МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ЯРУНІВ Максим Андрійович

Програмний модуль розподіленого зберігання та паралельної обробки фінансових даних / Software Module for Distributed Storage and Parallel Processing of Financial Data

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-43
М.А. Ярунів

Науковий керівник:
д.т.н., професор М.П. Комар

Кваліфікаційну роботу допущено до захисту
«___»_____ 2026 р.

Завідувач кафедри
_____ Н.В. Дзюбановська

Тернопіль – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ЯРУНІВУ Максиму Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Програмний модуль розподіленого зберігання та паралельної обробки фінансових даних / Software Module for Distributed Storage and Parallel Processing of Financial Data
керівник роботи д.т.н., професор М. П. Комар
затверджені наказом по університету від 01 грудня 2025 року № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити
- проаналізувати особливості фінансових даних як об'єкта розподіленого зберігання та паралельної обробки;
- виконати огляд існуючих підходів до хмарної організації обчислень і розподіленого зберігання;
- дослідити принципи паралельної обробки даних і обґрунтувати застосування Spark для аналітичних задач;
- визначити методи забезпечення узгодженості, балансування та надійності зберігання даних у розподіленому середовищі;
- сформувати вимоги до захисту фінансових даних та окреслити криптографічні механізми і контроль доступу;
- розробити архітектуру програмного модуля, визначити його компоненти, інтерфейси та потоки даних;
- реалізувати прототип модуля та провести експериментальне оцінювання швидкодії й масштабованості.
5. Перелік графічного матеріалу в роботі
- загальна архітектурна схема програмного модуля: взаємодія рівнів та основних компонентів;
- схема логічного конвеєра Spark-обробки фінансових даних;
- схема контролю якості та оброблення пропусків у Spark-конвеєрі;

– схема середовища розгортання.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2026 р.	

Студент_____ М.А. Ярунів
підпис

Керівник роботи_____ д.т.н., професор М. П. Комар
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль розподіленого зберігання та паралельної обробки фінансових даних» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 70 сторінок і містить 10 ілюстрацій, 18 таблиць, 2 додатки та 31 використане джерело.

Метою роботи є розроблення програмного модуля розподіленого зберігання та паралельної обробки фінансових даних, який забезпечує масштабованість, відмовостійкість, підвищену продуктивність аналітичних операцій, а також базові механізми захисту та контролю якості даних.

Методи дослідження включають системний аналіз, аналіз фінансових даних, методи розподіленого зберігання даних, паралельну обробку великих даних, попереднє оброблення та нормалізацію фінансових наборів, контроль якості даних, проєктування програмної архітектури, моделювання, експериментальне тестування та оцінювання продуктивності обробки.

Розроблено програмний модуль розподіленого зберігання та паралельної обробки фінансових даних, який забезпечує завантаження фінансових наборів, їх валідацію, нормалізацію та очищення, розміщення даних у розподіленому сховищі HDFS, організацію оперативного доступу за ключем через HBase, запуск паралельних обчислювальних задач у Apache Spark, формування агрегованих показників і вітрин даних, контроль якості, оброблення пропусків, фіксацію метаданих запусків та надання результатів через API.

Результати дослідження можуть бути використані для побудови масштабованих систем фінансової аналітики, автоматизованого оброблення транзакційних даних, формування звітних вітрин, прискорення аналітичних розрахунків, контролю якості фінансових наборів, підтримки управлінських рішень та інтеграції з BI-системами.

Ключові слова: ФІНАНСОВІ ДАНІ, РОЗПОДІЛЕНЕ ЗБЕРІГАННЯ, ПАРАЛЕЛЬНА ОБРОБКА, ВЕЛИКІ ДАНІ, КОНТРОЛЬ ЯКОСТІ ДАНИХ.

ANNOTATION

Qualification work on the topic «Software Module for Distributed Storage and Parallel Processing of Financial Data» for Bachelor's degree on speciality 122 «Computer Science» educational and professional program «Computer Science» is written on 70 pages and it contains 10 figures, 18 tables, 2 annexes and 31 sources.

The purpose of the work is to develop a software module for distributed storage and parallel processing of financial data, which ensures scalability, fault tolerance, improved performance of analytical operations, as well as basic mechanisms for data protection and quality control.

The research methods include system analysis, analysis of financial data, methods of distributed data storage, parallel processing of big data, preprocessing and normalization of financial datasets, data quality control, software architecture design, modeling, experimental testing, and evaluation of processing performance.

A software module for distributed storage and parallel processing of financial data has been developed. It provides loading of financial datasets, their validation, normalization and cleaning, placement of data in the HDFS distributed storage system, organization of key-based operational access through HBase, execution of parallel computational tasks in Apache Spark, generation of aggregated indicators and data marts, quality control, missing value handling, recording of execution metadata, and provision of results through an API.

The research results can be used to build scalable financial analytics systems, automate the processing of transactional data, generate reporting data marts, accelerate analytical calculations, control the quality of financial datasets, support managerial decision-making, and integrate with BI systems.

Keywords: FINANCIAL DATA, DISTRIBUTED STORAGE, PARALLEL PROCESSING, BIG DATA, DATA QUALITY CONTROL.

ЗМІСТ

ВСТУП

Цифрова трансформація фінансового сектору супроводжується стрімким зростанням обсягів даних, які формуються під час виконання платіжних операцій, бухгалтерського та управлінського обліку, бюджетування, ризик-менеджменту, взаємодії з клієнтами та зовнішніми інформаційними ресурсами. У таких умовах зростає роль технологій Big Data у фінансах, оскільки саме вони забезпечують можливість отримання своєчасних аналітичних висновків на основі великих масивів гетерогенних даних [7]. Водночас упровадження аналітики великих даних у банківських і фінансових установах потребує узгоджених організаційних та технологічних рішень, зокрема щодо управління даними, регламентів доступу, якості та життєвого циклу наборів даних [2, 8].

Актуальність розроблення програмного модуля розподіленого зберігання та паралельної обробки фінансових даних зумовлена обмеженнями традиційних централізованих підходів. Зі збільшенням обсягу транзакцій і зростанням вимог до оперативності аналітики підвищується навантаження на ресурси зберігання, зростають затримки під час виконання складних запитів, ускладнюється масштабування системи та забезпечення відмовостійкості [11]. Розв’язання цих проблем пов’язується з переходом до хмарних моделей обчислень та використанням розподілених файлових систем і NoSQL-сховищ, що дозволяють горизонтально масштабувати інфраструктуру та забезпечувати високу доступність даних [13]. Практично значущими для цього класу задач є технології HDFS як розподілена файлова система для зберігання великих наборів даних і HBase як колоночне NoSQL-сховище для оперативного доступу до записів [3, 4, 18].

Важливим компонентом сучасних рішень є паралельна обробка, що зменшує час виконання аналітичних задач за рахунок розподілу навантаження між вузлами кластера. Зокрема, платформа Apache Spark забезпечує виконання обчислень за DAG-моделлю, підтримує кешування проміжних результатів у пам’яті та дає можливість реалізувати як пакетну, так і потокову обробку даних [22]. Для фінансових даних критичним є також забезпечення узгодженості та

надійності під час розподіленого зберігання. Це передбачає застосування механізмів балансування, реплікації та/або кодованого зберігання, а також протоколів узгодження стану в розподілених системах [1, 9, 10, 12].

Окремої уваги потребують питання безпеки, оскільки фінансові набори даних належать до категорії чутливих і підлягають захисту від несанкціонованого доступу та витоків. До типових заходів належать криптографічний захист, зокрема використання симетричного шифру AES, контроль доступу та аудит операцій [14]. Паралельно з цим ключовим фактором ефективності є якість даних: наявність пропусків, аномальних значень, дублювань та несумісності форматів прямо впливає на коректність фінансових розрахунків і результатів аналітики. Для зменшення негативного ефекту застосовуються методи очищення та відновлення пропущених значень, зокрема підходи імпутації для великих даних та методи машинного навчання [6, 19, 27]. Практичні аспекти побудови рішень розподіленого зберігання й паралельної обробки фінансових наборів даних у хмарному середовищі розглядаються в роботі, присвяченій відповідній технології на платформі хмарних обчислень [24].

Мета роботи полягає у розробленні програмного модуля розподіленого зберігання та паралельної обробки фінансових даних, який забезпечує масштабованість, відмовостійкість, підвищену продуктивність аналітичних операцій, а також базові механізми захисту та контролю якості даних.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

– проаналізувати особливості фінансових даних як об’єкта розподіленого зберігання та паралельної обробки;

– виконати огляд існуючих підходів до хмарної організації обчислень і розподіленого зберігання;

– дослідити принципи паралельної обробки даних і обґрунтувати застосування Spark для аналітичних задач;

– визначити методи забезпечення узгодженості, балансування та надійності зберігання даних у розподіленому середовищі;

– сформувати вимоги до захисту фінансових даних та окреслити

криптографічні механізми і контроль доступу;

– розробити архітектуру програмного модуля, визначити його компоненти, інтерфейси та потоки даних;

– реалізувати прототип модуля та провести експериментальне оцінювання швидкодії й масштабованості.

Об’єкт дослідження – процеси зберігання та обробки фінансових даних у хмарному та кластерному середовищі.

Предмет дослідження – методи та програмні засоби розподіленого зберігання і паралельної обробки фінансових даних, а також підходи до забезпечення узгодженості, захисту і якості даних.

Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах IV Всеукраїнської науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» (ІКСМ), м. Тернопіль, ЗУНУ, 20 травня 2026 р. (додаток Б).

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної області розподіленого зберігання та паралельної обробки фінансових даних

Фінансові дані як об’єкт інформаційних технологій характеризуються високою інтенсивністю надходження, значною різнорідністю форматів і підвищеними вимогами до достовірності та безпеки. У сучасних корпоративних системах дані формуються не лише в межах бухгалтерського обліку, а й у процесах управлінського обліку, бюджетування, казначейських операцій, кредитного аналізу, виявлення фінансових ризиків і комплаєнсу. У результаті виникають великі за обсягом масиви, які одночасно мають цінність для оперативних рішень і для довгострокової аналітики. Такий клас задач природно відноситься до напрямку Big Data у фінансах, де ключовою є здатність системи обробляти великі та гетерогенні набори даних із прийнятною затримкою, забезпечуючи коректність висновків і відтворюваність результатів [7].

1.1.1 Джерела та типи фінансових даних

У практичних системах доцільно розрізняти внутрішні та зовнішні джерела фінансових даних. До внутрішніх належать транзакційні журнали платіжних систем, дані ERP/CRM, бухгалтерські проведення, рахунки-фактури, дані управлінської звітності, реєстри договорів і контрагентів, результати інвентаризацій, дані внутрішнього контролю. До зовнішніх джерел належать курси валют, відсоткові ставки, макроекономічні індикатори, біржові котирування, дані відкритого банкінгу та інші регламентовані інформаційні потоки, що використовуються для уточнення ризикових моделей або збагачення аналітики [2, 8]. У контексті інституційного впровадження Big Data аналітики важливими є не лише технічні канали отримання, але й правила управління даними: опис походження даних, узгодження метаданих, визначення відповідальності, контроль доступу та відповідність політикам організації [2].

За структурою фінансові дані поділяються на:

– структуровані (табличні записи транзакцій, залишки на рахунках,

балансові показники, довідники контрагентів);

– напівструктуровані (JSON/XML-повідомлення інтеграційних шлюзів, журнали подій сервісів, повідомлення брокерів черг);

– неструктуровані (скан-копії первинних документів, текстові пояснення, листування, звіти у форматі PDF тощо).

Зазначений поділ важливий для вибору підходу до зберігання та способів обробки. Наприклад, транзакційні дані часто потребують швидкого доступу за ключем (ідентифікатор рахунку, контрагента, договору) у поєднанні з часовими обмеженнями, тоді як історичні масиви для аналітики доцільно зберігати у форматах, оптимізованих під сканування й агрегування [11]. Окремий клас становлять дані фінансової звітності, для яких критичною є цілісність показників і їх узгодженість між формами звітів; водночас на практиці у великих масивах зустрічаються пропуски, нерівномірність заповнення та неоднорідність у правилах формування показників, що потребує процедур підготовки та відновлення даних [6].

З погляду технологічного контуру обробки доцільно додатково класифікувати дані за частотою надходження:

– потокові – транзакції, події авторизації, лог-події, телеметрія;

– пакетні – денні/тижневі/місячні вивантаження, консолідована звітність, архіви.

Вибір режиму безпосередньо впливає на проєктні рішення: потокові дані вимагають механізмів швидкого прийому та первинної валідації, пакетні – зручного повторного прогону та відтворюваності обчислень під час аудиту [22].

У таблиці 1.1 подано класифікацію фінансових даних та узагальнені вимоги до їх оброблення, що використовується під час формування архітектурних рішень для зберігання і паралельної обробки [7, 11].

Таблиця 1.1 – Класифікація фінансових даних та вимоги до їх оброблення

Тип даних	Приклади	Частота надходження	Критичність коректності	Типові вимоги до обробки
Транзакційні	платежі, проводки, списання/зарахування	висока	дуже висока	низька затримка, трасованість, контроль дублікатів [11]
Довідникові	контрагенти, рахунки, договори	середня	висока	узгодженість ключів, контроль версій [2]
Звітні	баланси, P&L, cash-flow	Низька / періодична	дуже висока	повнота, контроль пропусків, відтворюваність розрахунків [6, 7]
Подієві/логові	події сервісів, журнали доступу	висока	середня	агрегації, кореляція подій, зберігання великих обсягів [11]
Зовнішні ринкові	курси, ставки, індекси	Середня / висока	висока	валідація джерела, часові зрізи, версіонування [8]

1.1.2 Характеристики фінансових даних як Big Data та ключові виклики

У фінансових системах одночасно проявляються класичні ознаки Big Data: обсяг (volume), швидкість (velocity), різноманітність (variety) і достовірність (veracity). Висока швидкість надходження даних формує вимогу до оперативної обробки: затримки у побудові агрегацій або виявленні аномалій можуть призводити до втрати актуальності управлінських рішень. Різноманітність форматів ускладнює уніфікацію схеми та потребує гнучких механізмів перетворення й нормалізації. Достовірність фінансових даних має особливий статус, оскільки похибка або некоректні значення можуть спричинити неправильні управлінські висновки або порушення регламентів [7].

Практика впровадження аналітики великих даних у банкінгу та фінансових сервісах вказує на типові бар’єри: складність інтеграції джерел, нестачу узгоджених процесів управління даними, нерівномірну якість наборів даних та проблеми масштабування інфраструктури при зростанні навантаження [8]. З позиції організаційної готовності помітну роль відіграє культура роботи з даними та наявність політик управління даними, що узгоджують технічні рішення із процедурними вимогами установи [2]. Отже, предметна область даної роботи включає не лише програмні компоненти, а й вимоги до регламентованості процесів обробки.

Суттєвий виклик пов’язаний із тим, що фінансові дані часто повинні бути доступні у двох «вимірах»:

– операційному (швидкий доступ до конкретного рахунку/операції/періоду);

– аналітичному (масштабне сканування історичних масивів, побудова агрегатів, формування звітів, пошук закономірностей).

Зазначене протиріччя обумовлює потребу у поєднанні різних моделей зберігання: файлового «озера даних» для історичних наборів і таблично-ключового доступу для оперативних сценаріїв [11]. У межах розподіленого підходу саме розмежування «гарячих» і «холодних» даних дозволяє раціонально розподіляти ресурси й уникати перевантаження централізованої СУБД при масових аналітичних запитах.

Для обґрунтування вибору технологій зберігання та обробки фінансових даних доцільно виконати їх порівняння за критеріями масштабування, затримки доступу, типових сценаріїв використання та обмежень. У таблиці 1.2 подано узагальнене порівняння HDFS, HBase, централізованої реляційної БД та платформи Apache Spark [3, 4, 11, 18, 22].

Таблиця 1.2 – Порівняння технологій зберігання і обробки

Технологія	Призначення	Масштабування	Типова затримка доступу/обробки	Типові сценарії використання	Основні обмеження
1	2	3	4	5	6
Централізована реляційна БД (RDBMS)	Транзакційне зберігання структурованих даних із підтримкою ACID	Переважно вертикальне; горизонтальне можливе, але ускладнює адміністрування та узгодженість	Низька для точкових транзакцій і коротких запитів; зростає для великих сканувань і складних агрегацій	OLTP-операції, бухгалтерські реєстри, довідники, регламентні звіти на помірних обсягах [11]	Обмеження вертикального масштабування, висока вартість нарощування ресурсів, конкуренція OLTP та аналітики, зниження продуктивності на великих масивах [11]

Продовження таблиці 1.2

1	2	3	4	5	6
HDFS	Розподілене файлове зберігання великих наборів даних блоками з реплікацією	Горизонтальне (додавання вузлів кластера) [3, 18]	Оптимізовано під високий throughput при послідовному читанні; не призначено для низьколатентного доступу до окремих записів [3, 18]	Data lake, зберігання історичних фінансових наборів, архіви транзакцій, файли у Parquet/ORC для подальшої аналітики [3, 18]	Висока затримка для випадкового доступу, відсутність зручної моделі “read/write by key” без надбудов, складність дрібних оновлень файлів [3, 18]
HBase	Колонкове NoSQL-сховище поверх HDFS для доступу за ключем (wide tables)	Горизонтальне; масштабування за рахунок регіонів та розподілу даних [4]	Низька/середня для операцій читання/запису за ключем порівняно з файловим шаром; залежить від моделі ключа та розподілу навантаження [4, 9]	Оперативні вибірки транзакцій/показників за рахунком і періодом, зберігання “гарячих” даних, швидкі довідки та індексація під сервіси [4]	Вимогливість до проєктування rowkey (ризик hot-spot), складність складних SQL-агрегацій без додаткових інструментів, потреба тюнінгу кластера [4, 9, 11]
Apache Spark	Платформа паралельної обробки даних у кластері (batch/streaming) з DAG-плануванням	Горизонтальне за рахунок збільшення кількості виконавців і ресурсів кластера [22]	Низька/середня для повторних обчислень при кешуванні в пам’яті; для великих job-ів домінують I/O і мережеві витрати [22]	Масові агрегації фінансових транзакцій, побудова звітів, підготовка даних для моделей, перевірки якості, інтеграція з HDFS/HBase [22]	Не є сховищем; потребує зовнішнього storage, ефективність залежить від партиціювання й налаштувань, вимагає контролю ресурсів і спостережуваності [22, 11]

Наведене порівняння демонструє, що задачі довготривалого зберігання великих масивів доцільно реалізовувати на файловому шарі, тоді як оперативний доступ за ключем забезпечується NoSQL-сховищем, а обчислювально складні аналітичні процедури – паралельною обробкою у кластері [3, 4, 22].

1.1.3 Вимоги до зберігання: масштабованість, доступність, узгодженість, відмовостійкість

Традиційні централізовані сховища ефективні для транзакційних навантажень у межах визначеного масштабу, однак за зростання обсягів і кількості аналітичних запитів виникає проблема вертикального масштабування, зростання витрат і погіршення продуктивності під час складних агрегацій [11]. Для подолання цих обмежень застосовуються розподілені системи зберігання та обробки, що передбачають горизонтальне масштабування кластера.

Використання хмарної інфраструктури в цьому контексті розглядається як базове середовище розгортання, оскільки хмарна модель забезпечує еластичність ресурсів, стандартизовані механізми керування та можливість швидкого нарощування обчислювальних потужностей [13]. Відповідно до поширеного визначення хмарних обчислень, ресурс надається як сервіс за моделлю «на вимогу», з широким мережевим доступом і пулом спільних ресурсів [13]. Для фінансових даних така модель є доцільною за умови, що додатково забезпечуються вимоги безпеки та контроль доступу.

У сучасних рішеннях для зберігання великих обсягів даних важливе місце посідає Hadoop Distributed File System (HDFS), яка передбачає зберігання даних у вигляді блоків із реплікацією між вузлами кластера та оптимізована під послідовне читання великих файлів [18]. Архітектурні принципи HDFS і підхід до організації блоків і реплік відображені в офіційній документації та описі архітектури [3, 18]. Для сценаріїв, де потрібен швидкий доступ до записів за ключем, доцільним є застосування HBase – колоночного NoSQL-сховища поверх HDFS, яке підтримує операції читання/запису за ключем і зручне для зберігання широких таблиць із великим числом колонок [4]. Таким чином, у предметній області даної роботи розподілене зберігання розглядається як поєднання файлового шару для історичних масивів і ключового шару для оперативних вибірок.

Окремий блок вимог стосується балансування навантаження та розміщення даних. У розподіленому середовищі нерівномірний розподіл «гарячих» ключів може створювати вузькі місця та знижувати продуктивність.

Як базова ідея для рівномірного розміщення ключів у розподілених системах широко застосовується consistent hashing, який зменшує обсяг перерозподілу під час зміни кількості вузлів [9]. Питання балансування з урахуванням надлишковості даних та реплікації досліджуються також у контексті оцінювання продуктивності розподілених сховищ [1]. Для підвищення ефективності використання простору та збереження надійності застосовуються схеми кодованого зберігання (наприклад, конвертовані коди), що дозволяють змінювати параметри кодування при модифікації умов зберігання [12]. Зазначені підходи формують теоретичне підґрунтя вибору механізмів надійності й масштабованості.

Узгодженість стану даних у розподілених системах є критичною, особливо для фінансових операцій, де важлива коректність часових зрізів і відсутність суперечливих оновлень. Загальні принципи досягнення консенсусу в розподілених системах описуються через протоколи на кшталт Paxos, які формують основу для побудови узгоджених сервісів у середовищі з відмовами [10]. Хоча конкретні механізми в інструментах Hadoop-екосистеми реалізуються на прикладному рівні, саме розуміння проблеми консенсусу та компромісу між узгодженістю, доступністю і стійкістю до розділення мережі є важливим для постановки задачі та обґрунтування архітектури.

1.1.4 Вимоги до паралельної обробки та типові сценарії аналітики

Паралельна обробка у фінансовій сфері застосовується для задач, що обчислювально є важкими або потребують аналізу великих історичних масивів: агрегування транзакцій за періодами, формування звітів, розрахунок показників ліквідності, пошук аномалій, підготовка даних для моделей ризику. Під час виконання таких задач у централізованих системах часто виникає проблема тривалого часу виконання та конкуренції ресурсів між транзакційними та аналітичними навантаженнями [11]. У зв’язку з цим актуальним є використання обчислювальних платформ, що підтримують розподіл задач на підзадачі та виконання у кластері.

Одним із базових інструментів для паралельної обробки є Apache Spark,

який орієнтований на виконання обчислень із використанням робочих наборів даних (working sets) та підтримує кешування проміжних результатів у пам’яті, що зменшує витрати на введення-виведення [22]. Для предметної області фінансових даних це дозволяє прискорювати типові агрегації, повторні запуски обчислень і побудову багатокрокових конвеєрів підготовки даних. Технологічно Spark зручний тим, що підтримує інтеграцію з розподіленими сховищами та використовується як універсальний виконавчий шар для batch- і streaming-сценаріїв [22]. У межах бакалаврської роботи зазначений інструментарій розглядається як основа реалізації паралельних обчислювальних процедур.

З практичної точки зору важливо враховувати, що успіх впровадження аналітики великих даних визначається не лише вибором платформи, але й узгодженістю проєктних рішень із процесами організації: від підготовки даних до процедур експлуатації [8]. Тому предметна область включає вимоги до спостережуваності (моніторинг стану вузлів, журналювання, оцінка використання ресурсів), а також до можливості повторного виконання задач із контрольованим результатом.

Для узагальнення технологічного процесу роботи з фінансовими даними доцільно подати послідовність етапів від надходження інформації до отримання аналітичного результату. На рисунку 1.1 представлено узагальнену схему технологічного процесу, що відображає типову логіку побудови систем: приймання й валідація даних, розподілене зберігання, паралельна обробка та надання результатів через прикладні сервіси [3, 4, 11, 18, 22].

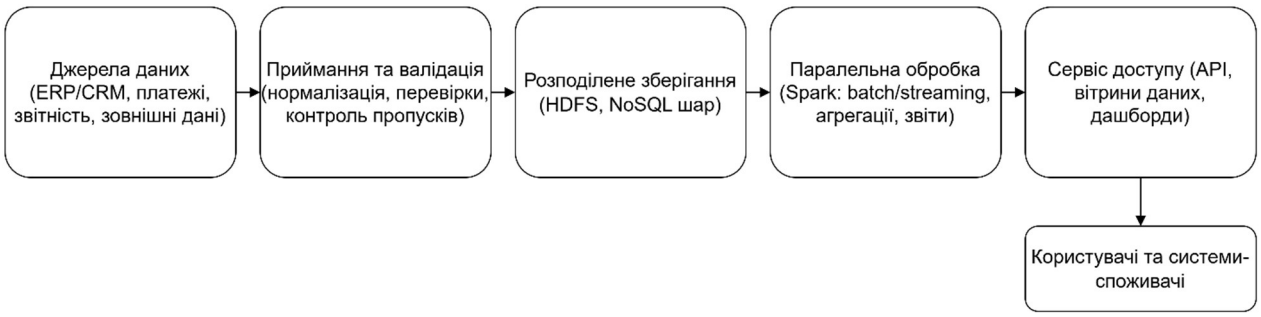

Рисунок 1.1 – Узагальнена схема технологічного процесу обробки фінансових даних у розподіленому середовищі

Для систематизації вимог, що визначають архітектуру програмного модуля, доцільно узагальнити ключові напрями: масштабованість, продуктивність паралельної обробки, забезпечення узгодженості та відмовостійкості, захист і контроль доступу, а також контур якості даних. На рисунку 1.2 подано карту проблем і вимог, яка буде використовуватися як основа для формування функціональних і нефункціональних вимог у підрозділі 1.3 [1, 3, 4, 10–12, 14, 18, 22].

Рисунок 1.2 – Карта проблем та вимог до програмного модуля розподіленого зберігання і паралельної обробки фінансових даних

1.2 Аналіз існуючих підходів та технологій

Розроблення програмного модуля розподіленого зберігання та паралельної обробки фінансових даних потребує узгодженого вибору технологій з урахуванням вимог до продуктивності, масштабованості, надійності, узгодженості й безпеки. У сучасних дослідженнях і прикладних роботах простежується спільна тенденція: традиційні централізовані сховища та послідовні обчислення поступово доповнюються або замінюються розподіленими підходами, які дозволяють горизонтально нарощувати ресурси та скорочувати час виконання аналітичних задач [11, 24]. При цьому ефективність

рішень визначається не лише вибором окремого інструмента, а цілою архітектурною композицією: шар зберігання, механізми розміщення/балансування даних, обчислювальний фреймворк, контур безпеки та якісні характеристики даних [11, 24].

1.2.1 Хмарні моделі обчислень як середовище для фінансової аналітики

Хмарні обчислення розглядаються як базове середовище розгортання систем Big Data завдяки еластичності, масштабованості та можливості надання ресурсів “на вимогу” [13]. Для фінансових задач ця модель є привабливою, оскільки навантаження часто має нерівномірний характер (піки в кінці періодів, під час масових звірок і консолідацій), а швидке виділення обчислювальних ресурсів дозволяє скорочувати час регламентних процедур. Разом із цим хмарний контекст посилює вимоги до контролю доступу, сегментації середовища, криптографічного захисту та аудиту дій, оскільки дані та сервіси розміщуються у багатокомпонентній інфраструктурі [13, 14].

У фінансовому секторі важливу роль відіграють організаційні фактори впровадження аналітики великих даних: готовність процесів управління даними, наявність політик data governance, узгодженість метаданих і відповідальності за якість наборів даних [2, 8]. Дослідження зосереджують увагу на тому, що технологічні інвестиції без належного управління та культури роботи з даними не гарантують стійкого ефекту від впровадження Big Data аналітики, зокрема у банківських і фінансових сервісах [2, 8]. Отже, аналіз підходів до зберігання й обробки має враховувати не лише технічні характеристики, але й експлуатаційні аспекти: відтворюваність розрахунків, прозорість конвеєрів даних, спостережуваність і керованість.

1.2.2 Розподілене зберігання: прикладні сценарії та типові проблеми

У прикладних роботах, узагальнених у статті-основі, розподілені сховища розглядаються в різних доменах: мобільні розподілені системи, просторові дані, моніторингові набори та спеціалізовані сценарії з підвищеними вимогами до надійності [24]. У мобільних середовищах акцент робиться на проблемі

адаптивного розміщення даних з урахуванням параметрів вузлів і мережевих характеристик; пропонуються стратегії, що враховують пропускну здатність та інші показники для підвищення успішності передавання й загальної ефективності [20]. У задачах просторової аналітики увага концентрується на порівнянні розподілених рушіїв зберігання/обробки та оцінюванні продуктивності під час виконання великих обчислювальних запитів, що є показовим з погляду кластерної обробки масивів [17]. Для доменів спостережень і моніторингу пропонуються моделі побудови розподілених сховищ із фокусом на сервісній ефективності та підтримці довготривалого архівування [21].

Незважаючи на різницю прикладних областей, повторюється спільний набір проблем: нерівномірний розподіл “гарячих” даних, необхідність балансування навантаження, підтримка відмовостійкості та мінімізація витрат на зберігання. Узагальнений практичний огляд інженерних підходів до проєктування розподілених сховищ підкреслює, що надійність і продуктивність досягаються комбінацією правильної моделі даних, продуманого партиціювання та коректних механізмів реплікації/відновлення [11]. У контексті продуктивності важливою є проблема балансування під час наявності надлишковості, оскільки реплікація або інші форми резервування змінюють профіль навантаження і можуть призводити до “перекосів” у використанні вузлів [1].

Розвиток напрямку зменшення витрат на зберігання при збереженні надійності пов’язується з кодованим зберіганням (erasure coding) та суміжними методами. Зокрема, досліджуються коди, що дозволяють ефективно змінювати параметри кодування, не виконуючи повного перезапису даних у розподіленому середовищі, що важливо для адаптації системи до росту або зміни профілю доступу [12]. У сценаріях із підвищеними вимогами до стійкості та ресурсної ефективності (наприклад, розподілені групи безпілотних систем) акцент робиться на процедурі проєктування сховища та процесі організації зберігання з урахуванням обмежень домену та вимог до надійності [15]. Наведені результати демонструють, що розподілене зберігання є універсальним підходом, однак конкретна ефективність залежить від того, як виконано розміщення даних, організовано резервування та забезпечено керування “гарячими” ключами [12].

1.2.3 Технологічна база HDFS/HBase: можливості та обмеження

На практиці поширеним базисом для зберігання великих наборів даних у кластері є екосистема Hadoop. HDFS орієнтована на зберігання великих файлів із блочним розміщенням і реплікацією, що забезпечує високу пропускну здатність при послідовному читанні й записі, а також відмовостійкість за рахунок дублювання блоків [3, 18]. Водночас HDFS не призначена для сценаріїв із низькою затримкою доступу до окремого запису, тому для оперативних запитів “за ключем” використовуються надбудови.

HBase належить до колоночних NoSQL-сховищ поверх HDFS і забезпечує модель доступу за ключем до широких таблиць, що дозволяє реалізувати оперативні вибірки транзакцій або показників за рахунком/контрагентом/періодом [4]. Для фінансових даних це створює підґрунтя поєднання двох контурів: файлового шару (архіви, історичні масиви, “холодні” дані) і ключового шару для “гарячих” вибірок (оперативні зрізи, індексація, швидкий пошук) [3, 4, 18]. Разом із цим HBase висуває підвищені вимоги до проєктування ключів (rowkey) та до балансування розподілу навантаження, оскільки невдалий дизайн може спричиняти концентрацію запитів на обмеженій кількості регіонів і деградацію продуктивності [4, 11].

1.2.4 Узгодженість, розміщення та балансування в розподілених системах

Питання узгодженості (consistency) у розподілених системах є критичним для фінансового домену, оскільки будь-які суперечливі оновлення або некоректні часові зрізи впливають на правильність звітів і управлінських рішень. Теоретичне підґрунтя для побудови узгоджених сервісів у середовищі з відмовами формують протоколи консенсусу, зокрема Paxos [10]. На прикладному рівні ці ідеї відображаються в підходах до координації компонентів, лідерства, журналювання та відновлення стану, а також у правилах, що визначають допустимі компроміси між узгодженістю та доступністю.

Окремим аспектом є розміщення та балансування даних у кластері. Для рівномірного розподілу ключів і зменшення вартості перерозподілу при зміні кількості вузлів застосовується consistent hashing, який історично розглядається

як ефективний інструмент “розвантаження гарячих точок” у розподіленому кешуванні та сховищах [9]. У середовищах із надлишковістю (реплікація, резервні фрагменти тощо) балансування ускладнюється, оскільки зростає кількість копій і змінюється профіль доступу; відповідно, оцінювання продуктивності балансування з урахуванням надлишковості є окремим напрямом досліджень [1]. У практичних прикладних роботах також простежуються підходи до підвищення продуктивності читання/запису через оптимізацію архітектурних шарів і розподіл функцій між обчислювальним шаром, доступом, зберіганням і агрегацією [23]. Для предметної області фінансових даних ці підходи є релевантними, оскільки одночасно потрібно забезпечувати і високу пропускну здатність (масові сканування), і прийнятну затримку для точкових запитів.

1.2.5 Паралельна обробка: підходи та вибір Spark

Аналітичні задачі у фінансах (агрегації за періодами, консолідація показників, виявлення аномалій, підготовка датасетів для моделей) часто є ресурсомісткими та виконуються над великими історичними масивами. У таких умовах ефективним є розпаралелювання обчислень у кластері. Apache Spark позиціонується як платформа кластерних обчислень із робочими наборами даних та підтримкою обчислювальних DAG, що дозволяє зменшувати витрати на введення-виведення завдяки кешуванню проміжних результатів у пам’яті [22]. Це має практичне значення для повторюваних фінансових розрахунків і конвеєрів підготовки даних, де одні й ті самі набори часто використовуються кілька разів у межах одного циклу обробки.

У [24] підкреслюється комбінування розподіленої архітектури зберігання з паралельною обробкою як спосіб підвищення швидкості читання/запису та зменшення часу виконання задач у порівнянні з традиційним підходом. Така постановка відповідає типовому архітектурному принципу: файлово-розподілений шар забезпечує зберігання великих масивів, а Spark виступає виконавчим шаром, який перетворює прикладну задачу на набір паралельних підзадач і виконує їх на ресурсах кластера [22, 24]. Доцільність подібної

композиції підтверджується також результатами порівняльних досліджень розподілених рушіїв у суміжних доменах, де кластерна обробка демонструє переваги при великих обсягах і складних запитах [17].

1.2.6 Безпека та якість даних у контурах Big Data для фінансів

Фінансові набори даних є конфіденційними, а тому технологічні рішення повинні враховувати криптографічний захист “на зберіганні” та/або “під час передавання”, контроль доступу і аудит дій користувачів та сервісів. Як базовий криптографічний стандарт у прикладних рішеннях широко використовується AES, визначений у FIPS 197 [14]. Окремі огляди зосереджуються на безпечних підходах у задачах фінансового ризик-аналізу та підкреслюють значущість комплексного поєднання захисту даних і надійності аналітичних процедур [16]. У контексті розподілених сховищ безпека стає міжкомпонентною властивістю: вона залежить від конфігурації сховища, мережевої взаємодії, політик доступу та коректної інтеграції прикладного шару.

Паралельно із безпекою визначальною є якість фінансових даних. Пропуски, дублікати, аномальні значення та неузгодженість форматів прямо впливають на коректність агрегатів і звітів. У наукових роботах пропонуються підходи до відновлення пропущених значень у великих наборах фінансової звітності з використанням методів машинного навчання [6]. Результати досліджень у сфері інтелектуальної обробки та аналізу великих даних акцентують увагу на необхідності інтеграції процедур підготовки та контролю якості в загальний технологічний контур, оскільки саме цей етап часто визначає надійність підсумкової аналітики [26, 28]. Окремо розглядаються підходи до імпутації пропусків у Big Data-інтерфейсах та методи відновлення відсутніх значень, що є релевантним для підготовки фінансових датасетів до подальшої обробки [19, 27]. Таким чином, у сучасних підходах якість даних розглядається як невід’ємна частина архітектури, а не як зовнішня процедура, що виконується “після” зберігання.

Отже, аналіз існуючих підходів показує, що ефективне опрацювання великих фінансових даних базується на поєднанні хмарного середовища,

розподіленого шару зберігання та паралельної обробки. Відомі рішення демонструють універсальність розподілених сховищ у різних доменах і актуальність задач оптимального розміщення та балансування даних [24]. Для формування архітектурних рішень доцільним є використання HDFS як файлового шару для великих історичних масивів та HBase як шару доступу за ключем для оперативних сценаріїв [3, 4, 18]. Паралельна обробка на базі Spark розглядається як практичний засіб зменшення часу виконання аналітичних задач у кластері [22]. Водночас для фінансової предметної області визначальними залишаються вимоги до узгодженості, балансування та надійності зберігання, що спираються на концепції консенсусу, consistent hashing і підходи до надлишковості та кодованого зберігання [1, 9, 10, 12]. Додатково обов'язковими складовими є криптографічний захист за стандартом AES та інтеграція процедур контролю якості і відновлення пропусків у конвеєр підготовки даних [14, 19, 26].

1.3 Постановка задачі дослідження

У межах кваліфікаційної роботи розглядається задача створення програмного модуля для розподіленого зберігання та паралельної обробки великих фінансових даних у кластерному (зокрема хмарному) середовищі. Актуальність задачі визначається зростанням обсягів фінансових масивів, різнорідністю джерел і форматів та потребою скорочення часу виконання аналітичних процедур без втрати відтворюваності й керованості [7, 11, 24]. Традиційні підходи на основі централізованих сховищ при збільшенні навантаження ускладнюють масштабування та підвищують ризики деградації продуктивності під час виконання масових агрегацій і регламентних розрахунків [11]. У зв'язку з цим доцільним є використання розподілених технологій зберігання та кластерної обробки, які забезпечують горизонтальне масштабування і підвищення пропускної здатності [13, 22].

Модуль передбачається реалізувати на основі поєднання файлового шару для великих історичних наборів (HDFS) і NoSQL-шару для оперативного доступу за ключем (HBase), а також застосування Apache Spark як виконавчого

шару паралельних обчислень [3, 4, 18, 22]. У хмарному контексті така архітектура узгоджується з моделлю надання ресурсів “на вимогу” та еластичним масштабуванням, що відповідає визначенню хмарних обчислень [13]. Для фінансових даних обов’язковими залишаються вимоги безпеки: контроль доступу, аудит та криптографічний захист [14, 16].

Мета роботи полягає у розробленні програмного модуля, який забезпечує масштабоване зберігання фінансових даних і паралельне виконання аналітичних задач із фіксацією метаданих, контролем якості та базовими механізмами безпеки.

Для досягнення мети необхідно розв’язати такі завдання:

– проаналізувати особливості фінансових даних як об’єкта розподіленого зберігання та паралельної обробки;

– виконати огляд існуючих підходів до хмарної організації обчислень і розподіленого зберігання;

– дослідити принципи паралельної обробки даних і обґрунтувати застосування Spark для аналітичних задач;

– визначити методи забезпечення узгодженості, балансування та надійності зберігання даних у розподіленому середовищі;

– сформувати вимоги до захисту фінансових даних та окреслити криптографічні механізми і контроль доступу;

– розробити архітектуру програмного модуля, визначити його компоненти, інтерфейси та потоки даних;

– реалізувати прототип модуля та провести експериментальне оцінювання швидкодії й масштабованості.

Вхідні дані – фінансові набори, що надходять у пакетному та/або потоковому режимах: транзакційні записи, показники звітності, довідникові дані та (за потреби) зовнішні параметри для перерахунків (курси, ставки) [8, 22]. Дані можуть бути представлені у структурованих (табличних) та напівструктурованих форматах (JSON/XML). Типовими проблемами є пропуски, дублікати, неузгодженість форматів дат і валют, що потребує процедур валідації та нормалізації [6, 19, 27]. За наявності регламентованої потреби допускається

застосування процедур відновлення пропущених значень із використанням методів, придатних для великих наборів даних [6, 19].

Вихідні дані – агрегати, вітрини даних і результати аналітичних обчислень (наприклад, підсумки за періодами, розрізи за рахунками/контрагентами/підрозділами), а також службова інформація: протоколи перевірок якості, журнали виконання задач та аудиту доступу [14, 16, 22]. Результати повинні бути придатними для подальшого використання прикладними системами (BI, звітність, ризик-аналіз) і відтворюваними за рахунок фіксації параметрів запусків і версій датасетів [8].

Вимоги поділяються на функціональні та нефункціональні.

Функціональні вимоги:

– приймання (ingestion) фінансових наборів із фіксацією метаданих джерела та часу надходження [8];

– валідація й нормалізація даних (схема, типи, ключі, часові поля, валюти), а також обробка дублікатів [11];

– контроль якості даних: перевірки повноти, діапазонів, узгодженості; формування звіту про дефекти [6, 27];

– робота з пропусками: виявлення та, за потреби, керована імпутація [6, 19];

– розподілене зберігання: запис історичних масивів у файловий шар і підтримка оперативного доступу за ключем у NoSQL-шарі [3, 4, 18];

– партиціювання й організація ключів доступу для зменшення ризику “гарячих точок” і підвищення рівномірності навантаження [4, 9];

– паралельна обробка: запуск Spark job для агрегацій, побудови вітрин і регламентних розрахунків [22, 24];

– публікація результатів: надання наборів даних та/або агрегатів через API або експорт у файлові формати [22];

– безпека та аудит: розмежування доступу, протоколювання критичних операцій, підтримка шифрування (за потреби) [14, 16].

Нефункціональні вимоги:

– продуктивність – скорочення часу виконання типових аналітичних

задач за рахунок паралельної обробки [22];

– масштабованість – можливість нарощування ресурсів шляхом додавання вузлів кластера без зміни логіки модуля [3, 18, 22];

– надійність і відмовостійкість – збереження даних при відмовах вузлів, використання реплікації/надлишковості; врахування впливу балансування за наявності надлишковості [1, 12, 18];

– узгодженість – коректні часові зрізи й контрольовані оновлення; концептуальне обґрунтування через протоколи консенсусу [10];

– безпека – забезпечення конфіденційності й контролю доступу, використання криптографічного захисту на основі AES [14];

– керованість – журналювання, фіксація метаданих запусків і параметрів обробки для відтворюваності результатів [8, 11].

Ефективність розробленого модуля доцільно оцінювати за такими показниками:

– пропускна здатність операцій читання/запису у розподіленому сховищі;

– час виконання типових Spark job (агрегації, формування вітрин, перевірки якості);

– масштабування при збільшенні ресурсів кластера (зміна часу виконання та використання ресурсів);

– показники якості даних після виконання перевірок і процедур оброблення пропусків (частка пропусків, кількість відхилених/помилкових записів).

План експерименту передбачає виконання однакових задач на фіксованому наборі фінансових даних із порівнянням двох конфігурацій:

1) послідовна/централізована обробка;

2) розподілене зберігання та паралельна обробка у кластері.

Для відтворюваності необхідно фіксувати конфігурацію середовища, параметри запусків і версії наборів даних [11, 22].

Отже, вданому підрозділі сформульовано постановку задачі розроблення програмного модуля розподіленого зберігання та паралельної обробки

фінансових даних. Визначено мету, завдання, вхідні та вихідні дані, а також ключові функціональні й нефункціональні вимоги. Запропоновано критерії оцінювання ефективності та загальний план експериментального дослідження.

2 ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО МОДУЛЯ

2.1 Проєктування архітектури програмного модуля

Архітектура програмного модуля розподіленого зберігання та паралельної обробки фінансових даних формується з урахуванням вимог продуктивності, масштабованості, відмовостійкості, узгодженості та безпеки, визначених у розділі 1 [11]. У фінансових системах одночасно існують два типи навантаження: операційне (швидкі вибірки за ключовими атрибутами) та аналітичне (масові сканування, агрегації, консолідація). Поєднання цих навантажень у межах одного централізованого сховища часто призводить до зростання затримок і ускладнює масштабування [11]. Тому доцільним є модульний підхід, у якому розподілене зберігання і паралельна обробка реалізуються як узгоджені компоненти єдиного контуру даних [22, 24].

Програмний модуль розглядається як інфраструктурний компонент між джерелами фінансових даних (ERP/CRM, транзакційні журнали, звітність, зовнішні показники) та споживачами результатів (BI, звітні підсистеми, прикладні сервіси аналітики). На концептуальному рівні модуль реалізує послідовність етапів: приймання даних → валідація/нормалізація → розподілене зберігання → паралельна обробка → публікація результатів [22, 24]. Така модель забезпечує відокремлення операційного та аналітичного контурів і дозволяє зберігати історичні масиви без погіршення доступу до “гарячих” даних.

Архітектура модуля доцільно поділяється на три рівні (рисунок 2.1) [29]:

1. Рівень зберігання (Storage Layer). Для довготривалого зберігання великих обсягів даних використовується HDFS як розподілена файлова система з блочною організацією та реплікацією [3, 18]. Для оперативного доступу за ключем застосовується HBase як колоночне NoSQL-сховище поверх HDFS [4]. Така комбінація забезпечує одночасно високу пропускну здатність при скануванні історичних даних і можливість швидких вибірок для сервісних сценаріїв [3, 4, 18].

2. Рівень обробки (Processing Layer). Паралельне виконання обчислювальних задач реалізується на Apache Spark, який підтримує кластерні

обчислення, DAG-планування та оптимізацію повторних обробок за рахунок роботи з робочими наборами даних [22]. Spark використовується для агрегацій, формування вітрин, регламентних розрахунків, перевірок якості та підготовки даних [22, 24].

3. Прикладний рівень доступу (Application/API Layer). Призначений для взаємодії із зовнішніми системами: керування завантаженнями, запуск задач, отримання результатів, перегляд статусів і протоколів перевірок. Наявність такого рівня узгоджується з практикою впровадження Big Data аналітики у фінансах, де важливими є керованість, прозорість обчислень і відтворюваність процедур [2, 8].

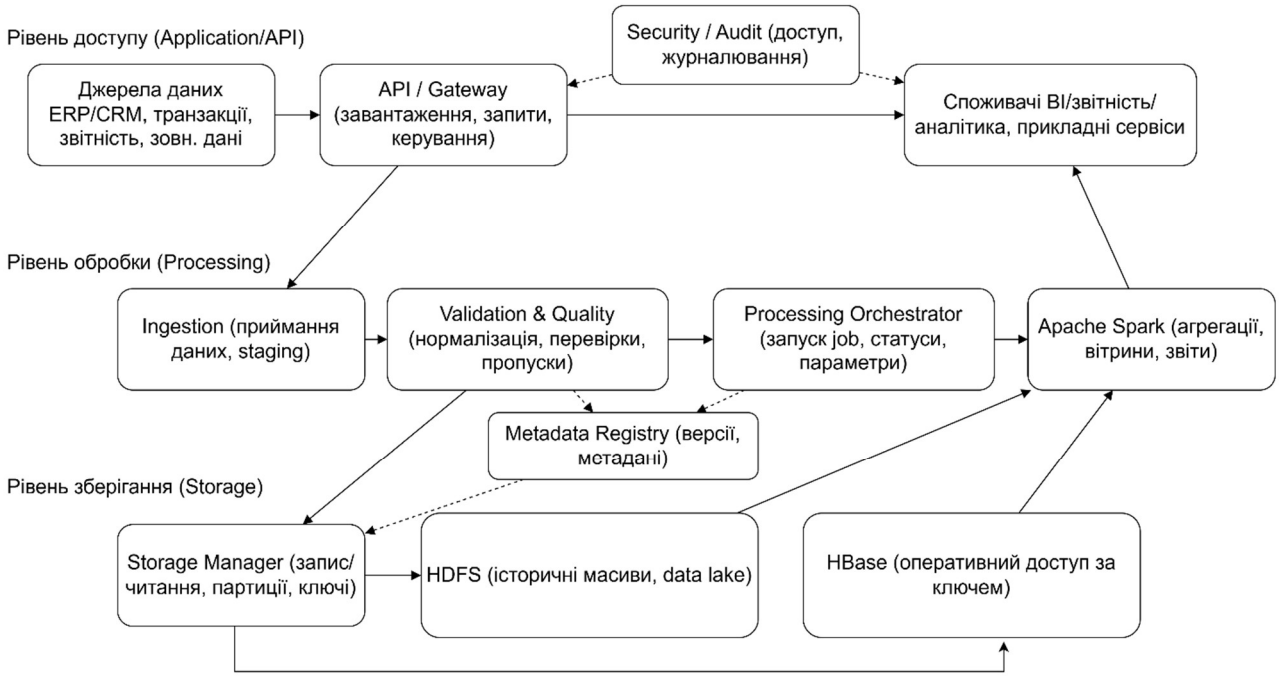

Рисунок 2.1 – Загальна архітектурна схема програмного модуля: взаємодія рівнів та основних компонентів [29]

Середовище розгортання визначається потребою в горизонтальному масштабуванні та керованості ресурсів. Для цього застосовується кластерна модель, яка може бути реалізована як у локальному дата-центрі, так і у хмарній інфраструктурі. Хмарні обчислення характеризуються наданням ресурсів “на вимогу”, широким мережевим доступом та еластичним масштабуванням, що створює передумови для ефективної обробки великих фінансових масивів при змінному навантаженні [13]. У межах модуля це означає можливість збільшення

ресурсів під час масових регламентних розрахунків і зменшення у періоди низької активності.

Водночас хмарний контекст підсилює вимоги до безпеки й контролю доступу. Для фінансових даних необхідно передбачити криптографічний захист і аудит операцій. Як базовий стандарт шифрування розглядається AES відповідно до FIPS 197 [14], а також протоколювання доступу і критичних дій у системі [16]. Концептуально коректність узгодженого стану в розподіленому середовищі пов’язується з ідеями консенсусу, що обґрунтовуються протоколами на кшталт Paxos [10]. На практичному рівні це відображається у вимогах до версіонування наборів даних, контрольованих повторних запусків job-ів і фіксації метаданих для відтворюваності результатів [8, 10].

Архітектура модуля формується як набір компонентів із чітко визначеними функціями та інтерфейсами [29]:

1. Ingestion-компонент. Приймає дані (файли/повідомлення), виконує первинну перевірку формату, фіксує метадані (джерело, час, тип набору) та розміщує дані у staging-зоні [8].

2. Компонент підготовки та контролю якості. Реалізує валідацію схеми, нормалізацію дат і валют, перевірку ключів, дедуплікацію, а також формує звіт про дефекти. Окремо передбачається виявлення пропусків і, за визначеними правилами, їх оброблення або імпутація, якщо це узгоджено з регламентом [6, 19, 27].

3. Storage Manager. Організує запис даних у HDFS (історичні, “холодні” набори) та у HBase (оперативні вибірки за ключем) [3, 4, 18]. Важливою умовою є коректне партиціювання і дизайн ключів, оскільки це впливає на рівномірність навантаження та ризик “гарячих точок” [4]. Як концептуальна основа вирівнювання розподілу ключів може використовуватися consistent hashing [9].

4. Processing Orchestrator. Керує запуском Spark job-ів, передає параметри виконання (період, набір даних, перелік метрик), отримує статуси та зберігає результати у визначених форматах [22]. Для підвищення продуктивності враховуються типові оптимізації Spark (розподіл партицій, мінімізація shuffle, використання кешування) [22].

5. Реєстр метаданих. Зберігає інформацію про версії датасетів, параметри запусків, результати перевірок якості, місця розміщення наборів у HDFS/HBase та технічні атрибути обробки. Такий реєстр є необхідним для data governance і відтворюваності розрахунків [2, 8].

6. Security/Audit-компонент. Забезпечує автентифікацію/авторизацію, розмежування прав доступу, журналювання та аудит критичних операцій, а також підтримку шифрування (за потреби) [14, 16].

7. API-шар. Надає уніфікований інтерфейс для завантаження, запуску задач, отримання результатів, метрик і протоколів якості. Це підвищує керованість та спрощує інтеграцію зі споживачами результатів [8].

Зазначені компоненти взаємодіють за принципом конвеєра: ingestion передає дані на підготовку, підготовка – в storage, storage забезпечує доступ обробці, Spark формує результати, а API надає результат і метадані зовнішнім системам. Такий підхід відповідає інженерним уявленням про проєктування розподілених сховищ і систем обробки великих даних, де важливою є декомпозиція на керовані компоненти [11, 24].

Відповідність ключових вимог компонентам архітектури подано у таблиці 2.1.

Таблиця 2.1 – Відповідність вимог компонентам архітектури

Вимога	Реалізація в модулі	Компонент(и)
Масштабованість	Горизонтальне нарощування вузлів кластера	HDFS/HBase/Spark [3, 4, 18, 22]
Відмовостійкість	Надлишковість зберігання, відновлення після відмов	HDFS (реплікація), політики доступу [18]
Продуктивність	Паралельні обчислення, оптимізація I/O	Spark + партиціювання HDFS [22]
Узгодженість/відтворюваність	Версіонування наборів, метадані запусків	Metadata Registry + правила ingestion [8, 10]
Безпека	Шифрування/контроль доступу/аудит	Security/Audit + AES [14, 16]
Якість даних	Перевірки, звіти, робота з пропусками	Validation & Quality [6, 19, 27]

Для перевірки повноти архітектури виділяються базові сценарії:

1. Сценарій завантаження: зовнішня система передає набір даних через

API; ingestion фіксує метадані; виконується валідація і контроль якості; дані розміщуються в HDFS/HBase; результати перевірок записуються в реєстр метаданих [3, 4, 8, 18].

2. Сценарій аналітичної обробки: через API ініціюється Spark job; orchestrator запускає обробку; результати (агрегати/вітрини) зберігаються у файловому шарі або таблицях для доступу; фіксуються параметри запуску й статус виконання [22, 24].

3. Сценарій контролю якості: виконуються перевірки повноти/узгодженості; формується звіт; за правилами може застосовуватися імпутація пропусків; результати процедур фіксуються для відтворюваності [6, 19, 27].

4. Сценарій безпечного доступу: запит результатів проходить перевірку прав; операція журналюється; результат надається у дозволеному форматі з урахуванням політик захисту [14, 16].

Отже, в даному підрозділі визначено архітектурну модель програмного модуля, що поєднує розподілене зберігання та паралельну обробку у кластерному або хмарному середовищі. Запропоновано ключові компоненти, описано їх взаємодію та типові потоки даних.

2.2 Проєктування підсистеми розподіленого зберігання

Підсистема зберігання визначає структуру даних, підхід до партиціювання, способи доступу та механізми надійності. Для фінансового домену критичними є:

1) здатність зберігати великі історичні масиви;

2) швидкий доступ до “гарячих” зрізів;

3) відмовостійкість;

4) контроль доступу й аудит;

5) підтримка версіонування для відтворюваності розрахунків [8, 11].

У модулі застосовано комбінацію HDFS + HBase, де HDFS використовується як файловий “data lake”, а HBase – як оперативний key-

value/column-family шар поверх HDFS [3, 4, 18].

HDFS орієнтована на зберігання великих файлів із блочною організацією, розміщенням блоків на різних вузлах та автоматичним відновленням при відмовах [18]. Практичні рекомендації щодо архітектури HDFS підкреслюють, що система оптимізована під високий throughput для великих обсягів і потоків читання/запису [3]. Це відповідає аналітичним задачам фінансів: формування звітності, агрегації за період, побудова вітрин [7, 22].

Для керованості дані доцільно організувати за зонами:

– RAW – сирі файли «як прийшли»;

– PROCESSED – дані після очищення і перевірок;

– CURATED/MART – готові вітрини й агрегати для BI [8, 22].

Такий поділ корисний ще й тим, що видно шлях даних: від первинного завантаження до готового результату.

Щоб Spark не читав «усе підряд», дані потрібно розкладати за папками (партиціями). Найчастіше для фінансових даних використовують:

– партиціювання за часом (рік/місяць/день);

– додатково за організаційними ознаками (компанія/підрозділ/тип операції).

Це зменшує обсяг даних для конкретного розрахунку і прискорює обробку [22]. Проте дрібні партиції можуть створити дуже багато файлів і дати зайві накладні витрати, тому потрібен баланс [11].

HBase – це NoSQL-сховище поверх HDFS, яке добре підходить для швидкого доступу за ключем [4]. Його доцільно використовувати тоді, коли потрібні швидкі вибірки, наприклад:

– отримати транзакції за рахунком/контрагентом;

– швидко видати підготовлені агрегати через API [4].

Головне рішення в HBase – як сформувати ключ рядка (rowkey). Якщо ключі зростають послідовно (наприклад, тільки час), то записи можуть «збиратися» на одному регіоні, і виникає гаряча точка [4, 11]. Щоб зменшити цей ефект, використовують підходи вирівнювання розподілу ключів. Ідея consistent hashing пояснює, як робити розподіл більш рівномірним і стабільним під час

масштабування [9]. На практиці це можна реалізувати префіксом (salt) або хешем, який “розкидає” записи по регіонах [4, 9].

У HDFS надійність забезпечується реплікацією блоків [18]. Водночас надлишковість впливає на продуктивність, бо з’являються різні копії даних і треба правильно вибирати, з якої читати. Питання балансування в системах із надлишковістю розглядаються в дослідженнях, де показано вплив політик доступу на затримку і розподіл навантаження [1].

Окремо існують підходи до більш економного зберігання через кодовані схеми (erasure/convertible codes). Вони дозволяють зменшити витрати зберігання при збереженні надійності, хоча складніші у впровадженні [12]. Для прототипу модуля базовим рішенням лишається реплікація.

Фінансові розрахунки повинні бути відтворюваними: якщо результат був отриманий на певній версії даних, його потрібно повторити при аудиті або перевірці. Тому в модулі передбачено:

– зберігання версій пакетів у HDFS (наприклад, окремі каталоги за датою/ідентифікатором завантаження);

– запис інформації про ці версії у Metadata Registry [8].

Концептуально потреба узгоджених станів у розподілених системах пояснюється підходами консенсусу (наприклад, Paxos) [10]. На практиці для модуля це означає: ідемпотентні завантаження, контроль статусів етапів та правила повторних запусків [8, 10].

2.3 Проєктування підсистеми паралельної обробки фінансових даних

Підсистема обробки призначена для задач, які складно або довго виконувати на одному сервері: агрегації, групування, об’єднання таблиць, перевірки якості, формування вітрин. Зростання ролі великих даних у фінансах підкреслюється у наукових оглядах, де зазначено, що дані стають фактором конкурентної переваги, але потребують інфраструктури для ефективної аналітики [7]. Тому обрано Spark як базовий інструмент паралельної обробки.

Spark підтримує кластерні обчислення і дозволяє виконувати перетворення

над даними паралельно. Він добре підходить для випадків, коли потрібно багато разів працювати з одними й тими самими наборами, бо може зберігати проміжні дані в пам'яті [22]. У межах модуля Spark читає дані з HDFS та, за потреби, використовує HBase як джерело оперативних довідників або агрегатів [3, 4, 22].

На рисунку 2.2 показано типовий конвеєр Spark-обробки: від RAW-зони до публікації результатів.

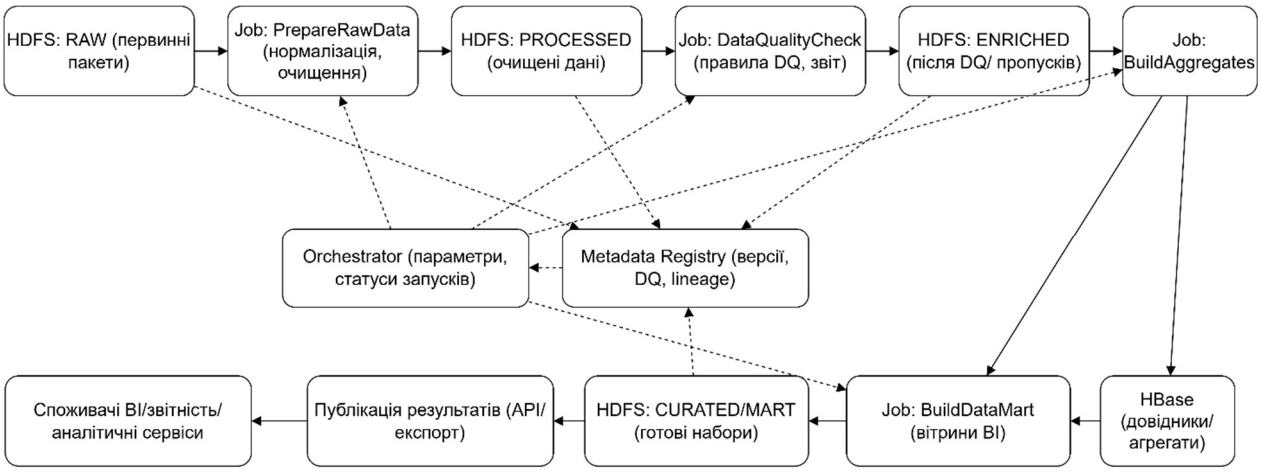

Рисунок 2.2 – Схема логічного конвеєра Spark-обробки фінансових даних (етапи та потоки)

Обробка організована як окремі завдання (job), які запускаються оркестратором. Типовий набір поданий у таблиці 2.3. Такий підхід дозволяє запускати лише потрібні етапи та спрощує супровід [11].

Таблиця 2.3 – Перелік Spark-завдань: призначення, вхід/вихід, режим виконання

Назва job	Призначення	Вхідні дані	Вихідні дані	Режим
PrepareRawData	Нормалізація схеми, типів, очищення	RAW (HDFS)	PROCESSED (HDFS)	batch
DataQualityCheck	Перевірки повноти/узгодженості	PROCESSED (HDFS)	DQ-звіт + rejects	batch
MissingValuesHandler	Обробка пропусків (за правилами)	PROCESSED (HDFS)	ENRICHED (HDFS) + лог змін	batch
BuildAggregates	Агрегації за періодами/сутностями	ENRICHED + довідники	AGG (HDFS/HBase)	batch
BuildDataMart	Побудова вітрин BI	AGG + довідники	CURATED/MART (HDFS)	batch
IncrementalUpdate	Обробка лише "дельти"	delta + метадані	оновлені partitions	batch

Щоб Spark працював швидко, важливо:

– правильно партиціювати дані (щоб читати не весь обсяг) [22];

– зменшувати shuffle (обмін даними між вузлами), бо він часто є «вузьким місцем» [22];

– використовувати broadcast join для малих довідників, коли це безпечно [22];

– не перевантажувати кластер надто великою кількістю паралельних завдань [11].

Ці правила не є складними, але суттєво впливають на час обробки та стабільність виконання [22].

Фінансові дані часто мають помилки: пропуски, некоректні коди, дублі, різні формати. Якщо не контролювати якість, агрегати та вітрини можуть бути неправильними. Тому в модулі обов'язково виконується Data Quality: перевірки повноти, діапазонів, логічних правил [6, 27].

Окремий випадок – пропущені значення. Для частини полів пропуск є критичним (наприклад, сума), а для частини – ні (наприклад, текстовий коментар). Тому потрібно:

– виявити пропуски;

– визначити критичність;

– або відхилити запис, або застосувати контрольовану імпутацію (лише там, де це допустимо) [6, 19, 27].

Підходи до імпутації у фінансових даних розглядаються у дослідженнях, зокрема на основі методів машинного навчання [6]. Напрацювання щодо відновлення пропусків у Big Data-середовищі також підтверджують практичну необхідність такого етапу [19, 27]. Схема цього процесу наведена на рисунку 2.3.

Щоб результат можна було повторити, кожен запуск завдання має бути описаний: який набір даних використано, за який період, які параметри застосовано, де збережено результат [8]. Такі дані фіксуються в Metadata Registry. Це відповідає підходам governance у фінансових організаціях, де важливі контроль і прозорість процесів [2, 8].

Рисунок 2.3 – Схема контролю якості та оброблення пропусків у Spark-конвеєрі

Оркестратор потрібний для запуску завдань у правильному порядку і для контролю їх завершення [11].

2.4 Проєктування інтерфейсів та прикладного шару (API)

API-шар забезпечує інтеграцію модуля з джерелами та споживачами. До ключових функцій належать:

– приймання пакетів даних і реєстрація метаданих ingestion;

– запуск обчислювальних job із параметрами (період, метрики, версія даних);

– отримання результатів (вітрини/агрегати) і супровідних DQ-звітів;

– отримання статусів виконання та журналів (для аудиту й аналізу збоїв) [8, 11].

У межах data governance важливим є надання метаданих: “який набір даних використано”, “які перетворення виконано”, “які дефекти якості виявлено” [2, 8]. Для фінансових споживачів це зменшує ризик неконтрольованих змін у конвеєрі та спрощує пояснення результатів.

2.5 Проєктування підсистеми безпеки та аудиту

Фінансові дані потребують контролю доступу, журналювання й криптографічного захисту. Базовим криптографічним механізмом для захисту даних на зберіганні/передаванні може виступати AES за стандартом FIPS 197 [14]. Підсистема Security/Audit реалізує:

– модель ролей і принцип мінімальних привілеїв;

– журналювання критичних операцій (завантаження, запуск job, експорт результатів);

– формування аудиторних записів для подальшого аналізу [16].

Для розподілених систем важливо також уникати неконсистентних станів при повторних запусках: застосовуються ідемпотентні операції ingestion та фіксація статусів етапів конвеєра. Концептуально необхідність узгоджених рішень у розподіленому середовищі пояснюється класами задач консенсусу [10]. Такі підходи підвищують стійкість конвеєра і знижують ризики помилок від “подвійних” завантажень або перезапусків.

Отже, у даному розділі сформовано проєктне рішення програмного модуля, що поєднує розподілене зберігання і паралельну обробку фінансових даних у кластерному/хмарному середовищі. Запропоновано загальну архітектуру з трьома рівнями та ключовими компонентами. Підсистема зберігання визначена як комбінація HDFS і HBase, із зонуванням і партиціюванням, а також правилами розподілу ключів для мінімізації “гарячих точок”. Підсистема паралельної обробки спроєктована на Apache Spark як набір завдань для підготовки даних, контролю якості, роботи з пропусками, агрегацій і побудови вітрин із урахуванням оптимізацій виконання. Окремо визначено принципи інтеграції через API, ведення метаданих для відтворюваності та підсистему безпеки й аудиту із застосуванням AES та журналюванням критичних операцій.

3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПРОГРАМНОГО МОДУЛЯ

3.1 Засоби та середовище реалізації програмного модуля

Реалізація модуля виконується засобами Python, оскільки PySpark дозволяє будувати паралельні обчислення у кластері без переходу на іншу мову програмування, а FastAPI забезпечує простий і зрозумілий спосіб організувати доступ через HTTP-інтерфейс. Обчислювальний шар реалізується на Apache Spark, який призначений для кластерної обробки великих обсягів даних і підтримує DAG-план виконання та оптимізацію перетворень [22]. Розподілене зберігання організовується на HDFS, що оптимізований для великих файлів і потокового читання/запису, а оперативний доступ за ключем забезпечується HBase [3, 4, 18].

Узагальнена схема розгортання наведена на рисунку 3.1.

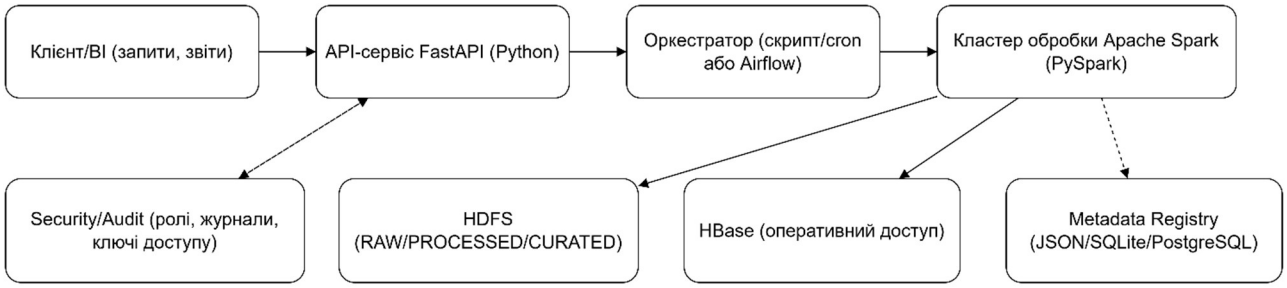

Рисунок 3.1 – Схема середовища розгортання

Клієнт (BI/аналітична система) надсилає запит до API-сервісу. API ініціює запуск обчислень через оркестратор (у простому випадку – системний планувальник cron або окремий Python-скрипт керування). Spark виконує обробку у кластері та записує результати у HDFS/HBase. Паралельно фіксуються технічні метадані запусків (ідентифікатор пакета, параметри, статус, посилання на результати), що підвищує відтворюваність і контроль процесу [8]. Для фінансових даних важливо передбачити контроль доступу і журналювання операцій; у роботі як базовий криптографічний орієнтир використовується стандарт AES (FIPS 197) [14], а організаційні аспекти керування даними узгоджуються з підходами data governance у фінансових установах [2].

Лістинг 3.1 – Запуск Spark-job на PySpark

```
# jobs/build_aggregates.py
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

def main(input_path: str, output_path: str, year: int, month: int) -> None:
    spark = (SparkSession.builder
             .appName("BuildAggregates")
             .getOrCreate())

    df = spark.read.parquet(f"{input_path}/year={year}/month={month:02d}/")

    # приклад агрегації: підсумок сум за контрагентом
    agg = (df.groupBy("counterparty_id")
             .agg(F.sum("amount").alias("sum_amount"),
                  F.count("*").alias("cnt_ops")))


agg.write.mode("overwrite").parquet(f"{output_path}/year={year}/month={month:02d
}/")
    spark.stop()

if __name__ == "__main__":
    main(
        input_path="/finance/processed/transactions",
        output_path="/finance/mart/agg_by_counterparty",
        year=2025,
        month=12
    )
```

3.2 Реалізація паралельної обробки даних у Spark

Підсистема зберігання реалізована як двокомпонентна:

1. HDFS використовується для накопичення великих обсягів пакетів даних та результатів пакетної обробки;

2. HBase застосовується для швидкого доступу до вибраних даних і агрегатів за ключем.

Такий поділ є практичним, оскільки HDFS оптимізований під великі файли та послідовне читання/запис, а HBase дає можливість отримувати потрібні записи без повного сканування великих наборів [3, 4, 18]. Для фінансових даних це доцільно, бо в одному сценарії потрібна масова аналітика (звітність, контроль), а в іншому – швидкі звернення за конкретними ідентифікаторами (пошук операції, витяг агрегату за контрагентом) [7, 11].

Зберігання в HDFS структуровано за принципом зон, що дозволяє розділити “сирі” дані, нормалізовані набори, очищені результати та фінальні

вітрини. Логіка зон і партиціювання показана на рисунку 3.2, а опис наведено в таблиці 3.1. Такий підхід підтримує контроль походження даних і повторюваність обробки, оскільки кожен етап має свій окремий шар [8, 22].

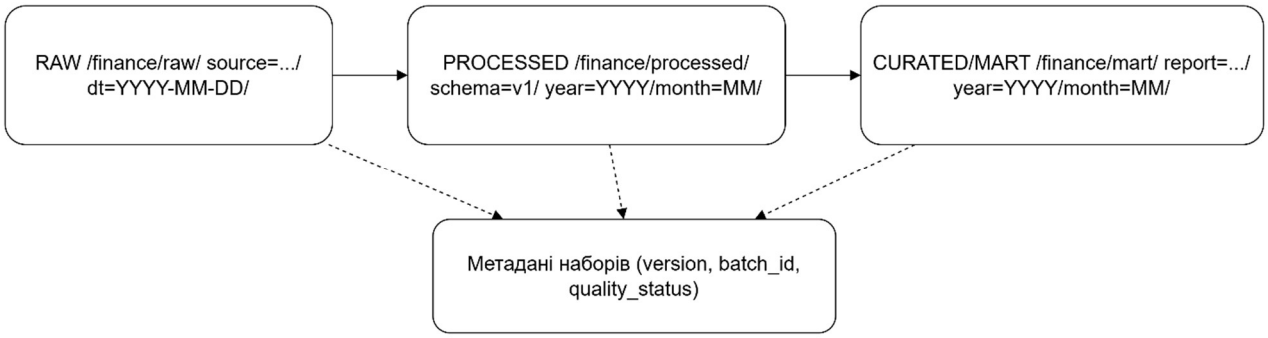

Рисунок 3.2 – Структура HDFS-зон і партицій

Таблиця 3.1 – Опис HDFS-зон, шляхів і партицій

Зона	Призначення	Приклад шляху	Партиції	Правило оновлення
RAW	сирі пакети	/finance/raw/source=bank_a/dt=2025-12-01/	source, dt	лише додавання
PROCESSED	нормалізовані дані	/finance/processed/schema=v1/year=2025/month=12/	year, month	перезапис періоду
ENRICHED	після DQ/пропусків	/finance/enriched/schema=v1/year=2025/month=12/	year, month	перезапис періоду
MART	вітрини/агрегати	/finance/mart/report=agg_by_counterparty/year=2025/month=12/	report, year, month	перезапис/ merge
REJECT	відхилені записи	/finance/reject/batch_id=2025-12-01_bank_a/	batch_id	лише додавання
DQ_REPORT	звіти якості	/finance/dq/batch_id=2025-12-01_bank_a/	batch_id	перезапис звіту

Партиціювання за year/month зменшує надлишкове читання, оскільки Spark може обробляти лише потрібний період. Це особливо важливо при регулярних розрахунках і оновленнях [22]. На рівні HDFS стабільність роботи забезпечується реплікацією блоків, що підвищує відмовостійкість та доступність даних [18]. Питання навантаження на сховище та балансування запитів є важливими для розподілених систем із надлишковістю, що відображається в сучасних дослідженнях з балансування навантаження [1].

HBase використовується для сценаріїв, де потрібна швидка вибірка “за ключем”, наприклад: отримання транзакцій за рахунком/контрагентом або

видача агрегатів за період [4]. Водночас неправильний дизайн ключа може створити “гарячі точки”, коли навантаження концентрується на частині регіонів. Для зменшення цього ризику застосовано складений ключ із “salt”-компонентом (приклад наведено на рисунку 3.3, опис в таблиці 3.2). Ідея рівномірного розподілу ключового простору узгоджується з підходами хешування у розподілених системах [9].

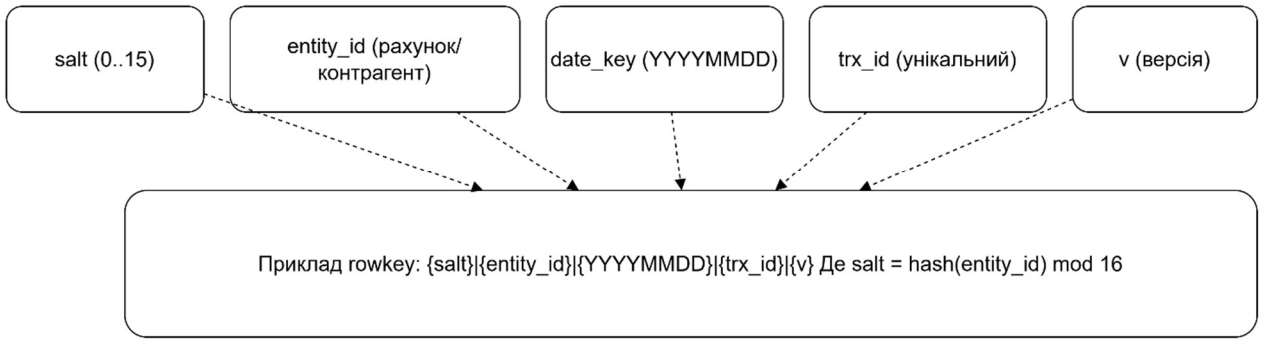

Рисунок 3.3 – Приклад формування ключа HBase (rowkey) для уникнення hot-spot

Таблиця 3.2 – Структура таблиць HBase

Таблиця HBase	Призначення	Rowkey (структура)	Column Family	Приклади колонок
finance_tx	швидкий доступ до транзакцій	{salt}|{entity_id}|{YYYYMMDD}|{trx_id}|v1	d	d:amount, d:currency, d:trx_date, d:type, d:counterparty_id
finance_agg_cp	агрегати за контрагентом	{salt}|{counterparty_id}|{YYYYMM}|v1	a	a:sum_amount, a:cnt_ops, a:last_update
ref_rates	курси валют	{YYYYMMDD}|{ccy}	r	r:rate, r:source

Паралельна обробка реалізована як ланцюжок окремих Spark-завдань (job), кожне з яких відповідає за конкретний етап: нормалізацію, контроль якості та пропусків, а також побудову агрегатів і вітрин. Такий підхід дозволяє запускати етапи незалежно й повторювати обробку лише там, де це потрібно. Загальна схема реалізованого конвеєра показана на рисунку 3.4.

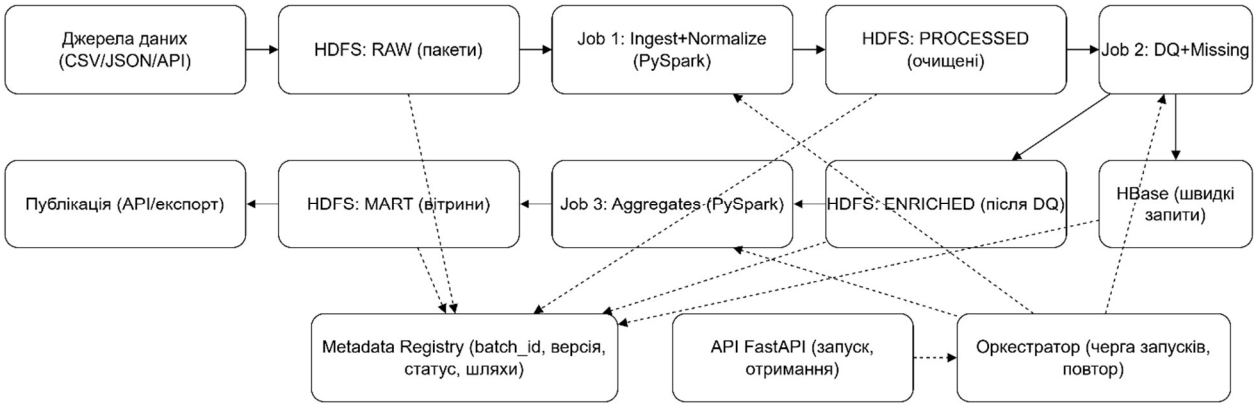

Рисунок 3.4 – Реалізований конвеєр обробки даних

Таблиця 3.3 – Реалізовані PySpark job-и та їх параметри

Job	Вхід	Основні параметри	Результат	Тип запуску
Job1_IngestNormalize	HDFS RAW	source, dt, schema_version=v1	HDFS PROCESSED	batch / incremental
Job2_DQ_Missing	HDFS PROCESSED	ruleset=default, allowed_ccy, impute_counterparty	HDFS ENRICHED + REJECT + DQ report	batch / incremental
Job3_BuildAggregates	HDFS ENRICHED	year, month, group_key	HDFS MART (+ опц. HBase)	batch / incremental

Перший етап конвеєра відповідає за приведення “сирих” даних до узгодженої схеми: встановлення типів, очищення строкових полів, базову перевірку формату дат, відкидання повністю порожніх записів. Результат записується у PROCESSED у форматі Parquet, оскільки він зручний для подальших обчислень у Spark [22].

Другий етап виконує перевірки якості, відхиляє критично некоректні записи і застосовує прості правила заповнення для некритичних пропусків. Логіка цього кроку показана на рисунку 3.5. Важливість коректної обробки пропусків у фінансових даних підтверджується дослідженнями щодо імпутації та підготовки фінансових наборів [6]. Для прототипу використано політики, де критичні поля відсікаються, а менш критичні можуть заповнюватися з позначенням факту імпутації [19, 27].

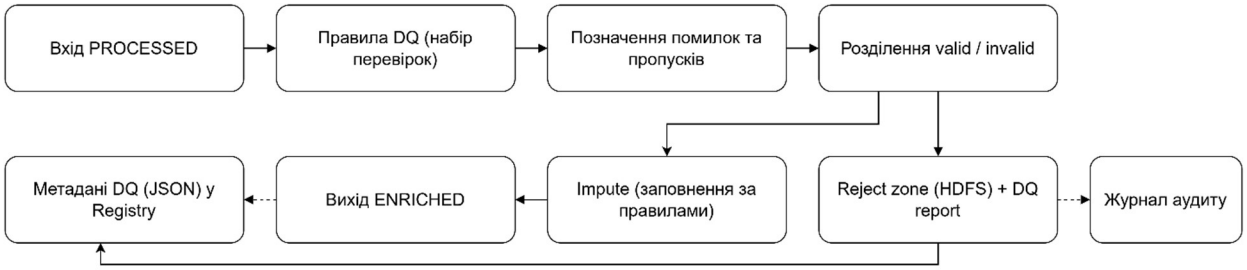

Рисунок 3.5 – Реалізація DQ та оброблення пропусків

Таблиця 3.4 – Правила контролю якості даних (DQ) для транзакцій

Код	Перевірка	Поле(я)	Рівень	Дія
DQ_01	amount не NULL	amount	критичне	reject
DQ_02	amount > 0	amount	критичне	reject
DQ_03	currency ∈ {UAH, USD, EUR}	currency	критичне	reject
DQ_04	trx_date не NULL	trx_date	критичне	reject
DQ_05	trx_id не NULL	trx_id	критичне	reject
DQ_06	counterparty_id може бути NULL	counterparty_id	некритичне	impute “UNKNOWN” + прапорець
DQ_07	дубль trx_id у межах batch	trx_id	критичне	reject

Таблиця 3.5 – Політики оброблення пропусків

Поле	Критичність	Дія	Як фіксується
amount	критичне	reject	причина amount_null
trx_date	критичне	reject	причина date_null
currency	критичне	reject	причина ccy_invalid/ccy_null
counterparty_id	середня	UNKNOWN + позначка	imputed_counterparty=1
comment	низька	залишити NULL	без змін

Третій етап формує агреговані показники, потрібні для звітів та аналітики: підсумки за контрагентом, кількість операцій, розподіли за періодами. Обчислення виконуються у Spark через операції групування та агрегації. Для зменшення часу виконання застосовуються практичні оптимізації Spark: партиціювання за періодом, обмеження shuffle, а також broadcast-join для малих довідників [22].

Таблиця 3.6 – Оптимізації Spark у реалізації та очікуваний ефект

Прийом	Де застосовується	Навіщо	Очікуваний ефект
Parquet + snappy	PROCESSED / ENRICHED/MART	зменшити I/O	швидше читання порівняно з CSV
Партиціювання year/month	Job2–Job3	читати лише період	економія часу читання
Broadcast join	Job3	прискорити join	корисно для довідників
Контроль shuffle partitions	Job3	стабільність часу	менше “провалів” продуктивності
Кешування (вибірково)	Job3	повторне використання DF	скорочення повторних читань

3.3 Реалізація API та керування виконанням

Для інтеграції з користувацькими системами (аналітика, BI, сервіс звітів) у модулі реалізовано простий REST API. API приймає параметри запуску обробки, повертає ідентифікатор запуску, а також дозволяє отримувати статус і розташування результатів. Такий підхід підвищує керованість процесу та спрощує контроль запусків, що є важливим для фінансових систем і governance-підходів [2, 8].

Таблиця 3.7 – Специфікація REST API

Метод	Endpoint	Призначення	Параметри	Відповідь
POST	/runs/start	старт обробки	year, month, mode	run_id
GET	/runs/{run_id}	статус	run_id	status, timestamps
GET	/runs/{run_id}/result	шляхи результатів	run_id	mart_path, enriched_path
GET	/runs/{run_id}/dq	DQ-звіт	run_id	показники якості

Фіксація метаданих забезпечує повторюваність обробки: можна визначити, який пакет і з якими параметрами був оброблений, де лежать результати, які були відхилення за DQ. Такий підхід практично пов’язаний із вимогами контролю процесів у розподілених системах, де важливий коректний статус та уникнення неконсистентних повторів [10].

Таблиця 3.8 – Поля Metadata Registry (приклад запису)

Поле	Опис	Приклад значення
run_id	ідентифікатор запуску	7e2c1b8a-1f24-4a0b-9c2a-1f8b7f2e4a10
batch_id	ідентифікатор пакета	2025-12-01_bank_a
period	період	2025-12
mode	режим	batch
status	стан	finished
input_paths	вхідні дані	/finance/raw/source=bank_a/dt=2025-12-01/
output_paths	вихідні дані	processed=…; enriched=…; mart=…
dq_report_path	звіт DQ	/finance/dq/batch_id=2025-12-01_bank_a/
started_at	час старту	2026-01-20 10:05:12
finished_at	час завершення	2026-01-20 10:07:06
error_message	помилка	NULL

У додатку А наведено ключові фрагменти програмної реалізації модуля: REST API для запуску обробки, скрипт керування виконанням, Spark job-и конвеєра, а також допоміжні функції для формування ключів HBase та запису даних.

3.4 Експериментальні дослідження ефективності модуля

Експериментальні дослідження проведено з метою оцінювання продуктивності конвеєра при зростанні обсягу транзакційних даних, а також для порівняння режимів batch та incremental. Необхідність подібних оцінок пояснюється тим, що сучасні фінансові організації активно використовують великі дані, а проблеми часу обробки і масштабування є ключовими.

Для експерименту використано кластерну конфігурацію, наведеної в таблиці 3.9. Умови відповідають загальним підходам до хмарних обчислень: ресурси є розподіленими та надаються як спільна інфраструктура. На рівні сховища застосована реплікація HDFS, що підвищує надійність зберігання.

Для тестування підготовлено 4 набори даних з різним обсягом та часткою пропусків (таблиця 3.10). Вибір частки пропусків обґрунтований тим, що фінансові набори часто містять відсутні значення і потребують коректної обробки.

Таблиця 3.9 – Конфігурація кластера/середовища для експериментів

Параметр	Значення
Кількість вузлів	5 (1 master + 4 worker)
CPU на вузол	8 vCPU (workers), 4 vCPU (master)
RAM на вузол	32 GB (workers), 16 GB (master)
Диск	SSD 500 GB на кожному worker
Мережа	1 Gbps
Реплікація HDFS	2
Spark executors	8 (по 2 на worker)
Spark executor memory	8 GB
Spark executor cores	4
Формат даних	Parquet (snappy)

Таблиця 3.10 – Тестові набори даних для експериментів

Набір	Опис	Обсяг, млн рядків	Розмір, ГБ	Частка пропусків, %
D1	базовий пакет транзакцій	1	0.45	2
D2	середній пакет транзакцій	5	2.20	3
D3	великий пакет транзакцій	10	4.40	5
D4	дуже великий пакет транзакцій	20	8.80	7

Оцінювання проводиться за метриками часу виконання job-ів, загального часу конвеєра, пропускної здатності, а також за показниками reject та імпутації (таблиця 3.11). Spark використовується як інструмент паралельної обробки, тому основна частина оцінювання зосереджена на часі виконання і стабільності конвеєра.

Таблиця 3.11 – Метрики оцінювання та спосіб вимірювання

Метрика	Позначення	Як береться	Одиниці
Час Job1	T1T_1T1	час spark-submit для Job1	хв
Час Job2	T2T_2T2	час spark-submit для Job2	хв
Час Job3	T3T_3T3	час spark-submit для Job3	хв
Час конвеєра	TpipeT_{pipe}Tpipe	T1+T2+T3T_1+T_2+T_3T1 +T2+T3	хв
Throughput	QQQ	rows_total/(T_pipe*60)	рядків/с
Частка reject	RRR	rows_rejected/rows_total*100	%
Частка імпутації	III	rows_imputed/(rows_total-rows_rejected)*100	%
Затримка API (опц.)	LLL	середнє за серією запитів	мс

Результати продуктивності при зростанні обсягу даних наведено в таблиці 3.12. Видно, що із зростанням обсягу даних зростає і час виконання конвеєра. Така поведінка є очікуваною для кластерної обробки: збільшення обсягу означає більший час читання, більше перетворень і більше операцій обміну даними між виконавцями. Партиціювання у HDFS та використання Parquet знижує накладні витрати на читання.

Таблиця 3.12 – Результати продуктивності job-ів залежно від обсягу

Обсяг, млн рядків	Job1, хв	Job2, хв	Job3, хв	Усього, хв
1	0.8	0.6	0.5	1.9
5	2.9	2.4	1.9	7.2
10	5.4	4.6	3.7	13.7
20	10.6	9.1	7.4	27.1

Для практичного використання важливо, що навіть при збільшенні обсягу до 20 млн рядків конвеєр залишається керованим і не демонструє різких “провалів”, що вказує на прийнятну конфігурацію ресурсів і коректну організацію зберігання/обробки.

Для реальних систем часто потрібні два режими роботи:

– batch – повний перерахунок періоду;

– incremental – обробка лише нового пакета та дозапис результатів.

Порівняння наведено у таблиці 3.13. Інкрементальний режим дає суттєвий виграш у часі при великих обсягах, оскільки обробляється менша частина даних. Водночас він потребує чіткого обліку метаданих запусків і контролю консистентності, щоб уникнути дублювання або пропуску оновлень.

Таблиця 3.13 – Порівняння режимів batch та incremental

Обсяг, млн рядків	Batch, хв	Incremental, хв	Різниця (Batch–Incr), хв
1	1.9	0.9	1.0
5	7.2	2.5	4.7
10	13.7	4.1	9.6
20	27.1	6.9	20.2

Показники якості наведено у таблиці 3.14. Частка reject зростає разом із

часткою пропусків і неякісних записів у даних, що відповідає очікуваній поведінці: більший обсяг і більша “брудність” дають більше відхилень. Частка імпутації демонструє, яку частину записів вдалося зберегти завдяки заповненню некритичних пропусків із фіксацією відповідного прапорця. Для фінансових даних важливо не лише заповнити значення, а й зафіксувати факт такого втручання, щоб результати можна було інтерпретувати коректно.

Таблиця 3.14 – Результати DQ та оброблення пропусків

Набір	rows_total	rows_rejected	reject, %	rows_imputed	imputed, %
D1	1 000 000	12 000	1.20	18 000	1.82
D2	5 000 000	70 000	1.40	120 000	2.44
D3	10 000 000	160 000	1.60	260 000	2.64
D4	20 000 000	360 000	1.80	520 000	2.65

Отже, у даному розділі реалізовано прототип програмного модуля, що поєднує HDFS як базове розподілене сховище, Spark як інструмент паралельної обробки та HBase для швидких вибірок за ключем. Реалізація побудована як конвеєр із трьох Spark-завдань: нормалізація даних, контроль якості та пропусків, формування агрегатів і вітрин. Для керованості процесу передбачено REST API та облік метаданих запусків, що відповідає практикам data governance у фінансових організаціях. Експериментальні результати показали, що зі зростанням обсягу даних час виконання конвеєра зростає прогнозовано, а інкрементальний режим дає суттєвий виграш у часі при великих обсягах за рахунок обробки лише нових пакетів. Контроль якості та політики пропусків дозволили відхиляти критично некоректні записи та зберігати частину даних через імпутацію некритичних полів із фіксацією відповідних ознак.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано завдання розроблення програмного модуля розподіленого зберігання та паралельної обробки фінансових даних. Актуальність теми зумовлена зростанням обсягів транзакційної інформації та потребою фінансових організацій у масштабованих рішеннях, які забезпечують прийнятний час обробки, контроль якості даних і відтворюваність результатів.

У межах виконання роботи отримано такі основні результати:

1. Проаналізовано предметну область розподіленого зберігання та паралельної обробки великих фінансових даних. Показано, що для фінансових сценаріїв доцільним є поєднання розподіленого файлового сховища для пакетної аналітики та спеціалізованого сховища для швидких вибірок за ключем.

2. Виконано огляд і порівняння сучасних підходів до побудови розподілених систем: принципи розподілу навантаження й надлишковості, підходи до рівномірного розміщення ключів, механізми узгодженості та інструменти кластерної обробки. Обґрунтовано доцільність використання Apache Spark як базового засобу паралельної обробки, а HDFS – як компонента довготривалого зберігання.

3. Сформульовано постановку задачі дослідження: визначено вхідні дані (пакети транзакцій), вимоги до вихідних результатів (нормалізовані набори, очищені дані, вітрини/агрегати), а також обмеження щодо якості, відмовостійкості та керованості процесу обробки.

4. Спроєктовано архітектуру програмного модуля з поділом на рівні зберігання, паралельної обробки, контролю якості, керування виконанням та інтеграції через API. Запропонована архітектура передбачає зонування даних у HDFS, використання HBase для оперативного доступу, а також оркестрацію запусків і реєстрацію метаданих.

5. Розроблено інформаційне та алгоритмічне забезпечення для підготовки даних та контролю їх якості: визначено базові правила DQ, політики відхилення некоректних записів і правила заповнення некритичних пропусків із фіксацією фактів імпутації. Такий підхід забезпечує прозорість перетворень і

коректність подальших агрегувань.

6. Реалізовано програмний прототип на Python із використанням PySpark для пакетної обробки та FastAPI для керування запуском і доступу до статусів та результатів. Реалізація виконана у вигляді конвеєра з окремих Spark job-ів (нормалізація, DQ/пропуски, агрегування), що спрощує повторні запуски, локалізацію помилок та масштабування обробки.

7. Проведено експериментальне оцінювання ефективності конвеєра на наборах даних різного обсягу. Результати показали прогнозоване зростання часу виконання при збільшенні обсягу транзакцій, а також суттєву перевагу інкрементального режиму обробки для регулярних оновлень, за умови коректного обліку запусків і метаданих. Окремо підтверджено практичну важливість кроку DQ та оброблення пропусків, оскільки він знижує ризики спотворення підсумкових показників.

8. Загалом отримані результати підтверджують, що запропонований програмний модуль забезпечує масштабоване зберігання і паралельну обробку фінансових даних, підтримує контроль якості та забезпечує керованість процесу через API та реєстрацію метаданих. Розроблений прототип може бути використаний як основа для впровадження в задачах аналітики фінансових операцій, формування вітрин і підготовки даних для подальших моделей ризик-аналізу та бізнес-аналізу.

9. Перспективами подальшого розвитку є: розширення набору правил DQ та механізмів валідації, впровадження більш повного оркестратора запусків, оптимізація зберігання (розширене партиціювання, керування малими файлами), а також підсилення засобів захисту даних і розмежування доступу відповідно до вимог безпеки фінансових систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aktas M. F., Behrouzi-Far A., Soljanin E. et al. Evaluating Load Balancing Performance in Distributed Storage With Redundancy. *IEEE Transactions on Information Theory*. 2021. Vol. 67, no. 6. P. 3623–3644.

2. Al-Afeef M., Ali O., Al-Tahat S. et al. The effect of big data governance on financial technology in Jordanian commercial banks: The mediation role of organizational culture. *International Journal of Data and Network Science*. 2023. Vol. 7, no. 3. P. 1283–1294.

3. Apache Software Foundation. *Apache Hadoop: HDFS Architecture Guide* [Електронний ресурс]. Режим доступу: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (дата звернення: 24.01.2026).

4. Apache Software Foundation. *Apache HBase® Reference Guide* [Електронний ресурс]. Режим доступу: https://hbase.apache.org/book.html (дата звернення: 24.01.2026).

5. Choi B. J., Kim C. S., Lee M. C. Research trends on distributed storage technology for blockchain transaction data. *Electronics and Telecommunications Trends*. 2022. Vol. 37, no. 3. P. 85–96.

6. Fujimoto S., Mizuno T., Ishikawa A. Interpolation of non-random missing values in financial statements' big data using CatBoost. *Journal of Computational Social Science*. 2022. Vol. 5, no. 2. P. 1281–1301.

7. Goldstein I., Spatt C. S., Ye M. Big data in finance. *The Review of Financial Studies*. 2021. Vol. 34, no. 7. P. 3213–3225.

8. Hajiheydari N., Delgosha M. S., Wang Y. et al. Exploring the paths to big data analytics implementation success in banking and financial service: an integrated approach. *Industrial Management & Data Systems*. 2021. Vol. 121, no. 12. P. 2498–2529.

9. Karger D. R., Lehman E., Leighton F. T. et al. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory*

of Computing (STOC ’97). 1997. P. 654–663.

10. Lamport L. Paxos Made Simple. *ACM SIGACT News (Distributed Computing Column)*. 2001. Vol. 32, no. 4. P. 51–58.

11. Legault M. A practitioner’s view on distributed storage systems: Overview, challenges and potential solutions. *Technology Innovation Management Review*. 2021. Vol. 11, no. 6. P. 32–41.

12. Maturana F., Rashmi K. V. Convertible codes: Enabling efficient conversion of coded data in distributed storage. *IEEE Transactions on Information Theory*. 2022. Vol. 68, no. 7. P. 4392–4407.

13. Mell P., Grance T. *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145. Gaithersburg, MD : National Institute of Standards and Technology, 2011.

14. National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard (AES)*. Gaithersburg, MD : NIST, 2001.

15. Qin Qiancong, Dong Xu, Guo Hu. Distributed storage method of UAV clusters for battlefield conditions. *Journal of Southwest Jiaotong University*. 2024. Vol. 59, no. 4. P. 942–958.

16. Regin R., Rajest S. S., Shynu T. A review of secure neural networks and big data mining applications in financial risk assessment. *Central Asian Journal of Innovations on Tourism Management and Finance*. 2023. Vol. 4, no. 2. P. 73–90.

17. Shin H., Lee K., Kwon H. Y. A comparative experimental study of distributed storage engines for big spatial data processing using GeoSpark. *The Journal of Supercomputing*. 2022. Vol. 78, no. 2. P. 2556–2579.

18. Shvachko K., Kuang H., Radia S., Chansler R. The Hadoop Distributed File System. In: *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 2010. P. 1–10.

19. Wang C., Shakhovska N., Sachenko A., Komar M. A New Approach for Missing Data Imputation in Big Data Interface. Information Technology and Control. – 2020. – Vol. 49. No 4. – Pp. 541-555.

20. Wu Daitao, Liu Long, Cui Peng. Adaptive data layout strategy in mobile distributed storage system. *Journal of Software*. 2024. Vol. 35, no. 10. P. 4912–4929.

21. Wu Zheng, Li Jun, Li Haishan. Research on construction mode and service efficiency of earthquake data distributed storage system. *Earthquake in China*. 2024. Vol. 40, no. 1. P. 251–259.

22. Zaharia M., Chowdhury M., Franklin M. J., Shenker S., Stoica I. Spark: Cluster Computing with Working Sets. In: *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. 2010. 10 p.

23. Zhang Maiqi. Distributed storage method for enterprise remote office files based on cloud platform. *Automation Technology and Application*. 2024. Vol. 43, no. 3. P. 112–115.

24. Zhang Xin. Distributed Storage and Parallel Processing Technology of Financial Big Data under Cloud Computing Platform. *Procedia Computer Science*. 2025. Vol. 262. P. 714–721.

25. Zhao Xiaofan, Du Shuming, Liu Chao. Exploration and research on building business analysis model based on distributed storage of power grid. *Automation Technology and Application*. 2024. Vol. 43, no. 3. P. 124–127.

26. Комар М.П. Інформаційна технологія інтелектуальної обробки та аналізу великих даних. *Вісник Хмельницького національного університету. Технічні науки*. – 2020. – № 5. – С. 125–130.

27. Комар М.П. Методи відновлення відсутніх даних у інтерфейсі великих даних. *Вимірювальна та обчислювальна техніка в технологічних процесах*. – 2020. – №5. – С. 97–103.

28. Комар М.П., Перевізник Р.М., Неспляк Д.Б. та ін. Проектування прикладних систем обробки та аналізу великих даних на основі глибоких нейронних мереж. *Актуальні задачі сучасних технологій : зб. тез доп. міжнар. наук.-техн. конф*. Тернопіль, 25-26 листопада, 2020. Т.2. С.30-31.

29. Ярунів М.А. Програмний модуль розподіленого зберігання та паралельної обробки фінансових даних. Інтелектуальні комп'ютерні системи та мережі» *(ІКСМ) : матеріали IV Всеукраїнської науково-практичної конференції молодих вчених і студентів* (м. Тернопіль, ЗУНУ, 20 травня 2026 р.). Тернопіль, 2026. С. 78-81.

30. Островерхов В. М., Біловус Л. І., Возьний К. З., Луцишин О. О.,

Монастирський Г. Л., Надвиничний С. А., Питель С. В., Шандрук С. К. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

31. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Коди реалізації програмного модуля

Лістинг А.1 – REST API сервіс (FastAPI) для запуску обробки

```
# api/app.py
from fastapi import FastAPI
from pydantic import BaseModel
import uuid
import json
from pathlib import Path
import subprocess
from datetime import datetime

app = FastAPI(title="Finance Distributed Module API", version="1.0")

REGISTRY_PATH = Path("./registry.json")


class StartRequest(BaseModel):
    year: int
    month: int
    mode: str = "batch"  # batch або incremental


def _load_registry() -> dict:
    if REGISTRY_PATH.exists():
        return json.loads(REGISTRY_PATH.read_text(encoding="utf-8"))
    return {}


def _save_registry(reg: dict) -> None:
    REGISTRY_PATH.write_text(
        json.dumps(reg, ensure_ascii=False, indent=2),
        encoding="utf-8"
    )


@app.post("/runs/start")
def start_run(req: StartRequest):
    run_id = str(uuid.uuid4())

    reg = _load_registry()
    reg[run_id] = {
        "status": "started",
        "year": req.year,
        "month": req.month,
        "mode": req.mode,
        "started_at": datetime.now().isoformat(timespec="seconds"),
        "finished_at": None,
        "error_message": None,
        "result": {}
    }
    _save_registry(reg)

    # Прототип: запуск runner окремим процесом.
    # У промисловому варіанті доцільні черга/оркестратор.
    subprocess.Popen(["python", "runner.py", run_id])

    return {"run_id": run_id}
```

```
@app.get("/runs/{run_id}")
def get_status(run_id: str):
    reg = _load_registry()
    if run_id not in reg:
        return {"error": "run_id not found"}
    return reg[run_id]


@app.get("/runs/{run_id}/result")
def get_result(run_id: str):
    reg = _load_registry()
    if run_id not in reg:
        return {"error": "run_id not found"}
    return reg[run_id].get("result", {})


@app.get("/runs/{run_id}/dq")
def get_dq(run_id: str):
    reg = _load_registry()
    if run_id not in reg:
        return {"error": "run_id not found"}
    return reg[run_id].get("result", {}).get("dq_report", {})
```

Лістинг А.2 – Скрипт керування виконанням конвеєра (runner)

```
# runner.py
import sys
import json
from pathlib import Path
from datetime import datetime
import subprocess

REGISTRY_PATH = Path("./registry.json")


def _load_registry() -> dict:
    return json.loads(REGISTRY_PATH.read_text(encoding="utf-8"))


def _save_registry(reg: dict) -> None:
    REGISTRY_PATH.write_text(
        json.dumps(reg, ensure_ascii=False, indent=2),
        encoding="utf-8"
    )


def update_run(run_id: str, patch: dict) -> None:
    reg = _load_registry()
    reg[run_id].update(patch)
    _save_registry(reg)


def update_result(run_id: str, patch: dict) -> None:
    reg = _load_registry()
    reg[run_id]["result"].update(patch)
    _save_registry(reg)


def main(run_id: str) -> None:
    reg = _load_registry()
    if run_id not in reg:
        return
```

```
    year = reg[run_id]["year"]
    month = reg[run_id]["month"]
    mode = reg[run_id]["mode"]

    # Приклад шляхів (можна винести у конфігураційний файл)
    raw_path = f"/finance/raw/source=bank_a/dt={year}-{month:02d}-01/"
    processed_path = f"/finance/processed/schema=v1/year={year}/month={month:02d}/"
    enriched_path = f"/finance/enriched/schema=v1/year={year}/month={month:02d}/"
    reject_path = f"/finance/reject/batch_id={year}-{month:02d}_bank_a/"
    dq_report_path = f"/finance/dq/batch_id={year}-{month:02d}_bank_a/"
    mart_path =
f"/finance/mart/report=agg_by_counterparty/year={year}/month={month:02d}/"

    try:
        update_run(run_id, {"status": "running"})

        subprocess.check_call([
            "spark-submit", "jobs/job1_ingest_normalize.py",
            raw_path, processed_path
        ])

        subprocess.check_call([
            "spark-submit", "jobs/job2_dq_missing.py",
            processed_path, enriched_path, reject_path, dq_report_path
        ])

        subprocess.check_call([
            "spark-submit", "jobs/job3_build_aggregates.py",
            enriched_path, mart_path, str(year), str(month)
        ])

        update_result(run_id, {
            "paths": {
                "raw": raw_path,
                "processed": processed_path,
                "enriched": enriched_path,
                "reject": reject_path,
                "dq_report": dq_report_path,
                "mart": mart_path
            }
        })

        update_run(run_id, {
            "status": "finished",
            "finished_at": datetime.now().isoformat(timespec="seconds")
        })

    except Exception as e:
        update_run(run_id, {
            "status": "failed",
            "finished_at": datetime.now().isoformat(timespec="seconds"),
            "error_message": str(e)
        })


if __name__ == "__main__":
    main(sys.argv[1])
```

Лістинг А.3 – Job 1: завантаження та нормалізація даних (PySpark)

```
# jobs/job1_ingest_normalize.py
import sys
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
```

```
def run(raw_path: str, processed_path: str) -> None:
    spark = SparkSession.builder.appName("Job1_IngestNormalize").getOrCreate()

    df = (spark.read
          .option("header", True)
          .csv(raw_path))

    df2 = (df
        .withColumn("amount", F.col("amount").cast("double"))
        .withColumn("trx_date", F.to_date("trx_date", "yyyy-MM-dd"))
        .withColumn("currency", F.upper(F.trim(F.col("currency"))))
        .withColumn("trx_id", F.trim(F.col("trx_id")))
        .withColumn("counterparty_id", F.trim(F.col("counterparty_id")))
    ).dropna(how="all")

    df2.write.mode("overwrite").parquet(processed_path)
    spark.stop()

if __name__ == "__main__":
    run(sys.argv[1], sys.argv[2])
```

Лістинг А.4 – Job 2: контроль якості та оброблення пропусків (PySpark)

```
# jobs/job2_dq_missing.py
import sys
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

ALLOWED_CCY = ["UAH", "USD", "EUR"]

def run(processed_path: str, enriched_path: str, reject_path: str, dq_report_path: str)
-> None:
    spark = SparkSession.builder.appName("Job2_DQ_Missing").getOrCreate()
    df = spark.read.parquet(processed_path)

    df_flags = (df
        .withColumn("err_amount_null", F.col("amount").isNull())
        .withColumn("err_amount_nonpos", F.col("amount") <= F.lit(0))
        .withColumn("err_ccy_invalid", ~F.col("currency").isin(ALLOWED_CCY))
        .withColumn("err_ccy_null", F.col("currency").isNull())
        .withColumn("err_date_null", F.col("trx_date").isNull())
        .withColumn("err_trx_null", F.col("trx_id").isNull())
        .withColumn("err_cp_null", F.col("counterparty_id").isNull())
    )

    df_flags = df_flags.withColumn(
        "is_invalid",
        F.col("err_amount_null")
        | F.col("err_amount_nonpos")
        | F.col("err_ccy_invalid")
        | F.col("err_ccy_null")
        | F.col("err_date_null")
        | F.col("err_trx_null")
    )

    invalid = df_flags.filter(F.col("is_invalid") == True)
    valid = df_flags.filter(F.col("is_invalid") == False)

    valid2 = (valid
        .withColumn(
            "counterparty_id",
            F.when(F.col("counterparty_id").isNull(), F.lit("UNKNOWN"))
             .otherwise(F.col("counterparty_id"))
```

```
        )
        .withColumn("imputed_counterparty",
                    F.when(F.col("err_cp_null"), F.lit(1)).otherwise(F.lit(0)))
    )

    valid2.write.mode("overwrite").parquet(enriched_path)
    invalid.write.mode("overwrite").parquet(reject_path)

    report = (df_flags.agg(
        F.count("*").alias("rows_total"),
        F.sum(F.col("is_invalid").cast("int")).alias("rows_rejected"),
        F.sum(F.col("err_cp_null").cast("int")).alias("rows_imputed"),
        F.sum(F.col("err_amount_null").cast("int")).alias("err_amount_null"),
        F.sum(F.col("err_amount_nonpos").cast("int")).alias("err_amount_nonpos"),
        F.sum(F.col("err_ccy_invalid").cast("int")).alias("err_ccy_invalid"),
        F.sum(F.col("err_ccy_null").cast("int")).alias("err_ccy_null"),
        F.sum(F.col("err_date_null").cast("int")).alias("err_date_null"),
        F.sum(F.col("err_trx_null").cast("int")).alias("err_trx_null")
    ))

    report.write.mode("overwrite").json(dq_report_path)
    spark.stop()

if __name__ == "__main__":
    run(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
```

Лістинг А.5 – Job 3: побудова агрегатів і вітрин (PySpark)

```
# jobs/job3_build_aggregates.py
import sys
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

def run(enriched_path: str, mart_path: str, year: int, month: int) -> None:
    spark = SparkSession.builder.appName("Job3_BuildAggregates").getOrCreate()

    df = spark.read.parquet(enriched_path)

    agg = (df.groupBy("counterparty_id")
             .agg(
                 F.sum("amount").alias("sum_amount"),
                 F.count("*").alias("cnt_ops")
             )
             .withColumn("year", F.lit(int(year)))
             .withColumn("month", F.lit(int(month)))
    )

    agg.write.mode("overwrite").parquet(mart_path)
    spark.stop()

if __name__ == "__main__":
    run(sys.argv[1], sys.argv[2], int(sys.argv[3]), int(sys.argv[4]))
```

Лістинг А.6 – Формування ключа HBase та запис (приклад для прототипу)

```
# storage/hbase_writer.py
import hashlib
import happybase

def _salt(entity_id: str, buckets: int = 16) -> str:
    h = int(hashlib.md5(entity_id.encode("utf-8")).hexdigest(), 16)
    return f"{h % buckets:02d}"

def make_rowkey(entity_id: str, yyyymmdd: str, trx_id: str, version: str = "v1") ->
```

```
bytes:
    rk = f"{_salt(entity_id)}|{entity_id}|{yyyymmdd}|{trx_id}|{version}"
    return rk.encode("utf-8")

def put_tx(host: str, entity_id: str, yyyymmdd: str, trx_id: str, amount: float,
currency: str) -> None:
    conn = happybase.Connection(host)
    table = conn.table("finance_tx")

    rk = make_rowkey(entity_id, yyyymmdd, trx_id)
    table.put(rk, {
        b"d:entity_id": entity_id.encode("utf-8"),
        b"d:trx_date": yyyymmdd.encode("utf-8"),
        b"d:trx_id": trx_id.encode("utf-8"),
        b"d:amount": str(amount).encode("utf-8"),
        b"d:currency": currency.encode("utf-8"),
    })

    conn.close()
```

Лістинг А.7 – Файл залежностей (requirements.txt)

```
fastapi==0.110.0
uvicorn==0.27.1
pydantic==2.6.1
happybase==1.2.0
```

Лістинг А.8 – Приклад запуску (команди)

```
# запуск API
uvicorn api.app:app --host 0.0.0.0 --port 8000

# ручний запуск job-ів (поза API)
spark-submit jobs/job1_ingest_normalize.py \
  /finance/raw/source=bank_a/dt=2025-12-01/ \
  /finance/processed/schema=v1/year=2025/month=12/

spark-submit jobs/job2_dq_missing.py \
  /finance/processed/schema=v1/year=2025/month=12/ \
  /finance/enriched/schema=v1/year=2025/month=12/ \
  /finance/reject/batch_id=2025-12_bank_a/ \
  /finance/dq/batch_id=2025-12_bank_a/

spark-submit jobs/job3_build_aggregates.py \
  /finance/enriched/schema=v1/year=2025/month=12/ \
  /finance/mart/report=agg_by_counterparty/year=2025/month=12/ \
  2025 12
```

Додаток Б

Апробація отриманих результатів

Почесні співголови конференції:

Десятнюк О.М., ректор Західноукраїнського національного університету, д-р економічних наук, професор;

Дивак М.П., д-р технічних наук, професор, проректор з наукової роботи ЗУНУ;

Голова конференції:

Березький О.М., д-р технічних наук, професор, професор кафедри комп'ютерної інженерії Західноукраїнський національний університет;

Заступники голови конференції:

Мельник Г.М., к.т.н, доцент, Західноукраїнський національний університет;

Піцун О.Й., к.т.н. доцент, Західноукраїнський національний університет;

ПРОГРАМНИЙ КОМІТЕТ КОНФЕРЕНЦІЇ

Склад програмного комітету:

Семанюк В.З., д-р економічних наук, професор, начальник науково-дослідної частини Західноукраїнського національного університету

Антощук С.Г., д.т.н, професор, Національний університет «Одеська політехніка»;

Баловсяк С.В., д.т.н, професор, Чернівецький національний університет

Бармак О.В., д.т.н., професор, Хмельницький національний університет

Батько Ю.М., к.т.н., доцент, Західноукраїнський національний університет

Березька К.М.,к.т.н.,доцент, Західноукраїнський національний університет

Винокурова О.А., д.т.н., професор, Львівський національний університет імені Івана Франка

Возна Н.Я., д.т.н., професор, Західноукраїнський національний університет;

Говорущенко Т.О., д.т.н., професор, Хмельницький національний університет;

Дубчак Л.О., к.т.н., доцент, Західноукраїнський національний університет;

Дунець Р.Б., д.т.н., професор, НУ "Львівська політехніка";

Ізонін І.В., д.т.н., доцент, НУ "Львівська політехніка";

Комар М.П., д.т.н, професор, Західноукраїнський національний університет;

Литвиненко В.І., д.н.т, професор, Національний технічний університет України "Київський політехнічний інститут";

Лупенко С.А., д.т.н., професор, Опольський технологічний університет, Польща;

Лящинський П.Б., доктор філософії з комп'ютерних наук, НУ "Львівська політехніка" ;

Мельникова Н.І., д.т.н., професор, НУ "Львівська політехніка" ;

Мулеса О.Ю., д.т.н., професор, Пряшівський університет, Словаччина

Пелешко Д.Д., д.т.н., професор, Львівський національний університет імені Івана Франка;

Сельський П.Р., д.м.н., професор, Тернопільський національний медичний університет імені І. Я. Горбачевського ;
Субботін С.О., д.т.н., професор, Національний університет «Запорізька політехніка» ;
Теслюк В.М., д.т.н., професор, НУ "Львівська політехніка" ;
Тимченко Л.І., д.т.н., професор, Державний університет інфраструктури та технологій;
Цмоць І.Г., д.т.н., професор, НУ "Львівська політехніка" ;
Якименко І.З., к.т.н., доцент, Західноукраїнський національний університет;
Яровий А.А., д.т.н., професор, Вінницький національний технічний університет ;
Яцків В.В., д.т.н., професор, Західноукраїнський національний університет.

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ КОНФЕРЕНЦІЇ

Склад організаційного комітету:

Мельник Г.М. к.т.н., доцент, Західноукраїнський національний університет
Піцун О.Й. к.т.н., доцент, Західноукраїнський національний університет

Технічна підтримка:

Зінькевич О. В. студентка, Західноукраїнський національний університет
Кіт М. О. студентка, Західноукраїнський національний університет
Лебединський Т.В. студент,Західноукраїнський національний університет;
Кришталь Е.Р., студент, Західноукраїнський національний університет;

Метою конференції є представлення та обговорення наукових і практичних результатів, сприяння активізації творчої і інноваційної діяльності студентів, аспірантів і молодих вчених.
Конференція проводиться із залученням Ради Молодих Вчених та Студентського Наукового Товариства ФКІТ ЗУНУ.

IV Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі» (ІКСМ весна 2026), м. Тернопіль, ЗУНУ, 29 травня 2026 р. Тернопіль, 2026

Адреса:
Вул. О. Теліги, 8, корпус №6, Тернопіль
URL: https://ki.wunu.edu.ua/conference/
e-mail: kistudconference@gmail.com

Тези доповідей подаються за оригіналом рукопису

перевірок. Також визначено два можливі режими роботи модуля: пакетний і потоковий. Пакетний режим є простішим для реалізації та експериментальної перевірки, тоді як потоковий режим краще відповідає умовам реального смарт-обліку, де дані надходять безперервно. Водночас потокова реалізація потребує додаткової уваги до синхронізації часу, затримок, пропусків і стабільності обробки.

Список літератури

1. Kabalci Y. A survey on smart metering and smart grid communication. Renewable and Sustainable Energy Reviews. 2016. Vol. 57. P. 302–318.
2. Zeng Yan, Ye Xiaoying, Wang Kang, Gao Yanhao, Zheng Hongliang. Design of an intelligent unattended metering system. Industrial Technology and Vocational Education. 2024. Vol. 22, no. 5. P. 1–5.
3. Ji X., Li T. Analysis on Safety Performance Improvement of Smart Metering System Based on Big data. Procedia Computer Science. 2025. Vol. 262. P. 909–918.
4. Wang J., Yang Y., Wang T., et al. Big data service architecture: a survey. Journal of Internet Technology. 2020. Vol. 21, no. 2. P. 393–405.
5. Zhang Yongmin, Liu Shengnan, Yi Ling, An Jiyuan. Power metering system based on filtering to avoid power theft behavior dynamic identification method. Mechanical Design and Manufacturing Engineering. 2024. Vol. 53, no. 3. P. 129–132.
6. Breiman L. Random Forests. Machine Learning. 2001. Vol. 45, no. 1. P. 5–32.
7. Chen T., Guestrin C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016. P. 785–794.
8. Ke G., Meng Q., Finley T., et al. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems (NeurIPS). 2017.
9. Friedman J. H. Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics. 2001. Vol. 29, no. 5. P. 1189–1232.

Ярунів М.А.
студент 4 курсу ФКІТ ЗУНУ
Науковий керівник д.т.н., професор Комар М.П., кафедра ІОСУ ЗУНУ

ПРОГРАМНИЙ МОДУЛЬ РОЗПОДІЛЕНОГО ЗБЕРІГАННЯ ТА ПАРАЛЕЛЬНОЇ ОБРОБКИ ФІНАНСОВИХ ДАНИХ

Вступ. Цифрова трансформація фінансового сектору супроводжується стрімким зростанням обсягів даних, які формуються під час виконання платіжних операцій, бухгалтерського та управлінського обліку, бюджетування, ризик-менеджменту, взаємодії з клієнтами та зовнішніми інформаційними ресурсами.

У таких умовах зростає роль технологій Big Data у фінансах, оскільки саме вони забезпечують можливість отримання своєчасних аналітичних висновків на основі великих масивів гетерогенних даних [1]. Водночас упровадження аналітики великих даних у банківських і фінансових установах потребує узгоджених організаційних та технологічних рішень, зокрема щодо управління даними, регламентів доступу, якості та життєвого циклу наборів даних [2, 3].

Постановка задачі. Цифрова трансформація фінансового сектору супроводжується стрімким зростанням обсягів даних, які формуються під час виконання платіжних операцій, бухгалтерського та управлінського обліку, бюджетування, ризик-менеджменту, взаємодії з клієнтами та зовнішніми інформаційними ресурсами. У таких умовах зростає роль технологій Big Data у фінансах, оскільки саме вони забезпечують можливість отримання своєчасних аналітичних висновків на основі великих масивів гетерогенних даних [1]. Водночас упровадження аналітики великих даних у банківських і фінансових установах потребує

узгоджених організаційних та технологічних рішень, зокрема щодо управління даними, регламентів доступу, якості та життєвого циклу наборів даних [2, 3].

Об'єкт дослідження – процеси зберігання та обробки фінансових даних у хмарному та кластерному середовищі.

Предмет дослідження – методи та програмні засоби розподіленого зберігання і паралельної обробки фінансових даних, а також підходи до забезпечення узгодженості, захисту і якості даних.

Мета роботи полягає у проєктуванні програмного модуля розподіленого зберігання та паралельної обробки фінансових даних, який забезпечує масштабованість, відмовостійкість, підвищену продуктивність аналітичних операцій, а також базові механізми захисту та контролю якості даних.

Основний матеріал. Архітектура програмного модуля розподіленого зберігання та паралельної обробки фінансових даних формується з урахуванням вимог продуктивності, масштабованості, відмовостійкості, узгодженості та безпеки [4]. У фінансових системах одночасно існують два типи навантаження: операційне (швидкі вибірки за ключовими атрибутами) та аналітичне (масові сканування, агрегації, консолідація). Поєднання цих навантажень у межах одного централізованого сховища часто призводить до зростання затримок і ускладнює масштабування [4]. Тому доцільним є модульний підхід, у якому розподілене зберігання і паралельна обробка реалізуються як узгоджені компоненти єдиного контуру даних [5].

Програмний модуль розглядається як інфраструктурний компонент між джерелами фінансових даних (ERP/CRM, транзакційні журнали, звітність, зовнішні показники) та споживачами результатів (BI, звітні підсистеми, прикладні сервіси аналітики). На концептуальному рівні модуль реалізує послідовність етапів: приймання даних → валідація/нормалізація → розподілене зберігання → паралельна обробка → публікація результатів [5]. Така модель забезпечує відокремлення операційного та аналітичного контурів і дозволяє зберігати історичні масиви без погіршення доступу до "гарячих" даних.

Архітектура модуля поділяється на три рівні (рисунок 1).

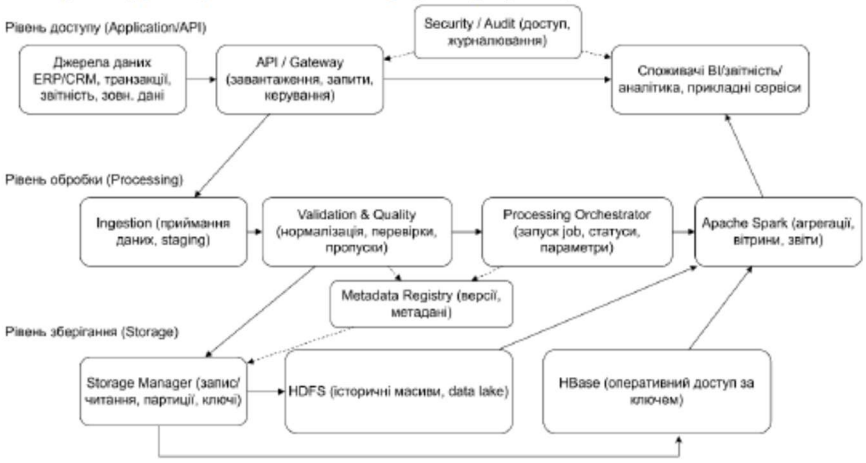

Рисунок 1 – Загальна архітектурна схема програмного модуля: взаємодія рівнів та основних компонентів

1. Рівень зберігання (Storage Layer). Для довготривалого зберігання великих обсягів даних використовується HDFS як розподілена файлова система з блочною організацією та реплікацією. Для оперативного доступу за ключем застосовується HBase як колоночне NoSQL-

сховище поверх HDFS. Така комбінація забезпечує одночасно високу пропускну здатність при скануванні історичних даних і можливість швидких вибірок для сервісних сценаріїв [6, 7].

2. Рівень обробки (Processing Layer). Паралельне виконання обчислювальних задач реалізується на Apache Spark, який підтримує кластерні обчислення, DAG-планування та оптимізацію повторних обробок за рахунок роботи з робочими наборами даних. Spark використовується для агрегацій, формування вітрин, регламентних розрахунків, перевірок якості та підготовки даних [5].

3. Прикладний рівень доступу (Application/API Layer). Призначений для взаємодії із зовнішніми системами: керування завантаженнями, запуск задач, отримання результатів, перегляд статусів і протоколів перевірок. Наявність такого рівня узгоджується з практикою впровадження Big Data аналітики у фінансах, де важливими є керованість, прозорість обчислень і відтворюваність процедур [2, 3].

Середовище розгортання визначається потребою в горизонтальному масштабуванні та керованості ресурсів. Для цього застосовується кластерна модель, яка може бути реалізована як у локальному дата-центрі, так і у хмарній інфраструктурі. Хмарні обчислення характеризуються наданням ресурсів "на вимогу", широким мережевим доступом та еластичним масштабуванням, що створює передумови для ефективної обробки великих фінансових масивів при змінному навантаженні. У межах модуля це означає можливість збільшення ресурсів під час масових регламентних розрахунків і зменшення у періоди низької активності.

Водночас хмарний контекст підсилює вимоги до безпеки й контролю доступу. Для фінансових даних необхідно передбачити криптографічний захист і аудит операцій. Як базовий стандарт шифрування розглядається AES відповідно до FIPS 197 [8], а також протоколювання доступу і критичних дій у системі [9]. Концептуально коректність узгодженого стану в розподіленому середовищі пов'язується з ідеями консенсусу, що обґрунтовуються протоколами на кшталт Paxos [10]. На практичному рівні це відображається у вимогах до версіонування наборів даних, контрольованих повторних запусків job-ів і фіксації метаданих для відтворюваності результатів [3, 10].

Архітектура модуля формується як набір компонентів із чітко визначеними функціями та інтерфейсами:

1. Ingestion-компонент. Приймає дані (файли/повідомлення), виконує первинну перевірку формату, фіксує метадані (джерело, час, тип набору) та розміщує дані у staging-зоні [3].

2. Компонент підготовки та контролю якості. Реалізує валідацію схеми, нормалізацію дат і валют, перевірку ключів, дедуплікацію, а також формує звіт про дефекти. Окремо передбачається виявлення пропусків і, за визначеними правилами, їх оброблення або імпутація, якщо це узгоджено з регламентом [11].

3. Storage Manager. Організує запис даних у HDFS (історичні, "холодні" набори) та у HBase (оперативні вибірки за ключем). Важливою умовою є коректне партиціювання і дизайн ключів, оскільки це впливає на рівномірність навантаження та ризик "гарячих точок". В якості концептуальної основи вирівнювання розподілу ключів може використовуватися consistent hashing [6, 7].

4. Processing Orchestrator. Керує запуском Spark job-ів, передає параметри виконання (період, набір даних, перелік метрик), отримує статуси та зберігає результати у визначених форматах. Для підвищення продуктивності враховуються типові оптимізації Spark (розподіл партицій, мінімізація shuffle, використання кешування) [12].

5. Реєстр метаданих. Зберігає інформацію про версії датасетів, параметри запусків, результати перевірок якості, місця розміщення наборів у HDFS/HBase та технічні атрибути обробки. Такий реєстр є необхідним для data governance і відтворюваності розрахунків [2, 3].

6. Security/Audit-компонент. Забезпечує автентифікацію/авторизацію, розмежування прав доступу, журналювання та аудит критичних операцій, а також підтримку шифрування (за потреби) [8].

7. API-шар. Надає уніфікований інтерфейс для завантаження, запуску задач, отримання результатів, метрик і протоколів якості. Це підвищує керованість та спрощує інтеграцію зі

споживачами результатів [3].

Зазначені компоненти взаємодіють за принципом конвеєра: ingestion передає дані на підготовку, підготовка – в storage, storage забезпечує доступ обробці, Spark формує результати, а API надає результат і метадані зовнішнім системам. Такий підхід відповідає інженерним уявленням про проєктування розподілених сховищ і систем обробки великих даних, де важливою є декомпозиція на керовані компоненти [5].

Висновки. Запропонована архітектура модуля має трирівневу структуру. Рівень зберігання забезпечує роботу з великими історичними масивами та швидкий доступ до окремих записів. Рівень обробки відповідає за паралельні обчислення, агрегації, формування вітрин і перевірки якості. Прикладний рівень доступу забезпечує взаємодію із зовнішніми системами, запуск задач, отримання результатів і перегляд службової інформації.

Обгрунтовано доцільність використання HDFS для довготривалого розподіленого зберігання фінансових даних, HBase – для оперативного доступу за ключем, Apache Spark – для паралельної обробки великих масивів даних. Така комбінація дозволяє відокремити аналітичні й операційні навантаження та зменшити обмеження, характерні для централізованих сховищ.

Окрему увагу приділено питанням якості, безпеки та керованості даних. Передбачено валідацію, нормалізацію, дедуплікацію, оброблення пропусків, фіксацію метаданих, аудит дій і криптографічний захист. Це забезпечує прозорість обробки, можливість повторного виконання задач і підвищує надійність результатів фінансової аналітики.

Отже, запропонований підхід формує цілісний технологічний контур роботи з фінансовими даними: від їх приймання та підготовки до зберігання, паралельної обробки й надання результатів споживачам. Розроблена архітектура може бути використана як основа для створення масштабованих інформаційних систем фінансової аналітики, звітності та підтримки управлінських рішень.

Список літератури

1. Goldstein I., Spatt C. S., Ye M. Big data in finance. The Review of Financial Studies. 2021. Vol. 34, no. 7. P. 3213–3225.
2. Al-Afeef M., Ali O., Al-Tahat S. et al. The effect of big data governance on financial technology in Jordanian commercial banks: The mediation role of organizational culture. International Journal of Data and Network Science. 2023. Vol. 7, no. 3. P. 1283–1294.
3. Hajiheydari N., Delgosha M. S., Wang Y. et al. Exploring the paths to big data analytics implementation success in banking and financial service: an integrated approach. Industrial Management & Data Systems. 2021. Vol. 121, no. 12. P. 2498–2529.
4. Legault M. A practitioner's view on distributed storage systems: Overview, challenges and potential solutions. Technology Innovation Management Review. 2021. Vol. 11, no. 6. P. 32–41.
5. Zhang Xin. Distributed Storage and Parallel Processing Technology of Financial Big Data under Cloud Computing Platform. Procedia Computer Science. 2025. Vol. 262. P. 714–721.
6. Apache Software Foundation. Apache Hadoop: HDFS Architecture Guide [Електронний ресурс]. Режим доступу: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (дата звернення: 24.01.2026).
7. Apache Software Foundation. Apache HBase® Reference Guide [Електронний ресурс]. Режим доступу: https://hbase.apache.org/book.html (дата звернення: 24.01.2026).
8. National Institute of Standards and Technology. FIPS PUB 197: Advanced Encryption Standard (AES). Gaithersburg, MD : NIST, 2001.
9. Regin R., Rajest S. S., Shynu T. A review of secure neural networks and big data mining applications in financial risk assessment. Central Asian Journal of Innovations on Tourism Management and Finance. 2023. Vol. 4, no. 2. P. 73–90.
10. Lamport L. Paxos Made Simple. ACM SIGACT News (Distributed Computing Column). 2001. Vol. 32, no. 4. P. 51–58.
11. Wang C., Shakhovska N., Sachenko A., Komar M. A New Approach for Missing Data Imputation in Big Data Interface. Information Technology and Control. – 2020. – Vol. 49. No 4. – Pp. 541-555.
12. Zaharia M., Chowdhury M., Franklin M. J., Shenker S., Stoica I. Spark: Cluster Computing with Working Sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10). 2010. 10 p.