором у цьому випадку виступає продуктивність підсистеми зберігання, яка визначає затримки доступу до даних і, відповідно, загальний час виконання операцій читання. Такий підхід дозволяє сфокусувати аналіз на найбільш критичних компонентах апаратної архітектури та отримати більш об’єктивну оцінку ефективності повторного використання даних у системі.

Таким чином, саме система накопичення даних (технологія, контролер і драйвер) є ключовим компонентом при розрахунках систем для роботи з великими об'ємами інформації.

Системи накопичення даних, що базуються на **SATA SSD** технології, пропонують доступ до читання даних з носія зі швидкістю **~500-560 МБ/s** визначена межею інтерфейсу SATA III. Існують більш сучасні і швидші технології **NVMe M.2 SSD**, що декларують швидкість доступу до даних починаючи від **~1000 МБ/с**, а топові моделі **v.5** можуть сягати **3500-7000+ МБ/с**.

Таблиця 3.4

Визначення повторно використовуваного обсягу даних

Параметр	Значення
Частка даних	30%
Обсяг сканування	300 ГБ

За таких умов стає можливим оцінити час, необхідний для отримання заданого обсягу даних із довготривалого сховища. Виходячи з відомої пропускної здатності інтерфейсу, можна визначити мінімальний теоретичний час читання даних, припускаючи послідовний доступ до інформації та відсутність додаткових затримок, пов’язаних із обробкою запитів, конкуренцією за ресурси або фрагментацією даних:

$$T = \frac{300\text{ГБ}}{500\text{МБ}} = \frac{300000\ \text{МБ}}{500\ \text{МБ}} = 600s$$

Таким чином, коефіцієнт виконання аналітичних запитів для отримання даних для повторного використання становить:

$$Q = \frac{1}{600} = 0.0017\ \text{запитів}/c$$

В свою чергу, для 50 функцій, отримаємо значення, що визначає зведену розраховану характеристику системи для отримання аналітичної інформації:

$$0.0017\ *\ 50\ = 0.085\ \text{запитів/с}$$

3.1.3 Оптимізація 1. Оптимізація запитів, збережені процедури

Оптимізація продуктивності у реляційних системах управління базами даних базується на комплексному поєднанні синтаксично вивірених SQL-запитів, архітектурно обґрунтованому використанні збережених процедур та стратегічному проектуванні систем індексації.

Застосування збережених процедур дозволяє ефективно перенести складну бізнес-логіку безпосередньо на рівень сервера бази даних, що радикально мінімізує мережеві витрати на передачу проміжних результатів та забезпечує багаторазове використання вже оптимізованих і скомпільованих планів виконання.

Під час обробки кожного запиту інтелектуальний оптимізатор СУБД проводить глибокий аналіз актуальної статистики розподілу даних та наявних індексних структур, вибираючи математично найбільш вигідний і найменш ресурсомісткий шлях доступу до інформації.

Ефективність такої системи суттєво підвищується завдяки використанню індексів, які дозволяють отримувати необхідні результати без звернення до основного сховища таблиць, та автоматичному вибору оптимальних алгоритмів з’єднання великих масивів даних. Впровадження цих методів забезпечує високу швидкість відгуку системи при конкурентному доступі багатьох користувачів, створюючи надійний фундамент для безперебійної роботи аналітичних модулів та інструментів візуалізації бізнес-процесів у реальному часі.

Окрім того, сучасні механізми оптимізації враховують можливості багатоядерних процесорів для паралельного виконання складних операцій, що дозволяє зберігати стабільну продуктивність навіть в умовах стрімкого зростання обсягів накопиченої інформації.

Індексація реалізується через спеціальні структури даних, насамперед B-tree та hash-індекси: B-tree індекси забезпечують логарифмічний час пошуку і

є оптимальними для діапазонних запитів за ключами та датами, тоді як hash-індекси забезпечують дуже швидкий доступ для точних рівностей (наприклад, за ідентифікаторами або типами показників). У практичних системах індекси створюються за первинними та зовнішніми ключами, часовими полями та категоріальними атрибутами, що дозволяє різко зменшити кількість зчитаних сторінок пам’яті та, відповідно, підвищити пропускну здатність всієї інформаційної системи.

Цей підхід достатньо просто реалізується шляхом інкапсуляції обчислювальної логіки безпосередньо у структурах бази даних, зокрема через використання механізмів попереднього об’єднання таблиць, агрегації показників або складних трансформацій значень у тілі збережених функцій (Рисунок 3.1).

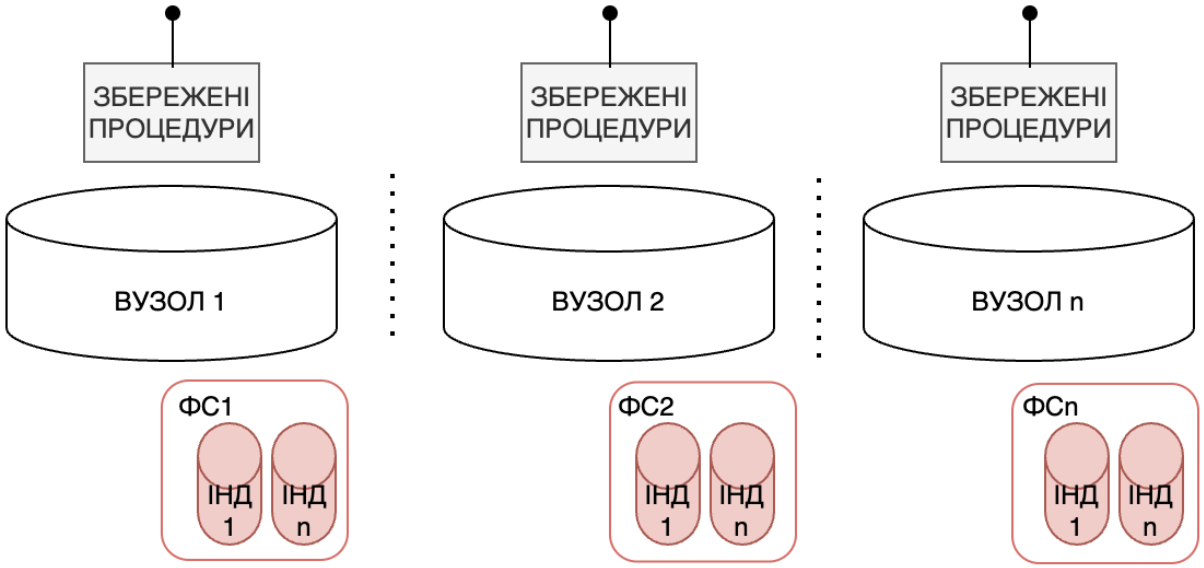


Рис. 3.1 Кінцевий стан системи після виконання оптимізації запитів і реалізації збережених процедур

Хоча самі по собі збережені функції не є першочерговим інструментом для радикального прискорення фізичних операцій читання чи запису, їхня ключова роль полягає в архітектурній стандартизації та виділенні повторюваних алгоритмів у єдині програмні блоки. Такий метод дозволяє

забезпечити уніфікацію обробки даних на рівні сервера, гарантуючи, що складні правила трансформації виконуються ідентично для всіх користувацьких запитів, незалежно від джерела їхнього виникнення.

Винесення бізнес-логіки у збережені функції сприяє суттєвому спрощенню клієнтського коду та зменшенню ймовірності виникнення помилок при реалізації складних аналітичних обчислень.

Крім того, централізоване управління алгоритмами через збережені об’єкти дозволяє проводити локальну оптимізацію коду без необхідності перегляду всієї прикладної логіки системи, що підвищує загальну підтримуваність та масштабованість інформаційної системи.

В результаті досягається високий рівень абстракції, де складна математична обробка та агрегація великих обсягів даних відбуваються максимально близько до місця їхнього фізичного зберігання, що мінімізує зайві обчислювальні витрати на передачу сирих даних.

За рахунок застосування багаторівневих B-tree індексів та селективних hash-індексів обсяг повного сканування таблиць скорочується до приблизно 5% від загального обсягу даних. Це означає, що замість послідовного перегляду всіх сторінок таблиці оптимізатор звертається лише до вузького підмножини індексних сторінок та відповідних рядків, що задовольняють умови запиту. В результаті зменшується кількість дискових операцій вводу-виводу, знижується навантаження на буферний кеш і досягається кратне прискорення виконання запитів, особливо для аналітичних вибірок за ключами, датами та типами показників.

$$1\text{ТБ} * 5\% = 50\text{ГБ}$$

Час, витрачений на отримання 50 ГБ даних при заданій пропускній здатності файлової системи, може досягати суттєвих значень, оскільки на нього впливають не лише номінальна швидкість дискової підсистеми, але й накладні витрати на позиціонування, кешування, конкурентний доступ та обробку

індексних структур. У контексті аналітичних запитів до великого сховища даних це означає, що без оптимізації доступу та попереднього скорочення обсягу сканованих даних загальний час виконання запиту визначається саме швидкістю вводу-виводу, а не обчислювальними ресурсами процесора. Для заданої швидкості файлової системи, час T виконання запиту складає:

$$T = \frac{50\text{ГБ}}{500\text{МБ}/c} = \frac{50000\text{МБ}}{500\text{МБ}/c} = 100\text{ с}$$

Розрахункова пропускна здатність системи після проведення процедур індексації ключових вимірів (час, продукт, регіон), оптимізації структури факт-таблиць та винесення спільної логіки з операцій JOIN і трансформацій у збережені процедури та представлення істотно зростає. За рахунок зменшення обсягу фізичного сканування даних, скорочення кількості об’єднань і повторного використання оптимізованих планів виконання, система здатна обробляти значно більшу кількість аналітичних запитів за одиницю часу при тій самій апаратній конфігурації, забезпечуючи стабільну роботу навіть при навантаженні, характерному для сховища даних обсягом 1 ТБ і сотень одночасних користувачів.

Тоді, пропускна здатність системи після додавання індексів, становить:

$$Q = \frac{50\text{ запитів}}{100\text{ с}} = 0.5\text{ запитів/с}$$

3.1.4 Оптимізація 2. Розділення даних

Логічне розділення даних у сховищі реалізується на основі класичної моделі «зірка» або «сніжинка» за принципом «факт-виміри», що створює стійкий фундамент для лінійного масштабування та високопродуктивної аналітичної обробки. Таблиці типу Fact (факти) виступають центральним ядром архітектури, акумулюючи величезні масиви кількісних показників і транзакційних подій, таких як числові значення метрик (value), вагові

коефіцієнти (weight) та часові мітки (time). Вони слугують єдиним джерелом істини для розрахунку агрегованих показників, забезпечуючи можливість деталізації (drill-down) до рівня окремої операції чи продукту в конкретному регіоні.

Паралельно з цим, таблиці типу Dimension (виміри), що описують час, продукти, географію та інші категорії, містять розширені набори текстових атрибутів, метаданих та складних ієрархій. Саме ці структури дозволяють кінцевим користувачам виконувати багатовимірну фільтрацію, логічне групування та побудову динамічних зрізів даних для глибинного аналізу бізнес-процесів. Завдяки такій декомпозиції радикально мінімізується дублювання інформації (надмірність), оскільки детальні описи зберігаються в одиничному екземплярі у довідниках, а не повторюються у кожному рядку багатомільйонної факт-таблиці.

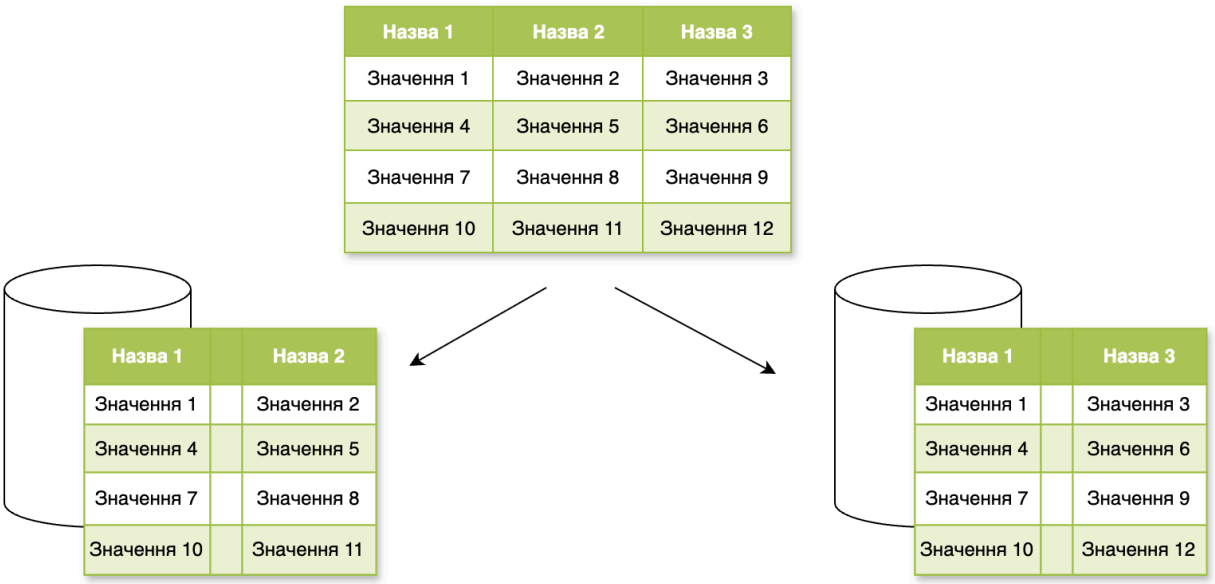


Рис. 3.2 Діаграма розділення даних

Цей архітектурний підхід суттєво спрощує процес оновлення довідникової інформації, дозволяючи змінювати атрибути об'єктів без переписування історичних транзакційних даних. Крім того, модель

«факт-виміри» критично оптимізує виконання складних SQL-запитів: замість ресурсомісткого з'єднання кількох гігантських таблиць, система виконує приєднання (JOIN) компактних вимірів до центрального факту, що дозволяє оптимізатору СУБД ефективніше використовувати індекси та оперативну пам'ять. У результаті забезпечується висока швидкість візуалізації звітів та стабільна робота системи навіть при інтенсивному зростанні обсягів накопиченої інформації.

Застосування архітектурної моделі «факт-виміри» у поєднанні зі стратегічно нормалізованими довідковими таблицями дозволяє досягти радикального приросту продуктивності, знижуючи кількість високовартісних операцій JOIN у типових аналітичних сценаріях орієнтовно в п’ять разів.

У традиційних денормалізованих або складних реляційних структурах виконання запиту часто потребує побудови громіздких ланцюжків з’єднань між численними таблицями великого обсягу, що призводить до експоненціального зростання навантаження на підсистему введення-виведення та центральний процесор.

Натомість запропонована модель спрямовує обчислювальний процес у русло обмеженої кількості приєднань компактних dimension-таблиць до єдиної центральної факт-таблиці, що дозволяє оптимізатору СУБД значно ефективніше використовувати алгоритми Hash Join або Nested Loops.

Така структура суттєво мінімізує обсяг проміжних наборів даних, які необхідно утримувати в оперативній пам’яті під час виконання запиту, що запобігає переповненню тимчасового простору (tempdb) та знижує ризик деградації продуктивності при конкурентному доступі багатьох користувачів.

Суттєве скорочення обчислювальних витрат напряму конвертується у зменшення часу відгуку системи, дозволяючи отримувати результати аналізу практично в режимі реального часу навіть на масивах даних критичного обсягу. Окрім суто технічних переваг, такий підхід забезпечує стабільність планів

виконання запитів, оскільки передбачуваність структури «зірка» дає змогу базі даних заздалегідь оптимізувати шляхи доступу до інформації.

В результаті, перехід до даної моделі стає ключовим фактором масштабованості аналітичної платформи, перетворюючи ресурсомісткі процеси обробки Big Data на швидкі та прогнозовані операції, необхідні для оперативної візуалізації бізнес-показників.

За рахунок індексації та розділення даних у моделі «факт-виміри» фактичний обсяг даних, що підлягає фізичному скануванню під час виконання типового аналітичного запиту, скорочується до приблизно 10 ГБ. Це означає, що з загального сховища обсягом близько 1 ТБ система звертається лише до релевантних партицій та індексних діапазонів, зменшуючи навантаження на дискову підсистему та забезпечуючи передбачувану й стабільну продуктивність запитів навіть при зростанні обсягу даних. Тому для отримання зменшеного об’єму даних, час доступу T до інформації буде складати:

$$T = \frac{10000}{500} = 20s$$

Після проведення стратегічної денормалізації та впровадження моделі логічного розділення таблиць пропускна здатність Q інформаційної системи демонструє суттєве зростання, що пояснюється радикальною зміною механіки доступу до даних.

Завдяки тому, що найбільш критичні для аналітичних розрахунків атрибути інтегруються безпосередньо у структуру факт-таблиць, система уникає необхідності виконання каскадних з’єднань для отримання базових метрик, що значно знижує навантаження на шину пам’яті та центральний процесор.

Виділення довідникових вимірів у компактні структури дозволяє максимально ефективно використовувати кеш-пам’ять СУБД, оскільки невеликі

за обсягом таблиці вимірів постійно перебувають в оперативній пам’яті, забезпечуючи миттєвий доступ до описових атрибутів.

Збільшення показника Q також зумовлене зменшенням обсягу операцій введення-виведення, оскільки зчитування одного рядка факт-таблиці тепер надає майже вичерпний набір даних для більшості фільтрів та групувань.

Це дозволяє системі опрацьовувати значно більшу кількість паралельних запитів на одиницю часу, забезпечуючи лінійну масштабованість при зростанні інтенсивності користувацького навантаження.

Окрім того, мінімізація логічних зв’язків спрощує роботу оптимізатора запитів, який отримує можливість будувати максимально короткі та стабільні плани виконання, що є критично важливим для підтримки високої пропускної здатності в умовах роботи з великими масивами даних (Big Data).

У кінцевому підсумку, така реструктуризація перетворює сховище з пасивної структури збереження на високопродуктивний обчислювальний хаб, здатний підтримувати інтенсивні аналітичні потоки без затримок у чергах обробки.

$$Q = 0.05 \Rightarrow 2.5 \text{ QPS}$$

Пропускна здатність системи після проведення денормалізації та логічного розділення таблиць істотно зростає, оскільки критичні для аналітики атрибути зберігаються безпосередньо у факт-таблицях, а довідникові виміри виділяються у компактні структури і становить 2.5 запитів за секунду.

3.1.5 Оптимізація 3. Кешування

Для радикальної оптимізації навантаження на реляційну базу даних та забезпечення миттєвого відгуку аналітичних інтерфейсів у системі впроваджено механізм кешування попередньо агрегованих показників на рівні проміжного шару обробки. Архітектура цього кешу базується на моделі

«ключ-значення» (Key-Value), що дозволяє досягти значного зменшення часу на доступ до результатів складних обчислень, які в іншому випадку вимагали б повторного сканування мільйонів рядків у основному сховищі.

Процес формування ключа реалізовано через генерацію унікальної композитної структури, що включає ідентифікатори вимірів (наприклад, часовий інтервал, категорію продукту та територіальну приналежність), що забезпечує точну відповідність кешованих даних параметрам користувацького запиту. Значення, що зберігається за цим ключем, містить набір цільових агрегатів, таких як суми, середні значення, зважені показники або унікальні лічильники, розраховані заздалегідь під час процесів фонового оновлення даних або першого звернення до системи.

key = product | region | date

Такий підхід дозволяє перенести значну частку обчислювального навантаження з дискової підсистеми СУБД у швидку оперативну пам’ять, що є критично важливим для підтримання високої частоти оновлення візуалізацій у реальному часі.

Використання кешування агрегатів не лише скорочує час виконання запитів, а й значно підвищує загальну пропускну здатність системи, оскільки звільняє ресурси процесора для виконання складних аналітичних задач, які не піддаються попередньому розрахунку.

Крім того, стратегія кешування передбачає механізми інтелектуальної інвалідації даних, що гарантує актуальність відображуваної інформації при зміні базових фактів у основній базі даних. У результаті поєднання моделі «факт-виміри» з високорівневим кешуванням створює багаторівневу систему захисту від перевантажень, забезпечуючи стабільну роботу аналітичної платформи навіть за умов екстремального зростання обсягів Big Data.

Таким чином, у кеші зберігається не сирий масив транзакцій, а готовий для аналітики агрегат, що дозволяє уникнути повторних обчислень при кожному запиті.

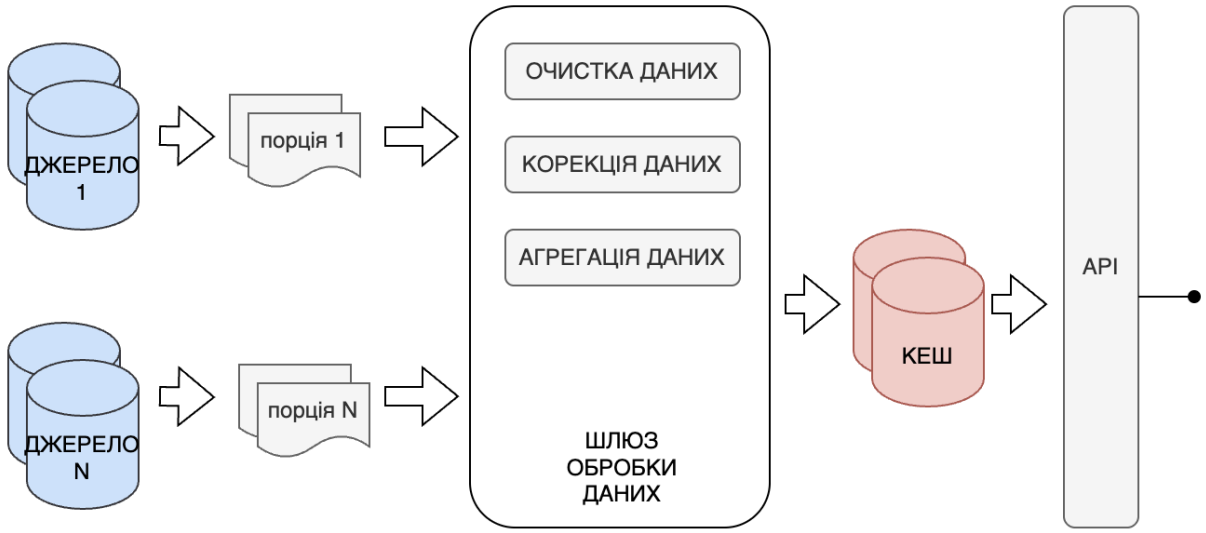


Рис. 3.3 Кешування узгодженого набору даних

Під час виконання запиту система працює лише з ключами кешу, виконуючи прямі операції читання без необхідності звертатися до великих факт-таблиць. Якщо припустити, що в середній аналітичний запит повертається приблизно до 100 ключів, а розмір одного ключа становить близько 100 байт, то загальний обсяг даних, що зчитується з кешу, дорівнює лише близько 10 КБ на запит. Уявимо, що оперативної інформації серед даних - 5%, що для 1ТБ => 50МБ. У порівнянні з читанням десятків гігабайт з дискового сховища це забезпечує на кілька порядків менші витрати вводу-виводу та практично миттєву відповідь системи, що істотно підвищує загальну пропускну здатність аналітичної платформи.

$$100 \times 100 = 10KB$$

Враховуючи досягнуту значну оптимізацію обсягу даних та технічні обмеження файлової підсистеми, що становить 500 МБ/с, розрахований час T на зчитування цільового датасету обсягом 50 МБ становить 0,1 секунди.

$$T = \frac{50\text{МБ}}{500\text{МБ/с}} = 0.1 \text{ с}$$

Цей показник відображає теоретичну межу швидкодії при лінійному читанні, проте в реальних умовах функціонування інформаційної системи на нього також впливають латентність дискових операцій та час на ініціалізацію потоків зчитування.

Отримане значення свідчить про високу ефективність проведеної денормалізації, оскільки мінімізація фізичного розміру даних дозволяє системі працювати в межах пропускної здатності сучасних SSD-накопичувачів, не створюючи критичних черг на рівні введення-виведення.

Далі визначаємо пропускну здатність Q послідовного виконання запитів такої системи для отримання результатів виконання 50 функцій:

$$Q = \frac{1}{0.1} = 10 \text{ запитів/с}$$

3.1.6 Оптимізація 4. Заміна форматів зберігання

Для цієї роботи ключовою перевагою колонкових СУБД перед реляційними є швидкість читання інформації за рахунок позиціонування за лінійним алгоритмом і звуження обсягу прочитаної інформації, обмеженої запитом, на відміну від реляційних баз, де весь рядок має бути вичитаний в оперативну пам'ять та згодом опрацьований.

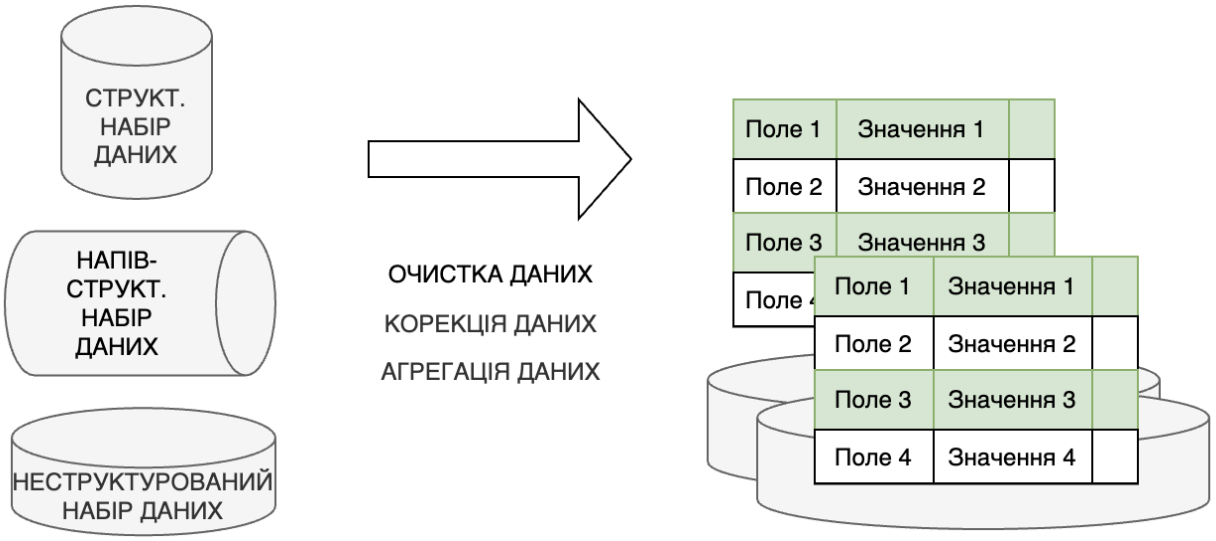


Рис. 3.4 Міграція даних в колонковий формат даних

Перехід від рядкового формату зберігання до колоночного (Column-Oriented DB) дозволяє зчитувати з диску лише ті атрибути, які реально беруть участь в обчисленнях, зокрема value та weight, і ігнорувати всі інші поля. За умови, що ці два стовпці становлять приблизно 10% від загального обсягу сховища розміром 1 ТБ, фактичний обсяг даних для фізичного читання скорочується до близько 100 ГБ. Додатково застосування колоночної компресії з коефіцієнтом приблизно x5 зменшуємо цей обсяг до 20 ГБ реальних даних, які необхідно зчитати з дискової підсистеми для виконання аналітичного запиту.

$$1\text{ТБ} * 10\% = 100\text{ГБ}$$

$$100\text{ГБ} \div 5 = 20\text{ГБ} \quad \text{(застосування компресії)}$$

За умови пропускної здатності файлової системи 500 МБ/с час повного сканування такого обсягу становить:

$$T = \frac{20\text{ГБ}}{500\text{МБ/с}} = \frac{20000\text{МБ}}{500\text{МБ/с}} = 40\text{ с}$$

Якщо припустити, що типовий аналітичний сценарій включає 50 послідовних запитів, то розрахункова пропускна здатність системи за такого режиму, обмеженого продуктивністю файлової системи дорівнює:

$$Q = \frac{50 \text{ запитів}}{T} = \frac{50 \text{ запитів}}{40 \text{ с}} = 1.25 \text{ запитів/с}$$

3.1.7 Результат моделювання

Таким чином ми отримуємо значення теоретичного розрахунку результатів проведеної оптимізації, розраховані для середніх вимог щодо навантаження інформаційної системи.

Таблиця 3.5

Розраховані показники роботи інформаційної системи

Метод	Обсяг даних для агрегації, ГБ	Час, с	Запитів/сек	Прискорення, разів
Без оптимізації	300	600	0.085	x1
Оптимізація запитів, збережені процедури	50	100	0.5	x6
Розділення даних	10	20	2.5	x30
Заміна форматів зберігання	20	40	1.25	x15
Кешування	0.50	0.1	10	x117

Приймаючи до уваги розрахункові характеристики (Таблиця 3.5) пропускної здатності методів оптимізації досліджуваної системи для оптимальної конфігурації апаратних характеристик системи, достатньої для оптимальної роботи системи згідно моделі навантаження (Таблиця 3.2), для кожного методу оптимізації приймається рішення про необхідність підвищення класу апаратного забезпечення.

Таблиця 3.6

Очікування щодо пропускної здатності системи

МЕТОД	**РОЗРОБКА**	**ПІДТРИМКА**	**ПРОПУСКНА ЗДАТНІСТЬ**
ОПТИМІЗАЦІЯ ЗАПИТІВ	50 індексів 50 процедур 100 сутностей	128 vCPUs/512 ГБ/NVMe SSD 2ТБ	Читання: ~10k-30k QPS Запис: ~3k-7k TPS
РОЗДІЛЕННЯ ДАНИХ	75 індексів 130 сутностей	128 vCPUs/512 ГБ/NVMe SSD 2ТБ	Читання: ~8k-25k QPS Запис: ~2k-8k TPS
КОЛОНКОВИЙ ВИГЛЯД	20 факт таблиць 60 таблиць вимірів 30 допоміжних	60 vCPUs / 512 ГБ NVMe SSD 3ТБ	Читання: 0.75-2 QPS Запис: 1-3 TPS
КЕШУВАННЯ	30 сутностей 60 індексів	144 vCPU/1 ГБ/ NVMe SSD 3ТБ	KV читання: 900k QPS KV запис: 450k TPS QL читання: 35k QPS Запис: 25k TPS

Таким чином ми отримуємо збалансовану конфігурацію апаратного забезпечення для отримання максимальних значень швидкодії системи, яка оперує заданим розміром набору даних (Таблиця 3.6).

3.2 Проведення експерименту та результати

Для верифікації гіпотези щодо досягнення оптимальних показників зваженої агрегації в умовах інтенсивного навантаження Big Data було

проведено комплексне практичне застосування розроблених методів оптимізації попередньої обробки даних.

Цей комплекс заходів охоплював інтеграцію алгоритмів зниження розмірності, впровадження моделі «факт-виміри» та механізмів інтелектуального кешування, що дозволило створити контрольоване середовище для тестування обчислювальної ефективності.

Експериментальна перевірка була зосереджена на оцінці швидкості формування зважених агрегатів, де кожен елемент даних корелювався з ваговими коефіцієнтами в реальному часі, що є одним із найбільш ресурсомістких завдань в аналітиці великих масивів.

Проведені випробування дозволили підтвердити, що поєднання структурної декомпозиції таблиць із попередньою фільтрацією на етапі вибірки даних забезпечує стабільну продуктивність системи навіть при експоненціальному зростанні обсягу вхідної інформації.

Таблиця 3.7

Стенд для проведення експериментального дослідження

CPU	RAM	Швидкість файлової системи
8 ядер, 3.2 ГГц	16 ГБ	500 Мб/с

Вказана апаратна конфігурація забезпечує достатню обчислювальну потужність для виконання розрахунків, обробки великих масивів даних та застосування спеціалізованого програмного забезпечення, необхідного для проведення експериментальної та аналітичної частин дослідження.

Таблиця 3.8

Фактичні показники виконання агрегації з перерахунком на отримання цільового набору даних

Метод	Час виконання запиту (мс)	CPU (%)	RAM (МБ)	IO (МБ/с)	QPS,	Запитів/сек розрах.
Без оптимізації	1366	50.82	22.31	0.4	19.67 (50.82МБ)	0.39 (300ГБ)
Оптимізація запитів, збережені процедури	796	26	8.14	46.25	1.257 (8.14МБ)	1.22 (50ГБ)
Розділення даних	833	71.05	202	0.48	1.20 (202МБ)	1.41 (10ГБ)
Заміна форматів зберігання	0.404	106.44	19378	363527	70 (19МБ)	66 (20ГБ)
Кешування	0.019	4.6	0.164	90	52631 (0.16МБ)	1,028 (50МБ)

Таблиця 3.8 містить фактичні результати вимірювань загального часу виконання програми, використання CPU і RAM, утилізації IO для виконання одного запиту. Таблиця доповнена виведеною пропускною здатністю яку обчислено як відношення часу виконання на читання фактичного обсягу даних до припущеного розміру ефективної вибірки досліджуваної системи.

3.3 Аналіз результатів та рекомендації

Згідно з дослідженням [22, с. 14928], для більшості задач достатньо одноядерного процесора, оскільки збільшення кількості ядер і потоків не дає суттєвого зростання продуктивності. Навпаки, гонка потоків може навіть зменшити ефективність виконання. Вибір процесора з високою тактовою частотою та великим кешем рівня L3 забезпечує максимальну продуктивність при виконанні послідовних операцій.

Експериментально підтверджено [23, с. 14928], що архітектурні особливості сучасних процесорів, такі як розширені векторні інструкції та об'єм кеш-пам'яті L3, відіграють вирішальну роль у мінімізації часу обробки складних зважених агрегатів.

Аналіз результатів, представлених Фактичних замірів продуктивності (Таблиця 3.8), свідчить про специфічний профіль навантаження, характерний для колонкових баз даних: алгоритми агрегації демонструють виражену залежність від обчислювальних потужностей, споживаючи значний обсяг процесорного часу при відносно помірному використанні оперативної пам'яті.

Оскільки колонкове зберігання дозволяє зчитувати лише необхідні для розрахунку атрибути, основне навантаження переміщується з операцій копіювання даних у пам'ять на етап їхньої безпосередньої обробки, де швидкість виконання операцій агрегації прямо корелює з тактовою частотою кожного ядра та загальною кількістю доступних обчислювальних потоків.

Однак після досягнення точки насичення обчислювальних потужностей, коли конфігурація центрального процесора повністю задовольняє потреби алгоритмів у паралельних обчисленнях, фокус обмежень продуктивності неминуче зміщується на наступний рівень ієрархії системи.

У цьому контексті наступним критичним обмежуючим фактором (bottleneck) стають технічні характеристики файлової системи, зокрема швидкість лінійного зчитування та латентність доступу до дисків.

Навіть за умови надлишкової потужності процесора, система не зможе продемонструвати вищу продуктивність, якщо швидкість подачі даних із накопичувачів не відповідатиме темпу їхньої обробки.

Таким чином, отримані дані підкреслюють необхідність збалансованого підходу до вибору апаратної конфігурації, де потужність CPU повинна бути синхронізована із пропускною здатністю підсистеми зберігання для забезпечення максимальної загальної ефективності обробки Big Data.

Шляхи збільшення продуктивності інформаційної системи охоплюють багатошаровий комплекс заходів, що починаються з фундаментальної оптимізації способу збереження даних. Це включає усунення надлишковості за допомогою методів нормалізації та актуалізації часто використовуваних цільових і проміжних результатів через впровадження ефективних стратегій кешування агрегованих даних за моделлю «ключ-значення».

Критично важливим етапом є врахування апаратних особливостей середовища на стадії проектування системи. Необхідно заздалегідь аналізувати доступну кількість ядер процесора, архітектуру паралелізму, характеристики дискових підсистем (швидкість I/O, латентність SSD/NVMe) та пропускну здатність мережевого доступу. Це дозволяє уникнути вузьких місць (bottlenecks) ще до початку розробки.

Паралельно з цим, програмна конфігурація СУБД потребує тонкого налаштування: управління пулами підключень для мінімізації витрат на встановлення сесій, оптимізація розміру кешу даних та планів виконання запитів, вибір найбільш ефективних параметрів індексації та адекватних типів даних для зберігання, а також управління механізмами блокування і конкурентного доступу до записів.

Лише після синхронізації всіх цих аспектів - правильного збереження даних (модель «факт-виміри»), використання багаторівневого кешування, апаратної оптимізації та програмних налаштувань СУБД - можна переходити до цільового використання системи та оцінки ефективності обчислень.

Такий комплексний підхід дозволяє значно підвищити загальну пропускну здатність інформаційної системи та зменшити накладні витрати з метою оптимізації алгоритмів обробки даних у програмних компонентах, при цьому зберігаючи незмінною або навіть ускладнюючи початкову бізнес-логіку.

3.4 Метрики оцінювання систем Big Data

Для забезпечення об’єктивної та всебічної оцінки ефективності проектованої інформаційної системи, а також для верифікації її відповідності встановленим експлуатаційним вимогам, необхідно сформувати науково обґрунтований набір метрик продуктивності.

Створення такої системи показників дозволяє не лише кількісно оцінити поточний стан системи в момент її розгортання, а й формує надійний фундамент для тривалого моніторингу та порівняльного аналізу в майбутньому.

Формалізація ключових метрик забезпечує можливість проведення регулярного регресійного тестування та бенчмаркінгу, що є критично важливим для оцінки оперативних показників у динаміці.

Такий підхід дозволяє коректно зіставляти ефективність запропонованих методів із альтернативними підходами, а також об’єктивно вимірювати вплив технологічних оновлень, модернізації апаратного забезпечення або вдосконалення алгоритмічної бази на загальну продуктивність системи. Наявність стандартизованих показників трансформує процес оцінки з суб’єктивного спостереження у строгу наукову процедуру, що піддається відтворенню та верифікації.

У межах даного розділу буде проведено детальний розгляд основних метрик, які складають ядро аналітичного інструментарію під час тестування системи. Особлива увага приділяється інтеграції цих показників із теоретичним базисом, викладеним раніше: ми продемонструємо практичне застосування

математичних моделей з розділу 1 для обчислення часових затримок, пропускної здатності та коефіцієнтів завантаження ресурсів.

Інтерпретація отриманих результатів крізь призму математичного моделювання дозволить не просто констатувати фактичні значення швидкодії, а й виявити приховані закономірності у функціонуванні системи, прогнозувати її поведінку при екстремальних навантаженнях та визначати оптимальні вектори для подальшої оптимізації програмних компонентів.

Таким чином, сформований набір метрик стає універсальною мовою опису ефективності, що поєднує теоретичні розрахунки з реальними умовами експлуатації інформаційної системи.

3.4.1 Пропускна здатність

Пропускна здатність (Throughput) є фундаментальною характеристикою при оцінці продуктивності інформаційних систем, оскільки вона кількісно визначає інтенсивність обробки даних, а саме кількість успішно завершених операцій за фіксовану одиницю часу.

Цей показник виступає ключовим критерієм для підтримки галузевих стандартів оцінки, аудиту та сертифікації інформаційних систем, дозволяючи встановити чіткі відповідності між технічними можливостями платформи та вимогами бізнес-процесів до швидкодії.

У контексті сучасних багаторівневих архітектур пропускна здатність слугує індикатором ефективності використання апаратних ресурсів, демонструючи, наскільки якісно система здатна масштабуватися під високим навантаженням.

Специфіка аналізу пропускної здатності в рамках підсистеми кешування полягає у визначенні кількості транзакцій на читання (read ops) або запис (write ops), які система здатна обробити за секунду без деградації часу відгуку. Цей показник безпосередньо корелює з ефективністю алгоритмів хешування та

швидкістю доступу до оперативної пам'яті, що дозволяє оцінити здатність кешу нівелювати «пікові» навантаження, які виникають при зверненні до основної бази даних.

Висока пропускна здатність кешу свідчить про мінімізацію накладних витрат на синхронізацію потоків та ефективне управління конкурентним доступом до даних.

Пропускна здатність характеризує кількість операцій або транзакцій, які інформаційна система здатна обробити за одиницю часу. Вона є ключовим показником продуктивності системи, оскільки дозволяє оцінити її здатність обслуговувати запити користувачів у заданих умовах експлуатації. Зазвичай пропускна здатність вимірюється у таких одиницях:

- TPS (Transactions Per Second) - кількість транзакцій, оброблених системою за одну секунду;
- TPM (Transactions Per Minute) - кількість транзакцій, оброблених системою за одну хвилину.

Вибір одиниці вимірювання залежить від характеру системи та масштабу навантаження: TPS використовується для систем з високою частотою транзакцій, тоді як TPM може бути більш зручним для систем із помірним або нерегулярним навантаженням.

Більше того, пропускна здатність є динамічним показником, який дозволяє виявити поріг насичення системи, після якого подальше зростання інтенсивності запитів призводить до експоненціального збільшення черг обробки.

Формалізація цієї метрики у поєднанні з математичними моделями дозволяє не лише констатувати поточний стан системи, а й здійснювати прогнозне моделювання (capacity planning) для майбутнього розширення системи. Таким чином, пропускна здатність стає універсальним мірилом продуктивності, що об'єднує програмні алгоритми обробки, мережеву інфраструктуру та дискову підсистему в єдину метричну систему координат.

Фактори, що визначають пропускну здатність системи, мають комплексну природу і охоплюють як апаратний, так і програмний рівні ієрархії інформаційної системи.

Як було зазначено раніше, критичний вплив мають архітектурні параметри центрального процесора, де кількість ядер, ефективність багатопотоковості (паралелізму) та частотні характеристики безпосередньо лімітують швидкість математичних обчислень і логічного виводу. Об’єм та пропускна здатність оперативної пам’яті (RAM) визначають швидкість обміну даними між кеш-рівнями та накопичувачами, тоді як продуктивність дискової підсистеми (I/O throughput) та мережевого обладнання формують фізичну межу швидкості подачі даних до обчислювальних вузлів.

Окремим вагомим фактором є накладні витрати на організацію обчислювального процесу безпосередньо всередині програмного забезпечення. Це стосується не лише ефективності реалізації алгоритмів у внутрішньому середовищі виконання, а й специфіки формування аналітичних запитів до зовнішніх джерел та інтеграційних шлюзів.

Ступінь оптимізації вихідного програмного коду, використання векторних інструкцій та мінімізація циклів очікування (wait states) дозволяють суттєво підвищити корисне навантаження на систему. Окрім того, на показник пропускної здатності суттєво впливає структурна складність транзакцій: кожна додаткова операція трансформації даних, інтенсивний пошук у неіндексованих масивах або складні математичні перетворення збільшують обчислювальну вартість одиничного запиту, що призводить до природного зниження загальної кількості оброблених операцій за одиницю часу.

1. Вбудований механізм логування:
    a. Програма фіксує часові мітки (timestamps) для кожної транзакції;
    b. Аналіз журналів дозволяє оцінити швидкість виконання.
2. Використання тестових сценаріїв (benchmark):

a. Створення типових трансакційних сценаріїв для порівняння.

b. Автоматизація вимірювання за допомогою інструментів моніторингу.

Діаграма концепції пропускної здатності зображена на рисунку 3.5, де продемонстровано вимірювання кількості даних, що можуть бути передані та отримані між компонентами системи (сервером, мережею, мережевим обладнанням і кінцевим пристроєм) за одиницю часу.

Рис. 3.5 Діаграма концепту пропускної здатності

Для оцінки продуктивності системи та порівняння її характеристик з іншими методами оптимізації використаємо формулу 3.1 для U = 1 яку було використано для моделювання системи.

Щоб порахувати це значення, припустимо, що за $T = 10$ секунд та $Q_f = 5000$ запитів та всі вони були успішні, то розрахунок відбувається таким чином:

$$QPS = \frac{1 * 5000}{10} = 500 \text{ запитів}/c$$

Показник пропускної здатності QPS показує, що система може обробляти в середньому 500 запитів за секунду. Ці значення підраховуються для кожного методу, алгоритму тощо, щоб визначити, наскільки вони забезпечують очікувані вимоги продуктивності системи.

3.4.2 Затримка

Часовий інтервал, необхідний комп'ютерній системі для завантаження даних з носія зберігання до їх появи в регістрах центрального процесора є затримкою. Відповідно до рисунку 3.6 латентність кеш-пам'яті першого рівня (L1) становить 1,5 наносекунди, що відповідає трьом тактам процесора (1 наносекунда дорівнює 10^{-9} секунди). Завантаження даних із кеш-пам'яті рівнів L2 та L3 потребує дещо більше часу, проте все одно виконується швидше, ніж звернення до основної оперативної пам'яті.

Латентність оперативної пам'яті на становить 60 наносекунд (120 тактів). Показники затримок для різних типів накопичувачів є значно вищими: 150 000 наносекунд для твердотільних накопичувачів (SSD) та 10 000 000 наносекунд для жорстких дисків (HDD) [24, с. 358].

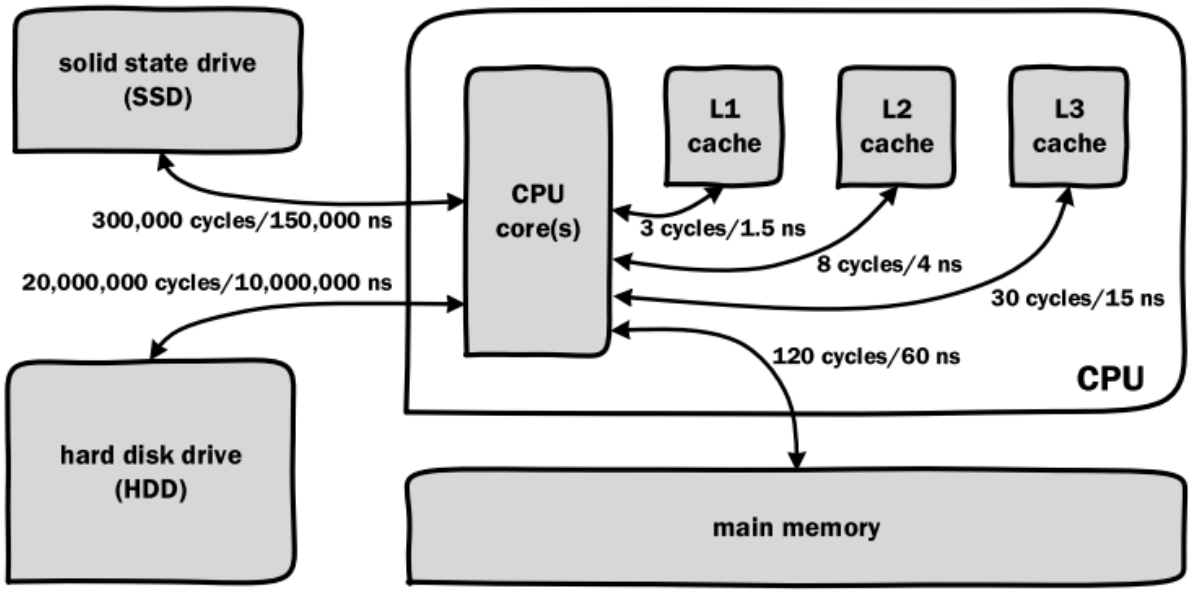


Рис. 3.6 Приклади затримки в апаратному середовищі

Системи тривалого зберігання даних оснащуються комплексом механізмів для зменшення шкідливого впливу затримки при отримання

інформації під час передачі її між компонентами. Необхідно контролювати цей параметр, щоб не сумувати затримку обох частин систем, намагаючись анулювати похибку нульової затримки та проаналізувати способи її усунення.

Цей параметр напряму впливає на збільшення часу доступу до запитуваних даних, оскільки при доступі до даних, що знаходяться на локальному вузлі затримка значно менша, ніж при отриманні даних з віддаленого компонента, тож затримка критично впливає на високонавантажені системи, де кожна мілісекунда впливає на продуктивність. Оптимізація взаємодії інтерфейсу системи з системою зберігання даних позитивно впливає на швидкодію системи та покращує досвід користувача.

Рис. 3.7 Приклади затримки у мережевому середовищі

Рисунок 3.7 ілюструє затримку, яка існує при передачі пакета даних від сервера до кінцевого пристрою через інтернет та мережеве обладнання. У контексті підготовки даних, цей фактор спричиняє зменшення обсягів формування нових або отримання цільових даних, і навпаки - зменшення його впливу дозволяє мінімізувати час передачі даних та покращити продуктивність системи.

Така поведінка системи є передбачуваною, проте, вимушеною, оскільки немає можливості зберегти всі дані на одному вузлі для наскрізної адресації, навіть у режимі віртуалізації. Ця проблема на теперішній час вирішення не має, оскільки, навіть якщо ми забезпечимо програмні реалізації спільним апаратним або віртуальним пристроєм для тривалого зберігання даних, проблема

“вузького” місця буде переміщена в програмну реалізацію багатопоточності і конкурентного доступу до фізичних даних в мікроконтролерах.

Імплементація механізмів обробки затримок безпосередньо на фізичному рівні або на рівні низькорівневих абстракцій дозволяє радикально підвищити швидкість ідентифікації та вибору оптимальної стратегії виконання запитів, проте такий підхід неминуче призводить до втрати гнучкості програмного забезпечення.

Це зумовлено вимушеним зміщенням логіки керування поведінкою системи значно глибше по програмному стеку: від гнучкого рівня прикладного додатку до жорстко детермінованого рівня доступу до даних, що реалізується на рівні драйверів пристроїв або навіть безпосередньо в апаратній логіці контролерів.

Така архітектурна трансформація означає, що стратегічні рішення щодо черговості обробки, пріоритезації запитів та методів компенсації латентності стають частиною інфраструктурного шару, що значно ускладнює їхню подальшу адаптацію до динамічних змін бізнес-логіки.

Хоча перенесення цих функцій «вниз» дозволяє мінімізувати накладні витрати на перемикання контексту та обробку високорівневих переривань, воно водночас створює жорстку залежність прикладного ПЗ від специфіки конкретного апаратного забезпечення або пропрієтарних драйверів.

Таким чином, вибір на користь низькорівневої обробки затримок є свідомим компромісом між максимальною сирої продуктивністю та адаптивністю архітектури, де виграш у швидкості пошуку стратегії виконання має бути співмірним із додатковими витратами на підтримку та складність модернізації таких «зашитих» рішень.

Гарантії затримки можна досягти в певних середовищах, якщо ресурси статично розділені (наприклад, виділене обладнання з ексклюзивним розподілом пропускної здатності). Проте такий підхід призводить до менш ефективного використання ресурсів і, відповідно, є дорожчим. Натомість

багатокористувацька оренда з динамічним розподілом ресурсів дає змогу досягти вищого рівня їх використання, що знижує вартість, але водночас супроводжується недоліком у вигляді нестабільних затримок [20, с. 281-286].

Прикладом вдалої реалізації віртуалізації фізичного середовища і організації спільного адресного простору для зберігання даних є система Hadoop [21, с. 43-50], система, яка працює над розподіленою файловою системою HDFS (Hadoop Distributed File System) [18, с. 6] яка дозволяє зберігати надвеликі файли даних розподілених між вузлами у кластері.

Пропускна здатність та латентність (затримка) даних є взаємозалежними метриками, які в сукупності визначають реальну продуктивність інформаційної системи, виражену в кількості успішно оброблених запитів за одиницю часу. Хоча пропускна здатність вказує на загальний обсяг роботи, який система здатна виконати, саме затримки на кожному етапі обчислювального циклу - від зчитування з дискового масиву до передачі через мережеві протоколи - формують фактичний поріг її швидкодії.

Практично доведено, що збільшення тривалості або частоти затримок безпосередньо призводить до деградації загальної кількості операцій, оскільки кожен цикл очікування (wait state) блокує обчислювальні ресурси, не дозволяючи їм переходити до виконання наступних завдань у черзі. У системах, що працюють з великими даними, навіть незначні затримки на фізичному рівні накопичуються експоненціально, створюючи “вузьке” місце, що паралізує паралельну обробку запитів.

Таким чином, критична залежність пропускної здатності від параметрів затримки диктує необхідність мінімізації латентності на всіх рівнях програмного стека:

Часті затримки розмивають обчислювальний бюджет процесора, змушуючи систему витрачати час на перемикання контексту та очікування відповіді від повільних підсистем;

Тривалі затримки обмежують глибину черги запитів, що призводить до простою потужних багатоядерних архітектур, які залишаються недовантаженими через несвоєчасну подачу даних.

Зрештою, оптимізація систем візуалізації та аналітики спрямована на розрив цього негативного зв'язку: чим меншою є латентність кожної окремої операції, тим вищою стає сумарна кількість транзакцій, які система здатна обробити за секунду.

Це забезпечує не лише теоретичну швидкість, а й практичну плавність роботи аналітичних модулів, де кожна секунда затримки означає втрату можливості виконати сотні додаткових аналітичних перетворень.

Баланс між затримкою та пропускною здатністю дозволяє обробляти ще більше запитів за одиницю часу на тому самому обладнанні і в тому самому середовищі, підвищуючи продуктивність. Висока затримка знижує ефективність системи, обмежуючи кількість запитів, які можуть бути оброблені. Взаємозалежність пропускної здатності максимізується за умови мінімальної затримки. Оптимізація затримки (через локальний кеш, CDN, стиснення) безпосередньо покращує продуктивність [22, с. 38].

Затримка (latency) є фундаментальним показником для вимірювання продуктивності інформаційних систем, оскільки вона визначає часовий інтервал між ініціацією запиту та моментом отримання результату. Для отримання об’єктивної картини функціонування системи недостатньо оперувати лише загальними даними; необхідно проводити комплексне вимірювання, що базується на аналізі середнього часу виконання операції та фіксації показників максимальної затримки.

Середній час операції (Average Latency) дозволяє оцінити загальну ефективність системи при типовому навантаженні та прогнозувати середньостатистичну швидкість відгуку для більшості користувацьких сценаріїв. Ця метрика є ключовою для розрахунку базової пропускної здатності

та планування ресурсів, проте вона часто маскує поодинокі збої або системні мікрозатримки, які можуть виникати під час пікових навантажень.

Максимальна затримка (Peak Latency), своєю чергою, є критичним показником для оцінки стабільності та передбачуваності системи. Вона дозволяє ідентифікувати так звані «викиди» (outliers), що виникають через блокування ресурсів, тривалі цикли збирання сміття (Garbage Collection) або затримки на рівні дискової підсистеми. У сучасних системах візуалізації бізнес-процесів саме максимальна затримка визначає найгірший сценарій досвіду користувача (Worst-case Scenario), і її мінімізація є пріоритетним завданням для забезпечення плавної роботи інтерфейсів у реальному часі.

Поєднання цих двох показників дозволяє розрахувати перцентилі затримки (наприклад, p90, p95 або p99), що дає можливість стверджувати, що певний відсоток усіх запитів виконуються швидше за певне порогове значення. Такий підхід до вимірювання продуктивності забезпечує глибоке розуміння динаміки обробки даних:

Аналіз середнього часу вказує на загальну оптимізацію алгоритмів.

Аналіз максимальної затримки виявляє архітектурні недоліки та обмеження апаратного забезпечення.

Зрештою, комплексний моніторинг затримок стає базою для тонкого налаштування системи, де метою є не лише зменшення середнього часу обробки, а й досягнення максимальної детермінованості системи, коли кожна операція виконується у чітко визначених часових межах без непередбачуваних затримок.

Для оцінки середнього часу виконання операції W (напр. вставка, отримання, видалення) та середнього часу відповіді системи R, застосуємо модель масового обслуговування для розрахунку відповідності цих показників щодо вимог цільової системи за формулою:

$$R = \frac{1}{\mu - \lambda} \tag{3.6}$$

$$W = \frac{1}{\mu}, \tag{3.7}$$

де μ - інтенсивність обробки запитів

λ - інтенсивність надходження запитів.

Розрахуємо показники затримки для умовних: $\lambda = 5$ запитів/с та $\mu = 500$ запитів/с.

Середній час відповіді системи:

$$R = \frac{5}{500-5} \approx 2.002 \text{ мс}$$

Середній час виконання однієї операції

$$W = \frac{1}{500} = 0.002 \text{ с} = 2 \text{ мс}$$

3.4.3 Узгодженість даних

Узгодженість даних це ступінь, яка описує гарантію актуальності та цілісності даних при конкурентному доступі або після відмов. Розглянути схему даного механізму можна на рисунку 3.4. Забезпечення узгодженості даних у схемах оптимізації має ключове значення для сучасних програм, оскільки це безпосередньо впливає на цілісність даних і надійність системи. Основною проблемою є виявлення неузгоджених даних і керування ними під час передачі даних до інших вузлів і підсистем.

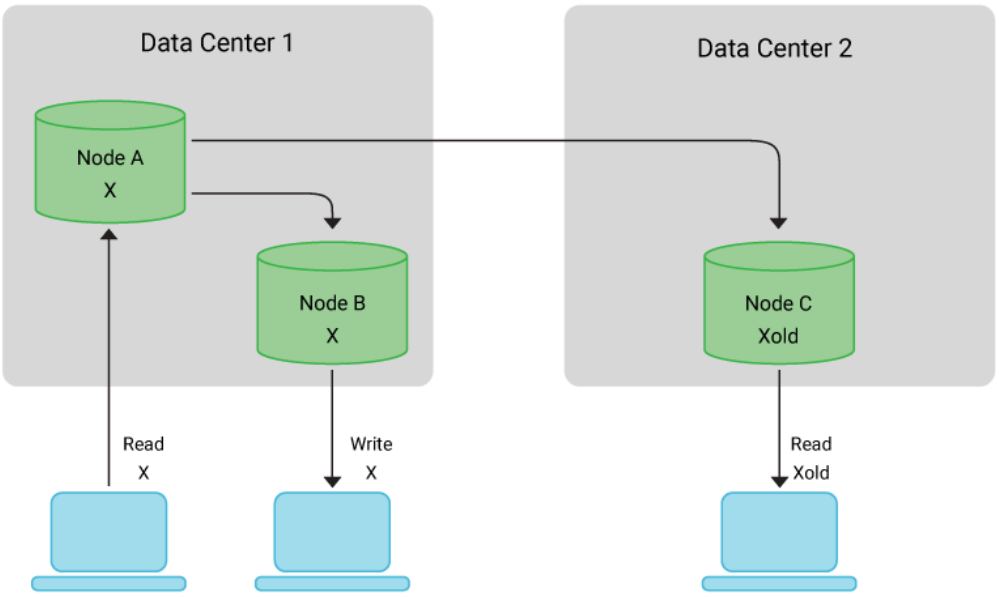


Рис. 3.8 Порушення узгодженості даних

Однак, якщо виявлено невідповідності, система забезпечує гнучкість обробки цих аномалій. Користувачі можуть або припинити копіювання, або продовжити процес, використовуючи налаштування відмовостійкості, що дозволяє пропускати неузгоджені дані. На рисунку 3.8 проілюстровано реплікацію даних які з'єднані між двома центрами обробки даних та їх невідповідність.

Узгодженість даних (Data Consistency) є фундаментальним критерієм якості функціонування інформаційної системи, який визначає логічну цілісність та ідентичність стану даних під час інтенсивного виконання конкурентних операцій читання та запису.

Ця характеристика гарантує, що в будь-який момент часу всі вузли системи та всі кінцеві користувачі отримують доступ до однакових, актуальних та валідних значень, незалежно від географічного розташування точки доступу, конкретного сервера обробки або часової послідовності ініціації запитів.

У контексті високонавантажених аналітичних платформ, забезпечення узгодженості стає особливо складним завданням через використання механізмів паралелізму та розподіленого кешування.

Система має підтримувати сувору детермінованість: якщо один процес завершив оновлення показника зваженої агрегації, будь-який наступний запит на читання від іншого користувача повинен миттєво відобразити це оновлення, уникаючи феномену «брудного читання» або застарілих даних.

Це досягається шляхом впровадження протоколів синхронізації та транзакційних моделей (таких як ACID або BASE залежно від архітектури), які регулюють порядок фіксації змін у сховищі.

Для оцінки факторів впливу на узгодженість даних використовуються наступні метрики:

Гонка даних (англ. Data Races).

Цілісність даних (англ. Data Integrity).

Атомарність операцій (англ. Operation Atomicity).

Сильна консистентність (англ. Strong Consistency).

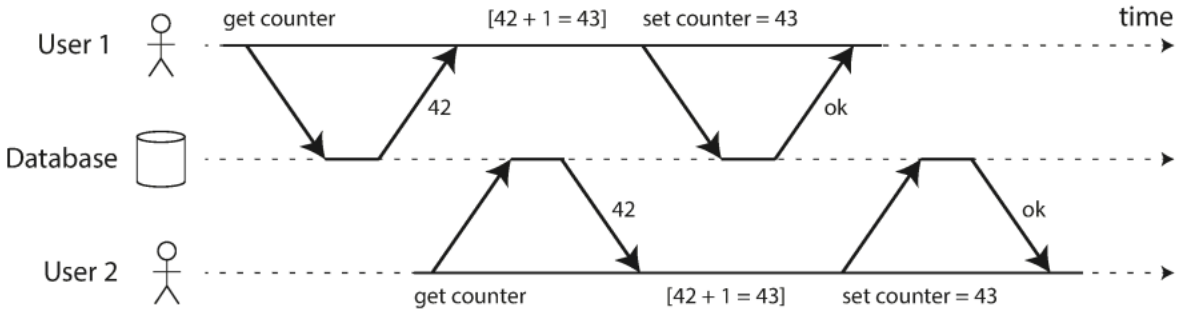


Рис. 3.9 Змагання між двома клієнтами, які одночасно збільшують значення лічильника

Гонка даних виникає, коли два або більше споживача здійснюють паралельний доступ до одного й того ж ресурсу, причому хоча б один із них виконує операцію запису [20, с. 226]. Наявність недостатньо ефективних

механізмів синхронізації, призначених для захисту спільного ресурсу, зазвичай призводить до виникнення неактуального стану даних або до небажаної поведінки чи непередбачених ситуацій у програмному забезпеченні.

Кожного разу, коли кілька завдань або потоків отримують доступ до спільного ресурсу без належного захисту одночасно, вони створюють невизначену або не передбачувану поведінку програми. Такі помилки можуть викликати в системі серйозні помилки, які важко знайти і виправити.

```
// Ініціалізація спільної змінної
shared Var ← 0
// Опис функції для збільшення спільної змінної
FUNCTION increment()
    FOR i FROM 1 TO 100,000 DO
        shared Var ← sharedVar + 1
    END FOR
END FUNCTION

// Виконання основної програми
BEGIN
    // Створення двох потоків
    thread1 ← CREATE_THREAD(increment)
    thread2 ← CREATE_THREAD(increment)
    // Очікування завершення потоків
    JOIN_THREAD(thread1)
    JOIN_THREAD(thread2)
    // Виведення результату
    PRINT "Final value of sharedVar: " + sharedVar
END
```

Рис. 3.10 Приклад гонки даних у програмі

Розглянемо простий приклад гонки даних у програмі представлена псевдокодом на рисунку 3.10. У цьому прикладі два потоки одночасно інкрементують спільну змінну “sharedVar” 100,000 разів кожен, щоб це значення в кінці повинно бути 200,000 але через те, що операції читання та запису виконуються некоректно без захисту, значення буде меншим.

Цілісність даних - це властивість, що забезпечує точність, узгодженість і повноту інформації протягом усього її життєвого циклу. Вона також гарантує, що дані залишаються незмінними та неушкодженими навіть у разі збоїв та відмов програмного забезпечення.

Атомарність операцій є одним із основних принципів управління транзакціями в нерозподілених базах даних і багатопотоковому програмуванні, забезпечує цілісність даних. Вона гарантує, що кожна операція виконується повністю або не виконується взагалі, не залишаючи дані в проміжному або некоректному стані.

Проте, як властивість ACID-транзакцій, якою вона є, не поводить себе повністю ортогонально щодо інших властивостей і, власне, узгодженість покладається на атомарність, очікуючи відкат в разі збоїв в ізоляції. ACID (Atomicity, Consistency, Isolation, Durability) - це сукупність властивостей (атомарність, узгодженість, ізольованість, довговічність), що забезпечують надійність обробки транзакцій у СУБД.

СУБД, що вимагають валідності даних відповідають наступним характеристикам щодо операцій:

1. Атомарність (Atomicity). Запобігає частковим змінам даних і забезпечує їхню узгодженість.
2. Узгодженість (Confidentiality). У разі неможливості коректного виконання операції відбувається відновлення стану системи до моменту початку операції.
3. Неподільність (Indivisibility). Операція не може бути поділена на окремі частини та розглядається як єдине ціле.
4. Цілісність (Integrity). Відмови або помилки під час виконання операції не впливають на загальний стан даних і не призводять до їх некоректного збереження, залишаються послідовними та коректними, навіть якщо кілька потоків намагаються їх змінити одночасно.

```
// Ініціалізація спільної змінної
sharedVar ← 0
// Ініціалізувати м'ютекс для синхронізації доступу до sharedVar
mutex ← CREATE_MUTEX()

//Опис функції для збільшення спільної змінної з синхронізацією
FUNCTION increment()
    FOR i FROM 1 TO 100,000 DO
        LOCK(mutex)
        sharedVar ← sharedVar + 1
        UNLOCK(mutex)
    END FOR
END FUNCTION

// Виконання основної програми
BEGIN
    // Створення двох потоків
    thread1 ← CREATE_THREAD(increment)
    thread2 ← CREATE_THREAD(increment)
    // Очікування завершення потоків
    JOIN_THREAD(thread1)
    JOIN_THREAD(thread2)
    // Виведення результату
    PRINT "Final value of sharedVar: " + sharedVar
END
```

Рис. 3.11 Приклад атомарності операцій

Узгодженість є також основною вимогою для розподілених СУБД які описуються CAP-теоремою, яка стверджує, що система розподілених даних може гарантувати лише дві з трьох властивостей:

1. Узгодженість (Consistency). Усі вузли системи бачать однакові дані у будь-який момент часу;
2. Доступність (Availability). Гарантія того, що кожен запит отримає коректну відповідь;
3. Стійкість до розділення (Partition Tolerance). Попри розділення на ізольовані секції або в разі втрати зв'язку з частиною вузлів, система не втрачає стабільність і здатність коректно відповідати на запити.

Теорема CAP описує фундаментальні компроміси між узгодженістю (consistency), яка гарантує, що операції читання враховують результати всіх попередніх записів, та доступністю (availability), за якої кожен запит повертає відповідь замість повідомлення про помилку.

Коректне формулювання теореми CAP стверджує, що в умовах виникнення мережевого розділення, розподілена система даних може забезпечувати або узгодженість, або доступність, але не обидві властивості

одночасно. У разі вибору на користь узгодженості (consistency) запит іноді повертатиме повідомлення про помилку замість відповіді. Якщо ж пріоритетом є доступність (availability), під час мережевого розділення операції читання можуть повертати застарілі результати. Найкращою властивістю узгодженості, яку можна реалізувати у високонадійній системі, є узгодженість у кінцевому підсумку (eventual consistency), за якої система повертається до цілісного стану після відновлення мережевого зв’язку.

Атомарність операцій є одним із методів, що застосовуються для запобігання гонкам даних. Інтерпретуючи її на рівні програмного коду, можна стверджувати, що певний блок коду виконується повністю та без переривань з боку інших потоків, забезпечуючи таким чином консистентний стан спільних ресурсів.

Протягом десятиліть транзакції залишаються основним механізмом для вирішення цих проблем. Транзакція - це спосіб об'єднання декількох операцій читання та запису у логічну одиницю на рівні застосунку. Концептуально всі операції читання та запису в межах транзакції виконуються як одна операція: транзакція або завершується успішно в цілому (commit), або зазнає невдачі (abort, rollback).

У разі невдачі застосунок може безпечно повторити спробу. Завдяки транзакціям обробка помилок значно спрощується, оскільки програмі не потрібно турбуватися про часткові збої - випадки, коли частина операцій виконується успішно, а частина завершується помилкою (з будь-яких причин). [20, с. 221-232].

Розглянувши приклад, наведений на рисунку 3.11, та застосувавши механізми синхронізації даних, було усунуто виявлену помилку. Псевдокод виправленої операції наведено на рисунку 3.12.

```
// Спільні змінні для кешу
sharedVar1 = 0
sharedVar2 = 0

// Мютекс для захисту спільних змінних
mutex ← CREATE_MUTEX()

// Функція для оновлення кешу
FUNCTION updateCache(val1, val2)
    acquire lock on mtx
    sharedVar1 = val1
    sharedVar2 = val2
    //  Необов'язково: Додати синхронізацію з іншими вузлами
    release lock on mtx
END FUNCTION

// Функція для читання з кешу
FUNCTION readCache()
    acquire lock on mtx
    print "sharedVar1:", sharedVar1, ", sharedVar2:", sharedVar2
    release lock on mtx
END FUNCTION

// Виконання основної програми
BEGIN
    // Створення потоків для оновлення кешу
    thread1 ← CREATE_THREAD(updateCache, 10, 20)
    thread2 ← CREATE_THREAD(updateCache, 30, 40)
    // Очікування завершення потоків
    JOIN_THREAD(thread1)
    JOIN_THREAD(thread2)


    // Читання з кешу
    readCache()

    return 0

END
```

Рис. 3.12 Приклад сильної консистентності

Сильна консистентність використовується для забезпечення можливості негайного зчитування оновлених даних після їх модифікації. Вона гарантує, що всі вузли системи бачать однаковий актуальний стан даних. Це є критично важливим показником для систем, у яких пріоритетом є точність та актуальність інформації.

ВИСНОВКИ

У даній роботі було досліджено методи та алгоритми оптимізації отримання даних за рахунок мінімізації обсягів інформації під час розрахунку складних аналітичних показників. Доведено, що інтеграція гетерогенних систем управління баз даних у поєднанні з диференціацією класів інформації, релевантних для формування аналітичної звітності, забезпечує суттєве підвищення продуктивності та швидкодії інформаційної системи в цілому.

Отримані результати виявляють як переваги, так і недоліки окремих систем баз даних. На їхній основі можна зробити висновок, що першочерговим інструментом оптимізації роботи з великими даними є сегментування даних аналітичної системи, класифікація цих сегментів та дотримання правил нормалізації баз даних відповідно до характеристик кожного сегмента.

Окрему увагу було приділено проблемам надлишковості та узгодженості даних. Запропоновано методи підтримки ключових оперативних показників (KPI) та забезпечення їх цілісності відповідно до вимог щодо даних, що дозволяє системам прийняття рішень отримувати своєчасну та актуальну аналітичну інформацію.

В експериментальній частині дослідження встановлено, що створення системи Big Data вимагає комплексного підходу до організації екосистеми взаємопов’язаних компонентів. Такий підхід забезпечує високу швидкість збору та паралельної обробки потоків даних, оптимальну пропускну здатність та мінімальні затримки при передачі і обробці інформації. Крім того, він гарантує масштабованість і надійність системи, що значною мірою визначається обраними інструментами та технологіями.

Результати тестування засвідчили найбільше зростання продуктивності для баз даних типу “ключ–значення”, а також суттєве прискорення аналітичних QL-запитів завдяки використанню кешування. Запропонований підхід є

найбільш доцільним для високонавантажених систем, у яких критичними є швидкість обробки та низька латентність.

Процеси з обробки даних, такі як, очищення даних від дублікатів, пропусків та аномалій, уніфікація форматів (дат, чисел, категорій), агрегація на відповідному рівні деталізації, перевірка узгодженості даних з різних джерел, оптимально виконувати на розподілених базах даних в напів-структурованому вигляді.

Результати роботи можуть бути використані при розробці високопродуктивних і високонавантажених систем, що потребують ефективного збору, обробки та аналізу великих обсягів даних.

Результати дослідження були представлені та апробовані на наукових конференціях:

1. Швець Р.П. ПОПЕРЕДНЯ ПІДГОТОВКА ДАНИХ В ЗАДАЧАХ ВІЗУАЛІЗАЦІЇ BIG DATA. Конференція: “Modern Perspectives on Global Scientific Solutions” Bergen, Norway, 21-30 грудня 2025 р. 180 ст.

2. Швець Р.П. ОПТИМІЗАЦІЯ ПОПЕРЕДНЬОЇ ПІДГОТОВКИ ДАНИХ В ЗАДАЧАХ ВІЗУАЛІЗАЦІЇ BIG DATA. ІІІ міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії». Київ, Україна, 08 грудня 2025 р. Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Spatiotemporal Data Analytics and Modeling / Satheesh Abimannan J. A. та ін. Singapore : Springer Nature, 2024. 420 p.

2. Reis J., Housley M. Fundamentals of Data Engineering. Sebastopol : O'Reilly Media, 2022. 448 p.

3. Wil van der Aalst. Process Mining: Data Science in Action. Springer. 2016. ISBN 978-3-662-49850-7.

4. Федірко О. Як і коли створювати сховища даних. 2025. URL: https://dou.ua/forums/topic/54249/ (дата звернення: 05.11.2025).

5. Rad, J. A., Chakraverty, S., & Parand, K. (Eds.). (2025). Dimensionality Reduction in Machine Learning. Elsevier/Morgan Kaufmann. ISBN 978-0-443-32818-3

6. Dimensionality Reduction in Data Science / M. Garzon, Ch.-Ch. Yang, D. Venugopal [et al.] ; eds. M. Garzon, Ch.-Ch. Yang, D. Venugopal, N. Kumar, K. Jana, L.-Y. Deng. – Cham : Springer, 2022. – XI, 265 p. – ISBN 978-3-031-05370-2.

7. Aggarwal C. C. Data Mining: The Textbook. Cham : Springer, 2015. 734 p.

8. Felix Gessert, Wolfram Wingerath, Norbert Ritter. Fast and Scalable Cloud Data Management. Cham : Springer, 2021. 320 p. ISBN 978-3-030-43505-9

9. Crickard P. Data Engineering with Python. Birmingham : Packt Publishing Ltd, 2020. 372 p.

10. Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning / M. T. Pilehvar, J. Camacho-Collados. – Cham : Springer, 2021. – XVIII, 157 p. – ISBN 978-3-031-01049-1

11. Data Warehousing and Analytics: Fueling the Data Engine / D. Taniar, W. Rahayu. – Cham (Switzerland) : Springer, 2021. – 635 p. – ISBN 978-3-030-81978-1

12. Business Statistics and Analytics in Practice / R. A. Hummel, G. W. Bowerman, L. D. Moninger [et al.]. – 9-е вид. – New York, NY : McGraw-Hill Education, 2024. – approx. 800 p. – ISBN 978-1-260-18749-6

13. Learning Spark / Karau H. та ін. Sebastopol : O'Reilly Media, 2015. 276 p.

14. Data Science for Business and Decision Making / L. P. Fávero, P. Belfiore. – Cham : Springer, 2022. – XII, 450 p. – ISBN 978-3-031-05500-3

15. Marz N., Warren J. Big Data: Principles and Best Practices of Scalable Real-Time Data Systems. New York : Manning Publications, 2015. 328 p.

16. Streaming Data: Understanding the Real-Time Pipeline / A. G. Psaltis. – 2-е вид. – Sebastopol, CA : O'Reilly Media, 2021. – 480 p. – ISBN 978-1-492-08756-1

17. Data Engineering for Business Analytics / A. Sathi. – 2-е вид. – Cham : Springer, 2022. – XII, 380 p. – ISBN 978-3-031-05672-7

18. Elmqvist N., Fekete J.-D. Hierarchical Aggregation for Information Visualization. 2015. 14 p. URL: https://www.cs.au.dk/~elm/pdf/hieragg.pdf (дата звернення: 05.11.2025).

19. Renaud Blanch, Eric Lecolinet. Browsing Zoomable Treemaps: Structure-Aware Multi-Scale Navigation Techniques. IEEE. 2007. p.1248 - 1253.

20. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

21. Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists / A. Zheng, A. Casari. – 2-е вид. – Sebastopol, CA : O'Reilly Media, 2022. – XIV, 380 p. – ISBN 978-1-492-08797-4

22. Borthakur D. HDFS Architecture Guide. 2015. 41 p. URL: https://homepages.cwi.nl/~boncz/lsde/papers/hdfs_design.pdf (дата звернення: 05.11.2025).

23. Vaibhav Verdhan. Data Without Labels: Practical Unsupervised Machine Learning. Manning Publications Co. 2025. 352 p.

24. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. New York : Springer, 2009. 745 p.

25. Han J., Kamber M., Pei J. Data Mining: Concepts and Techniques. Waltham : Morgan Kaufmann, 2012. 744 p.

26. Few S. Information Dashboard Design: Displaying Data for At-a-Glance Monitoring. Sebastopol : O'Reilly Media, 2013. 211 p.

27. Ivo D. Dinov. Data Science and Predictive Analytics: Biomedical and Health Applications using R. Cham : Springer, 2023. 832 p.

28. Multicomputer Multicore System Influence on Maximum Multi-Processes Execution Time / Zeebaree S. R. M. та ін. Test Engineering & Management, 2020. 14921-14931 p. URL: https://www.academia.edu/download/65634895/Multicomputer_Multicore_System_Influence_on_Maximum_Multi_Processes_Execution_Time.pdf (дата звернення: 05.11.2025).

29. Multi-Criteria Decision Analysis: Methods and Software / A. Ishizaka, P. Nemery. – 3-е вид. – Cham : Springer, 2021. – XX, 420 p. – ISBN 978-3-030-65418-1. с. 85–102.

30. The Analytic Hierarchy Process in Decision Making: Expert Systems Applications / T. L. Saaty, L. G. Vargas. – 3-е вид. – New York : Springer, 2021. – XII, 340 p. – ISBN 978-3-030-75012-3. – С. 55–78

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Оптимізація алгоритмів попередньої підготовки Big Data в задачах візуалізації бізнес-процесів»

Виконав: студент групи ПДМ-63 Руслан ШВЕЦЬ

Керівник: канд. фіз.-мат. наук, доцент, проф. кафедри ІПЗ, Володимир САДОВЕНКО

Київ - 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: оптимізація алгоритмів попередньої підготовки Big Data в задачах візуалізації бізнес-процесів.

Об’єкт дослідження: процеси попередньої підготовки даних у системах зберігання та обробки Big Data.

Предмет дослідження: алгоритми, методи та підходи попередньої підготовки Big Data, що застосовуються для візуалізації бізнес-процесів.

АКТУАЛЬНІСТЬ РОБОТИ

Зростання об‘ємів даних у світі за даними компанії IDC:

Рік	Кількість даних
2003	5 екзабайтів (1 ЕБ = 1 млрд ГБ)
2008	0,18 зетабайтів (1ЗБ = 1024 ЕБ)
2015	більше ніж 6,5 зетабайтів
2020	40-44 зетабайтів
2025 (прогноз)	400-440 зетабайтів

Актуальні проблеми візуалізації бізнес-процесів:

- швидкість доступу до даних;
- дані із різнорідних джерел, з різною структурою;
- масштабування алгоритмів попередньої обробки великих об’ємів даних.

3

ОБҐРУНТУВАННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ЩОДО БАЗОВОГО НАБОРУ ДАНИХ В SQL

Метод	Переваги	Недоліки
Оптимізація запитів, збережені процедури	Зменшення навантаження на сервер. Швидке виконання складних аналітичних запитів. Інкапсуляція логіки. Повторне використання.	Додаткове використання дискового простору. Складність тестування. Може ускладнити архітектуру. Складність у підтримці логіки.
Розділення даних	Прискорення запитів за рахунок зменшення об'єму. Підвищення продуктивності паралельної обробки. Зменшення витрат ресурсів.	Складність у проєктуванні. Складність підтримки актуальності.
Кешування	Значне прискорення доступу до даних. Зниження навантаження на основні системи. Висока пропускна здатність. Покращення масштабованості системи. Більше керування.	Непослідовність даних та складність їх узгодження. Необхідність додаткової пам’яті. Ускладнення архітектури. Додаткові витрати на підтримку інфраструктури та розробку ETL/ELT.
Заміна форматів зберігання	Швидке аналітичне читання. Краща компресія. Менше дискового простору. Прискорення агрегувань і сканування.	Повільніший запис. Не оптимально для OLTP. Складніша обробка швидких трансакційних змін. Потреба додаткової конвертації. Міграція даних.

4

МАТЕМАТИЧНА МОДЕЛЬ ЗВАЖЕНОГО АГРЕГАТУ

Зважений максимум - це агрегат **WMax** в якому зважуються елементи **x** за вагою **w** для визначення максимального елемента.

Нехай задано скінченну множину елементів

$$R = \{r_1, \dots, r_n\},$$

де кожному елементу відповідає пара

$$r_i = (x_i, w_i)$$

Визначимо функцію скаляризації, яка за необхідності може бути нормалізована наступним чином

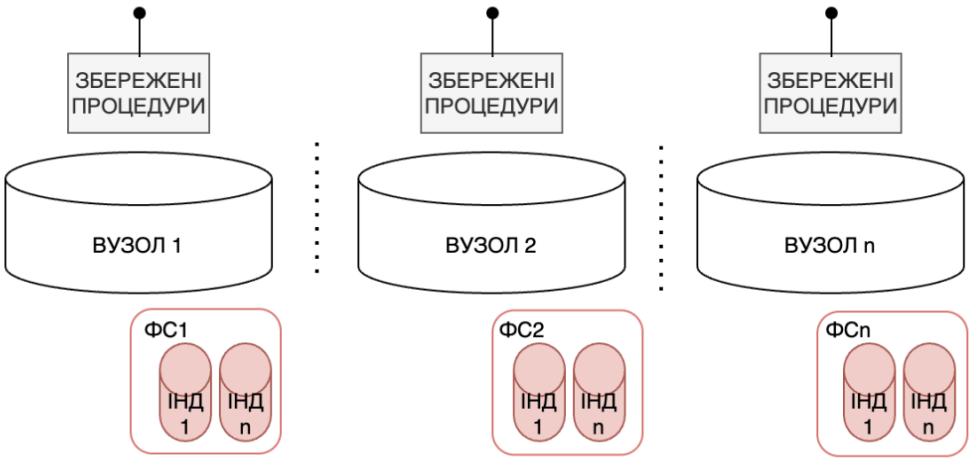
$$\hat{x} = \frac{x}{\sum_{i=0}^{n} x_i}, \quad s = f(x, w) = \hat{x} * w \quad (1)$$

Тоді зважений максимум визначається як

$$WMax(x, w) = \max_i(s_i) \quad (2)$$

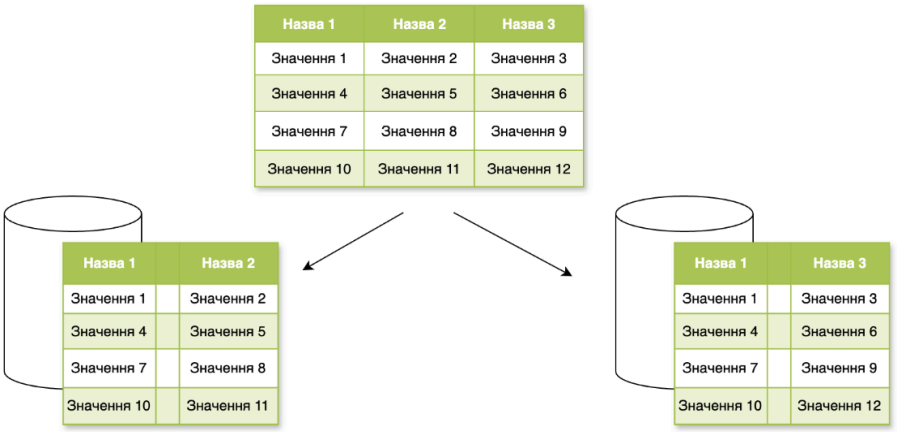
5

КІНЦЕВИЙ СТАН СИСТЕМИ ПІСЛЯ ВИКОНАННЯ ОПТИМІЗАЦІЇ ЗАПИТІВ І РЕАЛІЗАЦІЇ ЗБЕРЕЖЕНИХ ПРОЦЕДУР



6

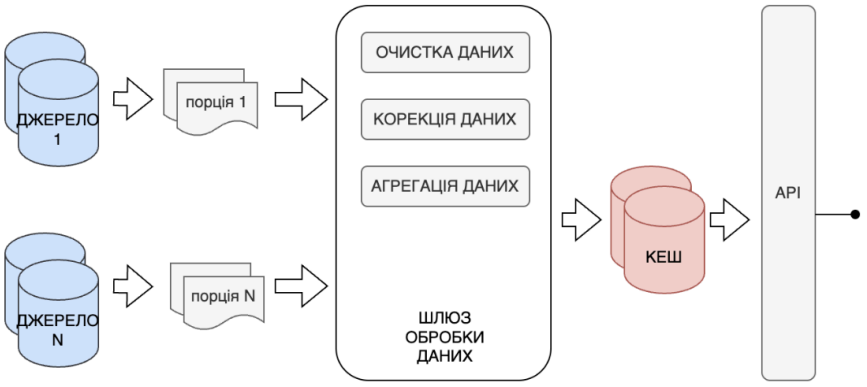
РОЗДІЛЕННЯ ДАНИХ



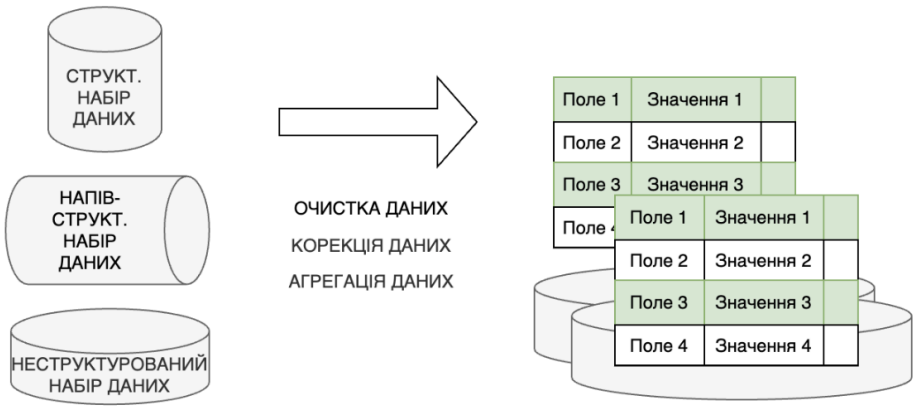
* стратегія розділення полягає у визначенні належності виміру до розподілу та/або денормалізації, згідно вимог щодо функціональностей системи

7

КЕШУВАННЯ УЗГОДЖЕНОГО НАБОРУ ДАНИХ



ЗМІНА ФОРМАТІВ ЗБЕРІГАННЯ



9

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ - ПОРІВНЯЛЬНИЙ АНАЛІЗ

для моделі зі 100 сутностей, 50 функціональностей та 1TB текстових даних

МЕТОД	РОЗРОБКА	ПІДТРИМКА	ПРОПУСКНА ЗДАТНІСТЬ
ОПТИМІЗАЦІЯ ЗАПИТІВ	50 індексів 50 процедур 100 сутностей ~20тиж: $125k+	128 vCPUs/512 GB/NVMe SSD 2Tb AWS: $3,000+/міс Support: $1000+/міс Cluster: ~$7k/міс DFS: $1,500+/міс	Читання: ~10k–30k QPS Запис: ~3k–7k TPS
РОЗДІЛЕННЯ ДАНИХ	75 індексів 130 сутностей ~35тиж: $500k+	128 vCPUs/512 GB/NVMe SSD 2Tb AWS: $3,000+/міс Support: $1000+/міс Cluster: ~$7k/міс DFS: $1,500+/міс	Читання: ~8k–25k QPS Запис: ~2k–8k TPS
КОЛОНКОВИЙ ВИГЛЯД	20 факт таблиць 60 таблиць вимірів 30 допоміжних ~30тиж: $500k ~10тиж: $150k міграція даних	60 vCPUs / 512 GB NVMe SSD 3 TB AWS: $5.5k/міс Support: $1k/міс Cluster: $5k/міс	Читання: 0.75-2 QPS Запис: 1-3 TPS
КЕШУВАННЯ	**30 сутностей** **60 індексів** **~25тиж: $300k+**	**144 vCPU/1,152 GB/3TB NVMe SSD** **AWS: $6,620/міс** **Support: $1000+/міс** **Cluster: ~$8k/міс**	**KV читання: 900k QPS** **KV запис: 450k TPS** **QL читання: 35k QPS** **Запис: 25k TPS**

10

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

1. Створена система попередньої обробки даних для Agencies у середовищі Python з використанням СУБД Couchbase;

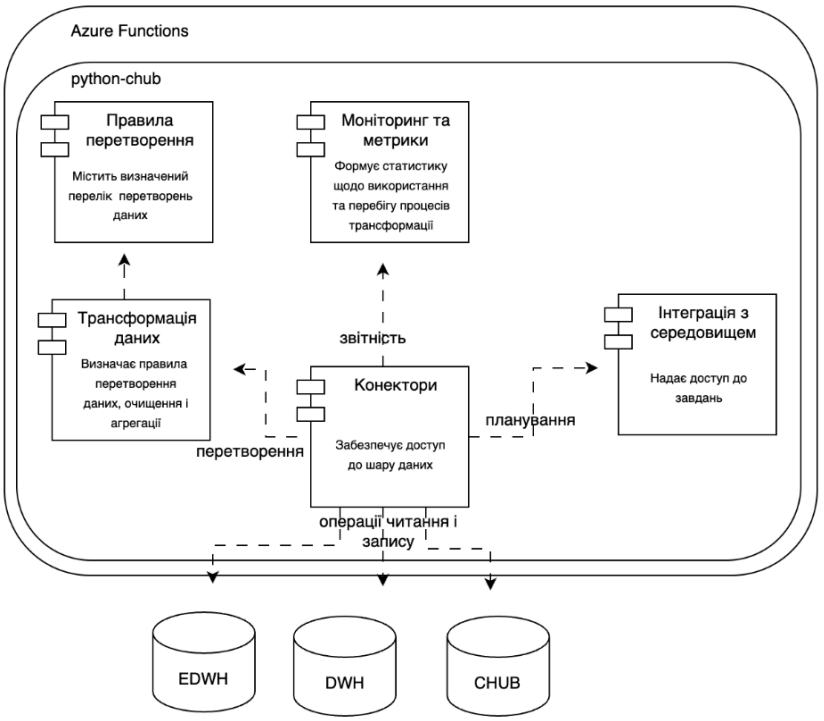
2. Кешування дало значний приріст обробки повторюваних запитів до словників даних і схожий результат при виконання аналітичних запитів;

3. Оскільки найчастіше використовувані агрегати, проміжні результати або трансформовані дані вже знаходяться в кеші, система не перераховує їх щоразу, що в свою чергу впливає на загальну продуктивність системи;

4. Час відповіді зменшується з секунд/хвилин до мілісекунд, надаючи системі високих характеристик оперативності.

10

СИСТЕМА ПОПЕРЕДНЬОЇ ОБРОБКИ ДАНИХ



11

ВИСНОВКИ

1. Проведено аналіз алгоритмів попередньої обробки даних Big Data та визначено можливості оптимізації.
2. Спроєктовано архітектуру системи та обрано метод зваженої агрегації для реалізації.
3. Розроблено програмну реалізацію системи попередньої обробки даних.
4. Тестування показало:
   - значне зростання продуктивності для баз типу «ключ–значення»,
   - прискорення аналітичних QL-запитів завдяки кешуванню.
5. Підхід може застосовуватися для автоматизації високонавантажених систем з критичною швидкістю обробки.
6. Результати представлені на наукових конференціях.
7. Подальші дослідження: масштабування методики на інші алгоритми попередньої підготовки даних.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Швець Р.П. ПОПЕРЕДНЯ ПІДГОТОВКА ДАНИХ В ЗАДАЧАХ ВІЗУАЛІЗАЦІЇ BIG DATA. Конференція: “Modern Perspectives on Global Scientific Solutions” Bergen, Norway, 21-30 грудня 2025 р. 180 ст.
2. Швець Р.П. ОПТИМІЗАЦІЯ ПОПЕРЕДНЬОЇ ПІДГОТОВКИ ДАНИХ В ЗАДАЧАХ ВІЗУАЛІЗАЦІЇ BIG DATA. III міжнародна науково-практична конференція «Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії». Київ, Україна, 08 грудня 2025 р. Подано до друку.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

requirement.txt

```
# Do not include azure-functions-worker in this file
# The Python Worker is managed by the Azure Functions platform
# Manually managing azure-functions-worker may cause unexpected issues

azure-functions
azure-identity
azure-keyvault-secrets
couchbase
marshmallow
numpy
oracledb
pyodbc
python-dateutil
pycountry
iso4217
```

requirements-dev.txt

```
-r requirements.txt
pre-commit
black
isort
ruff
dotenv
# mypy
coverage
```

host.json

```
{
 "version": "2.0",
 "logging": {
  "applicationInsights": {
   "samplingSettings": {
    "isEnabled": true,
    "excludedTypes": "Request"
   }
  },
  "fileLoggingMode": "always",
  "logLevel": {
   "default": "Information",
   "Function": "Information",
   "Host.Aggregator": "Trace",
   "Host.Results": "Trace"
  }
 },
 "extensionBundle": {
  "id": "Microsoft.Azure.Functions.ExtensionBundle",
  "version": "[4.*, 5.0.0)"
 },
 "functionTimeout": "02:00:00"
}
```

function_app.py

```
import azure.functions as func

from src import chub_entities, chub_executor
from src.logger import log,logger

app = func.FunctionApp()

logger.info("Loading chub main function")


@app.function_name(name="chub_delta_load_lead_age
ncies")
@app.schedule(
    schedule="0 0 * * * *",
    arg_name="myTimer",
    run_on_startup=False,
    use_monitor=False
)
@log
def chub_delta_load_lead_agencies(myTimer:
func.TimerRequest, context: func.Context) -> None:
    if myTimer is not None and myTimer.past_due:
        logger.info("The timer is past due!")

    logger.info(
        f"Running schedule from
{context.function_name}[{context.invocation_id}] "
        f"for timer {myTimer}"
    )


chub_executor.sync_data(chub_entities.LEAD_AGENC
Y, delta=True, context=context)
```

chub_field_mappings.py

```
sql_mappings = [
    # 9 - LEAD agency
    {
        "CD_TRAVEL_AGENCY_ID": ["EDWH",
"BPIdentificationNumber"], # PK
        "CD_AGENCY_VAT": ["EDWH",
"InputCD_AGENCY_VAT"],
        "DS_CITY": ["EDWH", "CityName"],
        "CD_CURRENCY_ANAG": [["EDWH",
"Currency"], ["EDWH", "PurchaseOrderCurrency"]],
        "DS_EMAIL": ["EDWH", "InputDS_EMAIL"],
        "DS_PEC": ["EDWH", "InputDS_PEC"],
        "DS_TELEX_NUMBER": ["EDWH",
"MobileNumber"],
        "DS_COMPANY_NAME": [["EDWH",
"OrganizationBPName1"], ["EDWH",
"OrganizationBPName2"]],
        "CD_ZIP": ["EDWH", "InputCD_ZIP"],
        "CD_OFFICE": ["EDWH", "InputCD_OFFICE"],
        "DS_FIRST_PHONE_NUMBER": ["EDWH",
"TelephoneNumber"],
        "FG_ACTIVE": ["EDWH", "BankStatus"],

        "CD_AGENCY_FISCAL_CODE": ["EDWH",
"InputCD_AGENCY_FISCAL_CODE"],
    },
]
```

chub_entities.py

```
import os
from dataclasses import dataclass
from typing import Union, Optional

LEAD_AGENCY = 9

@dataclass
class SqlConnectionConfiguration:
    server: str
    port: str
    database: str
    username: str
    provider: str

EDWH = SqlConnectionConfiguration(
    server =
os.environ.get("EDWH_SERVER_SERVER"),
    port = os.environ.get("EDWH_SERVER_PORT"),
    database =
os.environ.get("EDWH_SERVER_DATABASE"),
    username =
os.environ.get("EDWH_SERVER_USERNAME"),
    provider = PROVIDER.ORA,
)

@dataclass
class Entity:
    PK_SEPARATOR = "##"

    id: int
    table: str
    function: str
    scope: str
    collection: str
    last_update_column: str
    pk_column: Union[tuple, str]
    delta_only: bool
    retention_months: int
    db_config: SqlConnectionConfiguration
    traffic_light_queries: Optional[dict[str, str]] = None

@dataclass
class LeadAgency(Entity):
        self.traffic_light_queries = {
            "TDD_DM_TRAVEL_AGENCY": "select
max(data_rif) from chub_stg.wt_esito_caricamento
where table_name='TDD_DM_TRAVEL_AGENCY'
and flg_esito_caricamento='Y'"
        }

class EntityMap:
    BUCKET = "chub"

    def __init__(self):
        self.entities = [
            LeadAgency(LEAD_AGENCY,
"CHUB_OUT.VDD_TRAVEL_AGENCY_B2B_SAP_I
NTEGRATION", "", "b2b", "agencies",
"DT_INIT_VAL", ("CD_TRAVEL_AGENCY_ID",),
False, -1, EDWH),
        ]

ENTITY_MAP = EntityMap()
```

chub_executor.py

```
def sync_data(entity_id: int, delta=False, reverse=False, context=None):
    # Prevent some functions from running from env variables.
    ignored = None
    if not delta and "IGNORE_LOAD_FUNCTION" in os.environ:
        ignored = os.environ.get("IGNORE_LOAD_FUNCTION", "*")
    elif not reverse and "IGNORE_DELTA_FUNCTION" in os.environ:
        ignored = os.environ.get("IGNORE_DELTA_FUNCTION", "*")
    elif "IGNORE_DELTA_REVERSE_FUNCTION" in os.environ:
        ignored = os.environ.get("IGNORE_DELTA_REVERSE_FUNCTION", "*")

    if ignored is not None:
        ignored_list = ignored.split(',')
        ignore = ignored_list[0] == "*" or str(entity_id) in ignored_list

        if ignore:
            logger.info(f"[{ENTITY_MAP[entity_id]}] ignored from env variable")
            return

    ttt = time.time()
    get_environment_vars()
    if context is None:
        invocation_id = None
        function_name = None
    else:
        invocation_id = context.invocation_id
        function_name = context.function_name

    entity = ENTITY_MAP[entity_id]
    traffic_lights_warehouse = {}
    barrier_opened = True
    barriers = entity.traffic_light_queries

    if barriers:
        storage_result = chub_storage.get_value_attributes(entity_id, entity.TRAFFIC_LIGHT_PK, barriers.keys())
        traffic_lights_storage = {key: storage_result[index] for index, key in enumerate(barriers.keys())}

        for key, plain_query in barriers.items():
            traffic_lights_warehouse[key] = connectors[entity.db_config.provider].get_single_result(entity.db_config, plain_query)[0]

        for key in barriers.keys():
            wh_date: datetime = traffic_lights_warehouse.get(key)
            cb_date = traffic_lights_storage.get(key)

            if not cb_date:
                break
            elif not wh_date:
                logger.info(f"Traffic lights are red for [{entity}] warehouse:None")
                barrier_opened = False
                break
            elif not wh_date > datetime.strptime(cb_date, "%Y-%m-%dT%H:%M:%S"):
                logger.info(f"Traffic lights are red for [{entity}] warehouse:{wh_date.isoformat()} storage:{cb_date}")
                barrier_opened = False
                break

    if not barrier_opened:
        return

    try:
        chub_storage.connect(entity_id)

        start_date = None
        end_date = None
        if delta:
            if not reverse:
                start_date = chub_storage.get_most_recent(entity_id)
                logger.info(f"[{ENTITY_MAP[entity_id]}] Most recent date: {start_date}")
            else:
                start_date = chub_storage.get_least_recent(entity_id)
                logger.info(f"[{ENTITY_MAP[entity_id]}] Least recent date: {start_date}")

            # Delta without init should not happen.
            if start_date is None:
                logger.error(f"[{ENTITY_MAP[entity_id]}] Stopping processing as delta mode without most recent date")
                return


        else:
            # Fetch by start and end date if they are set.
            start_date_name = f"LOAD_START_DATE_{entity_id}"
            end_date_name = f"LOAD_END_DATE_{entity_id}"
```

```
        if start_date_name in os.environ and
os.environ.get(start_date_name) != "null":
            start_date = os.environ.get(start_date_name)
            logger.info(f"[{ENTITY_MAP[entity_id]}]
Init with start date: {start_date}")
            if end_date_name in os.environ and
os.environ.get(end_date_name) != "null":
                end_date = os.environ.get(end_date_name)
                logger.info(f"[{ENTITY_MAP[entity_id]}]
Init with end date: {end_date}")

        # Fetch only the items matching the retention
period on init sync.
        elif not ENTITY_MAP[entity_id].delta_only and
ENTITY_MAP[entity_id].retention_months > 0:
            # Approximation in days.
            retention_days =
ENTITY_MAP[entity_id].retention_months * 30
            current_date = datetime.now()
            oldest_date = current_date -
timedelta(days=retention_days)
            start_date = oldest_date.isoformat()
            logger.info(f"[{ENTITY_MAP[entity_id]}]
Oldest date: {start_date}")

    except Exception as e:
        logger.error(
            f"[{ENTITY_MAP[entity_id]}] Error connecting
to storage for {ENTITY_MAP[entity_id]}: {e}"
        )
        if invocation_id is not None:
            total_time = time.time() - ttt
            trace_data = {
                "invocation_id": invocation_id,
                "function": function_name,
                "status": FAILED_STATUS,
                "errors": e.__class__.__name__,
                "row_count": 0,
                "duration": int(total_time * 1000),
            }
            send_execution_trace(trace_data, entity_id)
        return

    all_fails, row_count, _ = process_sync(entity_id,
invocation_id, function_name, start_date, delta, reverse,
                                end_date=end_date)

    if barriers:
        set_value_attributes(entity_id,
ENTITY_MAP[entity_id].TRAFFIC_LIGHT_PK, {k:
v.isoformat() for k, v in
traffic_lights_warehouse.items()})

    # finalize running
    total_time = time.time() - ttt
    if invocation_id is not None:
        trace_data = {
            "invocation_id": invocation_id,
            "function": function_name,
            "row_count": row_count,
            "duration": int(total_time * 1000),
        }
        if all_fails:
            trace_data["status"] = FAILED_STATUS
            trace_data["errors"] = "|".join(all_fails)
        else:
            trace_data["status"] = SUCCESS_STATUS

        send_execution_trace(trace_data, entity_id)

    logger.info(
        f"[{ENTITY_MAP[entity_id]}] CHUB sync_data
executed in {timedelta(seconds=total_time)} from id:
{invocation_id} "
        f"inserting {row_count} rows for
{repr(ENTITY_MAP[entity_id])}"
    )

    recipient = os.environ.get("LEAD_SMTP_TO")
    if recipient and entity_id in
[chub_entities.LEAD_CUSTOMER,
chub_entities.LEAD_AGENCY] and
has_validation_errors():
        send_mail(
            Email(
                sender = 'no-reply@chub.com',
                recipient = recipient,
                subject = f"CHUB - LEAD: validation error
({'B2B Agencies' if entity_id ==
chub_entities.LEAD_AGENCY else 'B2C Customer' })",
                content = f"Some errors happened while
importing entities for {'B2B Agencies' if entity_id ==
chub_entities.LEAD_AGENCY else 'B2C Customer' }.
See details in the attachment",
                attachments =
[EmailAttachment(validation_errors_to_csv(), "csv",
"validation_errors.csv")],
            )
        )
```

```
def worker(queue, result_queue, entity, delta,
latest_update_date):
    total_worker = [0, 0]
    while True:
        try:
            rows, total = queue.get()
            if rows is not None and total > 0:
                total_worker[0] += 1
                total_worker[1] += len(rows)
                logger.info(
                    f"[{threading.current_thread().name}]
Writing {ENTITY_MAP[entity].collection} with {total}
Q: {queue.qsize()} T: {total_worker}"
                )

                transformed_rows = transform_rows(entity,
rows)

                latest_date = INITIAL_DATE
                for key in transformed_rows:
                    last_update =
ENTITY_MAP[entity].last_update_attribute(transformed
_rows[key])
                    latest_date = max(last_update, latest_date)
                latest_update_date.put(latest_date)

                fails = chub_storage.write_rows(entity,
transformed_rows, delta)
                if fails:
                    result_queue.put(fails)

                queue.task_done()
            else:
                queue.task_done()
                break
        except Exception as e:
            logger.exception("Worker error", e)
            break
```

transformers/storage_entities.py

```
def transform_rows(entity_id: int, rows) -> Dict[str,
Dict[str, Any]]:
    global mapping

    entities: List[Tuple[str, Dict[str, Any]]] = []

    for row in rows:
        data: Dict[str, Any] = {}

        # prep object with mapping
        for c_key, c_value in mapping[entity_id].items():
            if row[c_key] is None:
                continue

            final_value = convert_value(row[c_key])
            if isinstance(c_value, List):
                if len(c_value) > 0 and isinstance(c_value[0],
List):
                    for list in c_value:
                        map_values(list, data, final_value)
                else:
                    map_values(c_value, data, final_value)
            else:
                data[c_value] = final_value

        # Finish object prep.
        pk = ENTITY_MAP[entity_id].generate_pk(row)
        entities.append((pk, data))

    if entity_id == chub_entities.LEAD_AGENCY:
        entities = aggregate_agencies(entities)

    insert_bulk = {}
    for key, data in entities:
        update_row(entity_id, data)

        # Avoid resending the same agency multiple times a
day.
        if entity_id == chub_entities.LEAD_AGENCY and
'InputDT_INIT_VAL' in data['EDWH'] and
'InputPREVIOUS_DT_INIT_VAL' in
data['AzureFunctions'] and
data['EDWH']['InputDT_INIT_VAL'] ==
data['AzureFunctions']['InputPREVIOUS_DT_INIT_VA
L']:
            logger.info(f'Ignoring agency {key}')
            continue

        insert_bulk[key] = data

    return insert_bulk
```

```
def aggregate_agencies(agencies: List[Tuple[str, Dict[str,
Any]]]) -> List[Tuple[str, Dict[str, Any]]]:
    grouped_agencies = {}
    for key, agency in agencies:
        grouped_agencies[key] =
(grouped_agencies.get(key) or [])
        grouped_agencies[key].append(agency)

    result = []
    for key, agencies in grouped_agencies.items():
        root_agency = agencies[0]
        result.append((key, root_agency))

        root_agency["AzureFunctions"] =
root_agency.get("AzureFunctions", {})
        root_agency["AzureFunctions"]["BankDetails"] = []
        for agency in agencies:

            bank_details = {}
            copy_skipping_blank(agency, bank_details,
"BankAccount")
            copy_skipping_blank(agency, bank_details,
"BankAccountHolderName")
            copy_skipping_blank(agency, bank_details,
"BankAccountName")
            copy_skipping_blank(agency, bank_details,
"BankControlKey")
            copy_skipping_blank(agency, bank_details,
"BankCountryKey")
            copy_skipping_blank(agency, bank_details,
"BankdetailsIDinexternalsystem")
            copy_skipping_blank(agency, bank_details,
"BankIdentification")
            copy_skipping_blank(agency, bank_details,
"BankNumber")
            copy_skipping_blank(agency, bank_details,
"BankValidityEndDate")
            copy_skipping_blank(agency, bank_details,
"IBAN")
            copy_skipping_blank(agency, bank_details,
"InputID_HIST_TRAVEL_AGENCY")
            copy_skipping_blank(agency, bank_details,
"MandateCountry")
            copy_skipping_blank(agency, bank_details,
"MandateIBAN")
            copy_skipping_blank(agency, bank_details,
"MandateID")
            copy_skipping_blank(agency, bank_details,
"MandateValidityFrom")
            copy_skipping_blank(agency, bank_details,
"MandateValidityTo")
            copy_skipping_blank(agency, bank_details,
"BankStatus")

root_agency["AzureFunctions"]["BankDetails"].append(
bank_details)

    return result
```

```
def update_row(entity, data) -> None:
    global validation_errors

    if entity == chub_entities.LEAD_AGENCY or entity
== chub_entities.LEAD_CUSTOMER:
        del data["Last_Update"]
        data["AzureFunctions"] =
data.get("AzureFunctions", {})

        valid = True
        valid_vat = True
        errors = []
        if entity == chub_entities.LEAD_AGENCY:

            try:
                AgencySchema().load(data, partial=True)
                AgencySchema().load(data)
            except ValidationError as err:
                logger.error(f"Validation error(s) for {data}':
{str(err)}")
                valid = False
                if IS_NOT_UNIQUE in
str(err.messages.values()):
                    valid_vat = False
                errors.append(err.messages)

            try:
                dump = AgencySchema().dump(data)
                data['EDWH'] = dump['EDWH']
                data['AzureFunctions'] =
dump['AzureFunctions']
            except ValidationError as err:
                logger.error(f"Validation error(s) for {data}':
{str(err)}")
                valid = False

            data["AzureFunctions"]["VatOwner"] = valid_vat

        data["AzureFunctions"]["Valid"] = valid
```

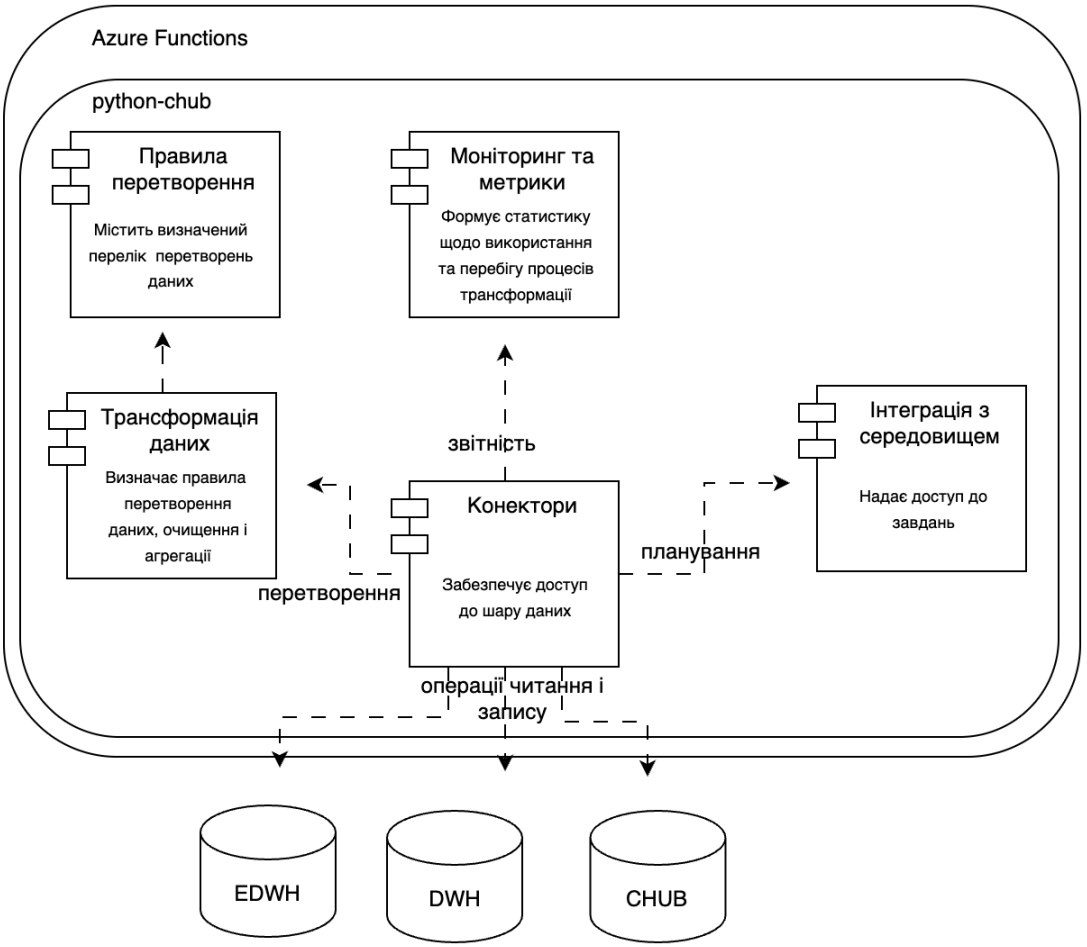
chub_storage_agency.py

```
vat_owners = {}
fiscal_owners = {}

class BankDetailsSchema(Schema):
   # omit for shortening
   class Meta:
      unknown = INCLUDE

class EdwhSchema(Schema):
   # omit for shortening
   class Meta:
      unknown = INCLUDE


class AzureFunctionsSchema(Schema):
   BankDetails =
fields.List(fields.Nested(BankDetailsSchema,
required=True))
   # omit for shortening
   class Meta:
      unknown = INCLUDE

class AgencySchema(Schema):
   EDWH = fields.Nested(EdwhSchema, required=True)
   AzureFunctions =
fields.Nested(AzureFunctionsSchema, required=False)

   @validates_schema
   def validate_vat_owner(self, data, **kwargs):
      global vat_owners, fiscal_owners

      entity_pk =
ENTITY_MAP[LEAD_AGENCY].extract_pk(data)

      vat = data['EDWH'].get('InputCD_AGENCY_VAT')
      if vat is not None and vat != NOVAT:
         check_uniqueness(entity_pk, vat_owners,
"AzureFunctions.BPTaxNumber", vat)

      fiscal =
data['EDWH'].get('InputCD_AGENCY_FISCAL_CODE
')
      if fiscal is not None and
is_germany(iso3_to_iso2(data["EDWH"].get("InputCD_
COUNTRY"))):
         check_uniqueness(entity_pk, fiscal_owners,
"EDWH.InputCD_AGENCY_FISCAL_CODE", fiscal)
```

```
  @pre_load
   def B_populate_azure_functions(self, data, **kwargs):

      ## BANK DETAILS BLOCK
      bank_details = azf.get("BankDetails")
      if is_canada(country) and bank_details:
         del azf["BankDetails"]
      elif bank_details:
         for i, details in enumerate(bank_details):
            country_iso2 =
iso3_to_iso2(details.get("BankCountryKey"))
            if country_iso2:
               details["BankCountryKey"] = country_iso2
            currency =
details.get('InputCD_BANK_CURRENCY_ANAG')
            if currency:
               details['BankIdentification'] = currency +
str(i)
         resolve_bank_account_name(data)
      ## BANK DETAILS BLOCK

      return data


  @pre_load
   def C_load_previous_state(self, data, **kwargs):
      key =
ENTITY_MAP[LEAD_AGENCY].extract_pk(data)

      commission, valid_from, valid_to, last_update =
chub_storage.get_value_attributes(LEAD_AGENCY,
key, [

"EDWH.InputCD_PAYMODEX_COMMISSION",
         "AzureFunctions.PaymodeXValidFrom",
         "AzureFunctions.PaymodeXValidTo",
         "EDWH.InputDT_INIT_VAL",
      ])


assign_not_empty(data[AF],"InputPREVIOUS_CD_PA
YMODEX_COMMISSION", commission)

assign_not_empty(data[AF],"InputPREVIOUS_Paymod
eXValidFrom", valid_from)

assign_not_empty(data[AF],"InputPREVIOUS_Paymod
eXValidTo", valid_to)

assign_not_empty(data[AF],"InputPREVIOUS_DT_INI
T_VAL", last_update)

      return data
```

```
def resolve_bank_account_name(data):
    bank_details =
data.get("AzureFunctions").get("BankDetails")
    for bank in bank_details:
        currency =
bank.get("InputCD_BANK_CURRENCY_ANAG")
        if currency:
            bank["BankAccountName"] = currency +
"_COM"

    bank_details.sort(reverse=True, key=bank_ranking)

    bank = bank_details[0]
    currency =
bank.get("InputCD_BANK_CURRENCY_ANAG")
    if currency:
        bank["BankAccountName"] = currency + "_COM"
+ "_00"

def check_uniqueness(pk, owners, path, value):
    cached_pk = owners.get(value)
    if cached_pk and cached_pk != pk:
        raise ValidationError(f"{value} " +
IS_NOT_UNIQUE, path)

    vat_owner_pk =
chub_storage.get_vat_owner_pk(LEAD_AGENCY, path,
value)
    if vat_owner_pk and vat_owner_pk != pk:
        raise ValidationError(f"{value} " +
IS_NOT_UNIQUE, path)

    owners[value] = pk
```

ДОДАТОК В. ДІАГРАМА СИСТЕМИ ОБРОБКИ ДАНИХ CHUB



EDWH, DWH, CHUB - джерела даних

Модуль “Конектори” - забезпечують підключення джерел даних

Модуль “Моніторинг” - здійснює знімання оперативних показників середовища виконання програмних модулів

Модуль “Трансформація” - містить стратегії застосування правил

Модуль “Правила перетворення” - здійснює фактичне перетворення даних згідно вимог

Модуль “Інтеграція з середовищем” - забезпечує інтеграцію з середовищем виконання

Azure Functions - це серверлес-сервіс Microsoft Azure для виконання окремих фрагментів коду