НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет/(ННІ) інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету (Директор ННІ)	**Завідувач кафедри**
інформаційних технологій	комп'ютерних наук
(назва факультету (ННІ))	(назва кафедри)
_____ Ігор Болбот	_____ Белла Голуб
(підпис) (ім'я ПРІЗВИЩЕ)	(підпис) (ім'я ПРІЗВИЩЕ)
"___" _____ 20_ р.	"___" _____ 20_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему **Застосування алгоритмів паралельної обробки інформації в системах високопродуктивних рішень**

Спеціальність 121 "Інженерія програмного забезпечення"
(код і найменування)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
к.т.н., доцент _____ Белла Голуб
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
д.т.н., проф. _____ Хиленко В.В.
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ Денис НЄВРОВ
(підпис) (ім'я ПРІЗВИЩЕ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) __інформаційних технологій_____

ЗАТВЕРДЖУЮ
Завідувач кафедри__комп'ютерних наук_
___доцент, к.т.н._____ _____ _Белла Голуб
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"01" листопада 2024 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____Нєвров Денис Олександрович_____
(прізвище, ім'я, по батькові)

Спеціальність 121 «Інженерія програмного забезпечення»
(код і назва)

Освітня програма ____Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми_____освітньо-професійна _____

Тема магістерської кваліфікаційної роботи Застосування алгоритмів паралельної обробки інформації в системах високопродуктивних рішень
затверджена наказом ректора НУБіП України від "_1_" _листопада__2024р. №1964 «С»
Термін подання завершеної роботи на кафедру 11 листопада 2025_____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: Нормативно-довідкова та наукова література з інформаційних систем, баз даних, методів візуалізації та обробки даних; технічна документація з проєктування та розробки програмних комплексів; програмні засоби для розробки веб-застосунків і інтерактивних інтерфейсів; вимоги стандартів до побудови інформаційних систем і користувацьких інтерфейсів.

Перелік питань, що підлягають дослідженню:

1. Системний аналіз предметної області
2. Моделювання та архітектурне проєктування системи
3. Реалізація програмного забезпечення та технологічна інфраструктура системи
4. Тестування та оцінювання ефективності системи

Перелік графічного матеріалу (за потреби) презентація, постер, схеми та діаграми архітектури системи

Дата видачі завдання "__1__" листопада 2024 р.

Керівник магістерської кваліфікаційної роботи _____ ___Хиленко В.В.
(підпис) (ім'я ПРІЗВИЩЕ))

Завдання прийняв до виконання _____ Денис НЄВРОВ
(підпис) (ім'я ПРІЗВИЩЕ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

– API (Application Programming Interface) — інтерфейс прикладного програмування, що забезпечує взаємодію між програмними компонентами.

– ARIMA / SARIMA — авторегресійна інтегрована модель ковзного середнього (сезонна модифікація) для прогнозування часових рядів.

– CVaR (Conditional Value-at-Risk) — умовне математичне очікування втрат, що перевищують певний поріг ризику (VaR).

– CRUD (Create, Read, Update, Delete) — базові операції з даними у реляційних системах управління базами даних.

– CSV (Comma-Separated Values) — текстовий формат представлення табличних даних, у якому значення розділяються комами.

– DSS (Decision Support System) — система підтримки прийняття рішень, призначена для аналітичної обробки даних і формування управлінських висновків.

– ETL (Extract, Transform, Load) — процес вилучення, перетворення та завантаження даних у сховище.

– GBM (Gradient Boosting Machine) — метод ансамблевого навчання на основі градієнтного бустингу дерев рішень.

– GUI (Graphical User Interface) — графічний інтерфейс користувача.

– JSON (JavaScript Object Notation) — текстовий формат обміну даними між клієнтом і сервером.

– KPI (Key Performance Indicator) — ключовий показник ефективності, що характеризує результативність діяльності компанії.

– MAPE (Mean Absolute Percentage Error) — середня абсолютна відносна похибка прогнозування.

– MILP (Mixed Integer Linear Programming) — змішане цілочисельне лінійне програмування.

– ML (Machine Learning) — машинне навчання, підрозділ штучного інтелекту, що забезпечує автоматичне виявлення закономірностей у даних.

– NDCG (Normalized Discounted Cumulative Gain) — метрика оцінювання якості ранжування рекомендаційних моделей.

– OLAP (Online Analytical Processing) — технологія оперативного аналітичного оброблення даних із багатовимірною структурою.

– PCA (Principal Component Analysis) — метод головних компонент, який використовується для зниження розмірності даних.

– REST (Representational State Transfer) — архітектурний стиль веб-сервісів, що базується на HTTP-протоколі.

– RMSE (Root Mean Squared Error) — середньоквадратична похибка прогнозування.

– SQL (Structured Query Language) — структурована мова запитів до баз даних.

– TLS (Transport Layer Security) — криптографічний протокол для захищеного передавання даних у мережі.

– UI (User Interface) — інтерфейс користувача, засіб взаємодії користувача із системою.

– VaR (Value-at-Risk) — максимальні можливі втрати при заданому рівні довіри за визначений період.

– XGBoost (Extreme Gradient Boosting) — алгоритм градієнтного бустингу з оптимізованою продуктивністю.

ВСТУП

Сучасні високопродуктивні обчислювальні системи (High Performance Computing, HPC) є фундаментом для розвитку обчислювальної науки, штучного інтелекту, промислової аналітики, симуляцій складних процесів і обробки великих даних. Ефективність їх функціонування безпосередньо залежить від застосування алгоритмів **паралельної обробки інформації**, які забезпечують раціональний розподіл навантаження, синхронізацію процесів та зниження часу виконання задач [1]. З огляду на швидке зростання обсягів даних і складності моделей, розроблення нових підходів до динамічного планування та балансування ресурсів стає ключовою умовою підвищення продуктивності HPC-систем [2].

Актуальність теми зумовлена потребою створення інтелектуальних алгоритмів паралельної обробки, що поєднують високу масштабованість із гнучким розподілом задач між вузлами кластеру. Такі системи дозволяють досягти оптимальної швидкодії при змінних навантаженнях, мінімізувати затримки синхронізації та підвищити відмовостійкість інфраструктури. Розробка подібних рішень є актуальною для сучасних наукових обчислень, інженерного моделювання та обробки даних у режимі реального часу [3].

Метою роботи є розроблення та дослідження системи паралельної обробки інформації із застосуванням алгоритмів розподілених обчислень, динамічного планування та моніторингу продуктивності з метою підвищення ефективності високопродуктивних рішень.

Для успішного виконання поставленої мети потрібно вирішити наступні **задачі:**

– провести системний аналіз предметної області паралельних обчислень і сучасних HPC-архітектур;

– дослідити алгоритми MPI, OpenMP та паралельного I/O для реалізації розподілених задач;

– розробити архітектуру системи високопродуктивних обчислень із підтримкою адаптивного балансування;

– побудувати UML-моделі (прецедентів, послідовності, активності, класів і компонентів);

– реалізувати програмні модулі розподілу задач і моніторингу навантаження;

– провести тестування продуктивності, масштабованості та стійкості системи до збоїв.

Об’єкт дослідження – процеси паралельної обробки інформації у високопродуктивних обчислювальних системах.

Предмет дослідження – алгоритми планування, синхронізації, балансування навантаження й оптимізації обчислювальних процесів у розподілених середовищах.

Методологічна основа дослідження ґрунтується на поєднанні теоретичних положень інформатики, теорії паралельних і розподілених алгоритмів, обчислювальної математики та системного аналізу. Для розроблення й перевірки моделі використано методи UML-моделювання (структурні й поведінкові діаграми), теорію графів і черг, методи оцінки масштабованості (закони Амдала й Густафсона), а також методи експериментального аналізу продуктивності. У практичній частині застосовано мови Java (для серверної логіки) та Python (для аналітичних модулів), а також бібліотеки SQLite JDBC**,** MPI4Py**,** NumPy**,** Matplotlib і JUnit для тестування.

Наукова новизна полягає у створенні адаптивної моделі управління обчислювальними процесами, яка динамічно змінює схеми планування й синхронізації залежно від навантаження системи. Запропоновано комбінований підхід, що поєднує механізми MPI-комунікацій з аналітичним моніторингом стану вузлів і дозволяє зменшити затримки між процесами без втрати точності розрахунків.

Практична цінність роботи полягає у можливості застосування розробленої системи в наукових, навчальних і промислових HPC-кластерах для

підвищення продуктивності паралельних задач, зокрема при аналізі великих даних, симуляції фізичних процесів і навчанні моделей штучного інтелекту. Розроблені програмні модулі можуть бути інтегровані у хмарні обчислювальні середовища з мінімальними змінами архітектури.

Структура роботи. Магістерська робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків.

У першому розділі подано системний аналіз предметної області, класифікацію алгоритмів паралельної обробки та формалізацію вимог до системи.

У другому розділі розроблено UML-моделі, які відображають логіку функціонування, структуру компонентів і взаємодію процесів.

Третій розділ присвячено розробленню програмного забезпечення, реалізації архітектури, бази даних і ключових алгоритмів системи.

Четвертий розділ містить результати тестування системи, оцінку ефективності паралельного виконання, масштабованості та відмовостійкості.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

У сучасних високопродуктивних обчислювальних (HPC) системах відбувається інтенсивна еволюція методів паралельної обробки даних, зумовлена потребою в оперативному аналізі великих потоків інформації, чисельному моделюванні та навчанні складних моделей машинного навчання [1]. Предметна область дослідження охоплює принципи організації паралельних процесів, використання різних архітектур (CPU, GPU, кластери) та оптимізацію показників масштабованості. Структурну схему взаємозв'язків основних елементів предметної області наведено на рис. 1.1, де відображено основні компоненти – типові задачі, навантаження, платформи, шаблони обчислень і показники ефективності.

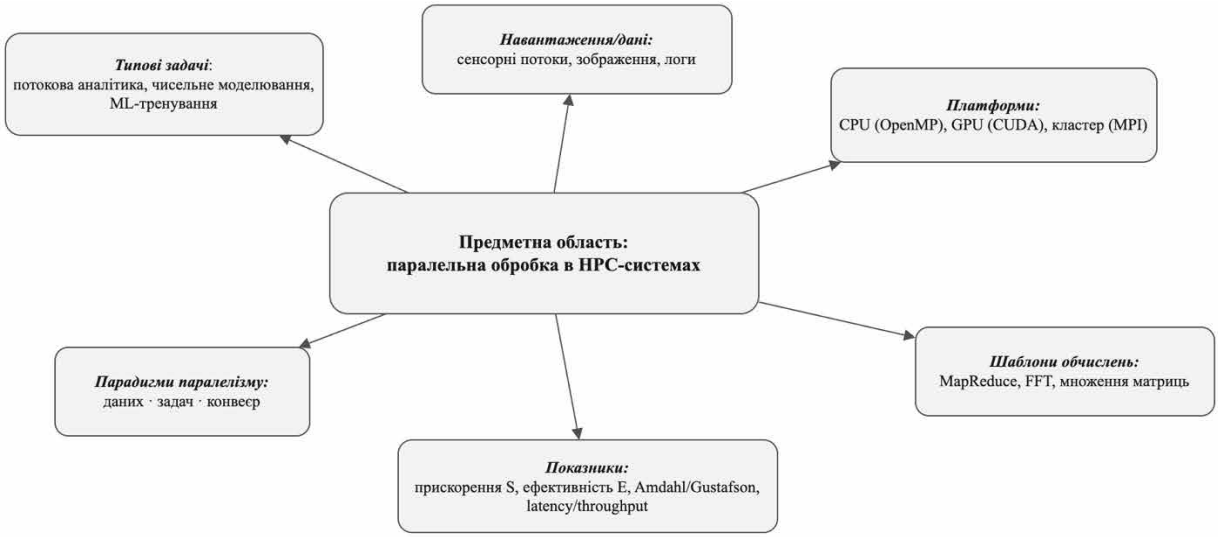

Рис. 1.1 – Структурна схема предметної області паралельної обробки в HPC-системах

Предметна область включає типові задачі потокової аналітики, чисельного моделювання та ML-тренування, що реалізуються в умовах великих масивів даних і потребують багаторівневої паралелізації. Згідно з парадигмами паралелізму (рис. 1.1), реалізація може бути орієнтована на дані, задачі або конвеєр, залежно від типу навантаження та обраної

обчислювальної моделі. Основними платформами для таких рішень є CPU-архітектури з OpenMP, GPU-платформи з CUDA, а також багатовузлові кластери з протоколом MPI, які забезпечують масштабування на рівні сотень процесорів.

Для практичної реалізації алгоритмів часто використовують стандартні шаблони обчислень – MapReduce, FFT, множення матриць – які характеризуються передбачуваною структурою потоків даних. Основні показники ефективності системи – прискорення S, ефективність E та пропускна здатність (throughput), що визначаються співвідношенням часу виконання паралельного та послідовного процесів і враховують закони Амдала та Густафсона.

На основі аналітичних і експериментальних вимірювань побудовано графік масштабованості (рис. 1.2), який демонструє залежність прискорення S(p) від кількості процесів p. Як видно, експериментальна крива розміщується нижче теоретичних оцінок через накладні витрати синхронізації та обмеження пропускної здатності мережі.

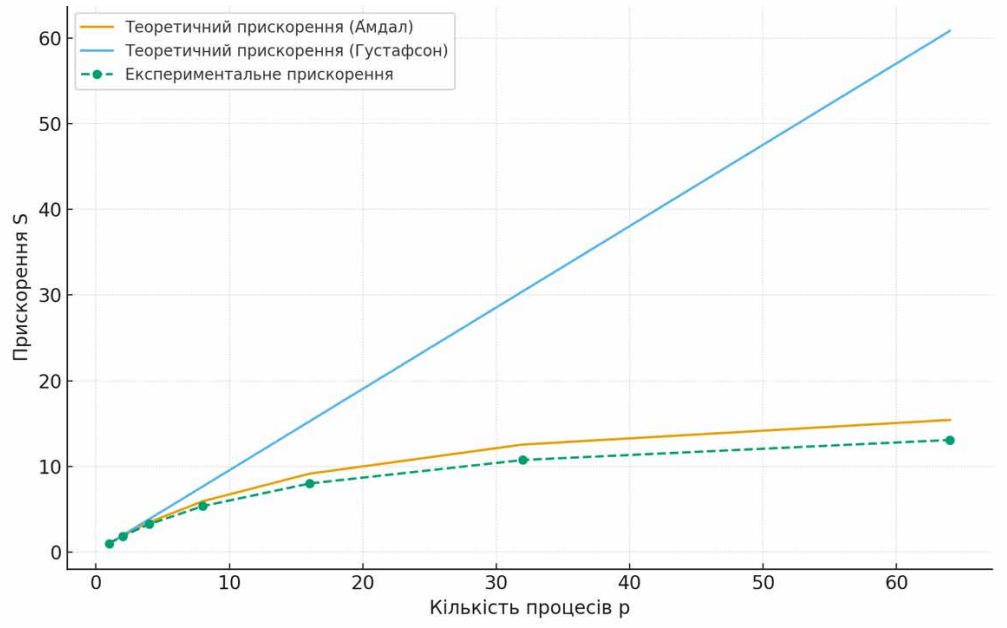

Рис. 1.2 – Графік масштабованості HPC-завдання за кількістю процесів p*

Для узагальнення характеристик паралельного середовища у табл. 1.1 наведено порівняння типів платформ HPC-систем, їх ключових параметрів і можливостей масштабування.

Таблиця 1.1 – Основні характеристики HPC-платформ

Платформа	Тип паралелізму	Інтерфейс / API	Основна область застосування	Особливості масштабування
CPU (OpenMP)	Паралелізм потоків	OpenMP, POSIX threads	Наукові обчислення, аналітика	Обмежена внутрішнім числом ядер
GPU (CUDA)	Даних / векторний	CUDA, OpenCL	ML-тренування, візуалізація	Висока локальна масштабованість
MPI-кластер	Задач / вузлів	MPI, OpenMPI	Розподілені симуляції, HPC-моделі	Масштабування до тисяч процесів
Гібридна (CPU+GPU)	Змішаний	MPI+OpenMP/CUDA	Моделювання фізичних процесів	Балансування між типами пристроїв

Предметна область паралельної обробки в HPC-системах охоплює взаємопов’язані аспекти архітектури, парадигм і показників ефективності, що забезпечують можливість адаптивного масштабування обчислень залежно від задачі та ресурсу системи. Це формує теоретичну базу для подальшої розробки моделі оптимізації паралельних процесів і побудови високопродуктивного програмного комплексу.

1.2 Теоретико-методологічні засади та стан наукових досліджень

Теоретико-методологічна основа паралельних обчислень ґрунтується на концепціях розподіленого виконання задач**,** балансування навантаження та моделювання продуктивності**,** які формалізуються через рівняння масштабованості та часові моделі взаємодії процесів. У сучасній HPC-парадигмі центральним є принцип розподілу великих задач на множину дрібніших підзадач, що виконуються паралельно на різних обчислювальних вузлах із мінімізацією комунікаційних витрат [5].

Наукові дослідження в галузі високопродуктивних обчислень активно розвивають теорії моделей продуктивності**,** серед яких найбільш поширені Roofline-модель (рис. 1.3) і логічні схеми взаємодії процесів (рис. 1.4). Roofline-підхід дозволяє оцінити граничну ефективність системи, виходячи з обмежень пропускної здатності пам’яті (BW) та пікової обчислювальної потужності (P_{peak}). На графіку (рис. 1.3) чітко розрізняються дві зони: область пам’яттєвих обмежень при низькій інтенсивності I (операцій на байт) та область обчислювальних лімітів при високих I, де продуктивність обмежується лише архітектурним піком системи.

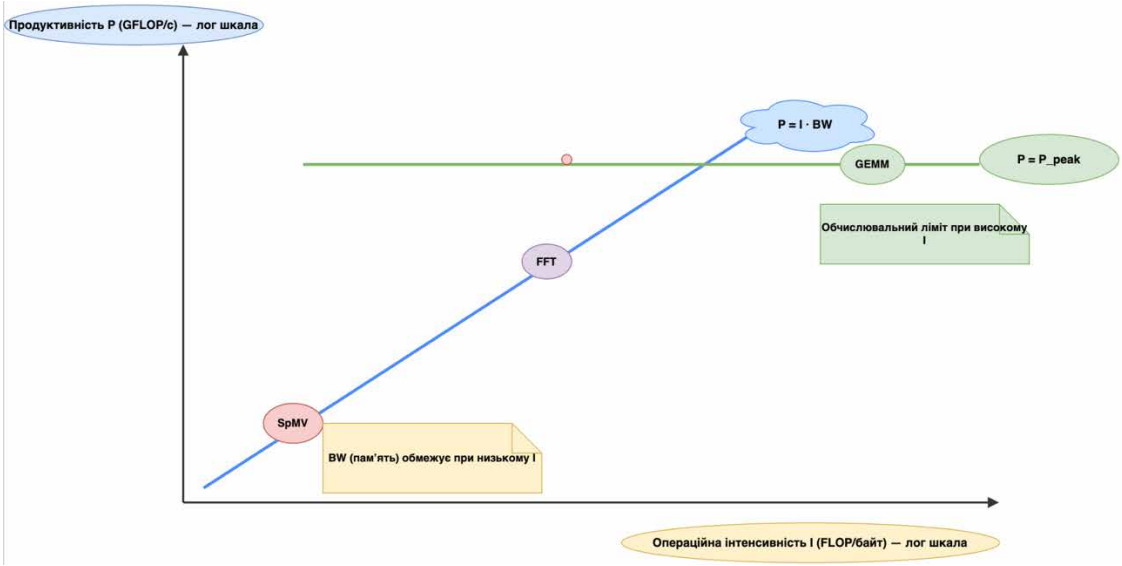

Рис. 1.3 – Roofline-модель оцінювання граничної продуктивності HPC-системи

Застосування Roofline-моделі дає змогу виявити, до якого типу належать конкретні алгоритми - memory-bound(обмежені пам’яттю, як SpMV) чи **c**ompute-bound (обмежені обчисленнями, як GEMM, FFT). Це дозволяє адаптувати методи оптимізації під конкретну архітектуру, зокрема за рахунок підвищення операційної інтенсивності, кешування даних або зменшення накладних витрат синхронізації.

Для аналізу часових характеристик системи застосовується модель передачі повідомлень, яка формалізує затримки обчислень і комунікацій (рис.

1.4). Вона враховує локальний час обробки w_i, час очікування $h_i \cdot g$ та латентність L, що формують сумарний час виконання:

$$T_{total} = i\sum(w_i + h_i \cdot g + L)$$

Де α - стала ініціалізації, $\beta \cdot m$ - лінійна складова передавання повідомлення розміром m.

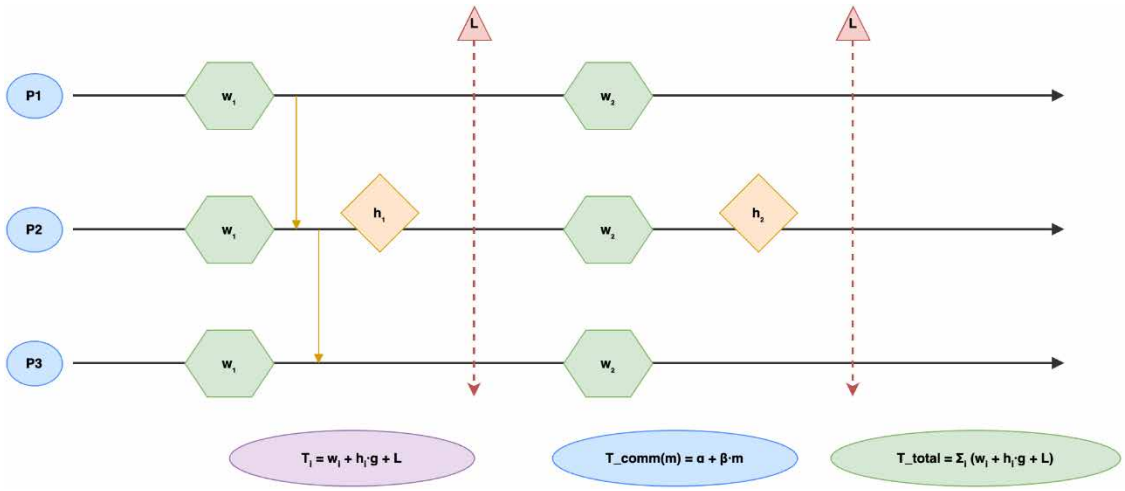

Рис. 1.4 – Модель часових затримок у системі паралельної взаємодії процесів

Методологічною основою проведеного дослідження є поєднання аналітичного моделювання (для оцінки теоретичних обмежень), експериментального вимірювання продуктивності (для валідації моделі) та порівняльного аналізу сучасних HPC-платформ. У табл. 1.2 наведено коротке узагальнення сучасних підходів і бібліотек, що реалізують принципи паралельної обробки.

Таблиця 1.2 – Сучасні підходи та інструменти паралельних обчислень

Система / бібліотека	Мова реалізації	Парадигма	Особливості використання
MPI (OpenMPI, MPICH)	C/C++/Fortran	Міжпроцесна комунікація	Класичний стандарт для кластерних систем; ручне керування передачею повідомлень
OpenMP	C/C++/Fortran	Потоковий паралелізм	Спільна пам’ять; зручна для CPU-архітектур
CUDA / OpenCL	C/C++	Даних / GPU	Масштабовані обчислення на графічних процесорах

Продовження таблиці 1.2

Dask / Ray	Python	Завдань / потоків	Динамічний розподіл обчислень у середовищах Big Data
oneAPI	C++ / Python	Гібридна	Єдина модель програмування CPU–GPU–FPGA

Сучасна наукова парадигма HPC-досліджень базується на інтеграції математичних моделей продуктивності, алгоритмічної оптимізації та архітектурної адаптації, що уможливлює створення ефективних паралельних рішень для аналітичних і моделювальних задач. Отримані теоретичні засади становлять методологічну основу для подальшої реалізації системи паралельної обробки даних у рамках даної роботи.

1.3 Аналіз існуючих рішень

У сучасних дослідженнях паралельної обробки даних розроблено низку ефективних інструментів і фреймворків, які реалізують різні підходи до управління потоками задач, обчислювальними графами та апаратним прискоренням. Аналіз цих рішень дозволяє визначити сильні сторони наявних архітектур і виявити напрямки оптимізації для розроблюваної системи.

Одним із найпоширеніших рішень є Dask - бібліотека Python для розподілених обчислень, яка дозволяє будувати динамічні графи задач і виконувати їх у середовищі багатопроцесорних або кластерних систем. Модель роботи Dask зображена на рис. 1.5: вона складається з етапів створення колекцій (Collections), побудови графів задач (Task Graph) та виконання їх за допомогою планувальників (Schedulers). Dask підтримує різні типи структур даних - Array, DataFrame, Bag - і може масштабуватися від одного вузла до розподіленого кластера.

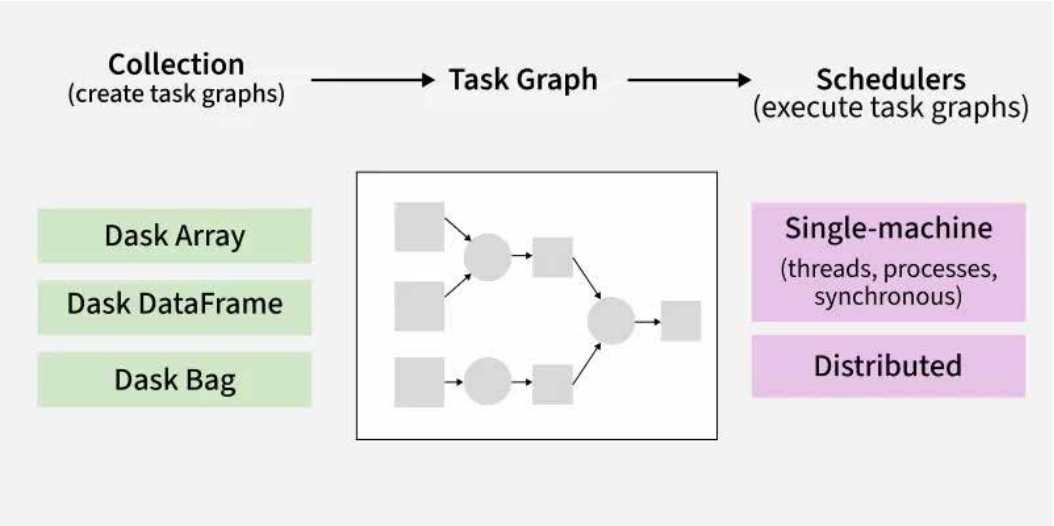

Рис. 1.5 – Архітектурна модель системи Dask для побудови та виконання графів задач

Іншим популярним інструментом є Ray, який реалізує аналогічну парадигму task graph, але з акцентом на автоматичному розподілі задач між вузлами з урахуванням залежностей (рис. 1.6). Граф задач описує взаємозв'язок операцій, де вузли відповідають окремим обчисленням, а ребра - передачі даних. Такий підхід забезпечує гнучке масштабування та можливість асинхронного виконання.

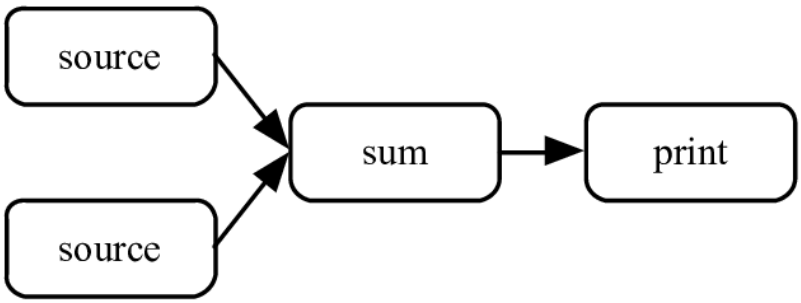

Рис. 1.6 – Приклад графа задач у середовищі паралельних обчислень (Ray/Dask task graph)

До наукових систем низькорівневого рівня належить RaftLib - бібліотека C++ для потоково-орієнтованих обчислень, у якій задачі реалізуються як вузли графа із чергами повідомлень між ними. Приклад загальної схеми роботи RaftLib наведено на рис. 1.7, де показано взаємодію компонентів під час передачі даних у конвеєрному режимі. Такий підхід дозволяє ефективно реалізовувати

конвеєрні обчислення, що особливо корисно для задач аналізу потокових даних та фільтрації сигналів.

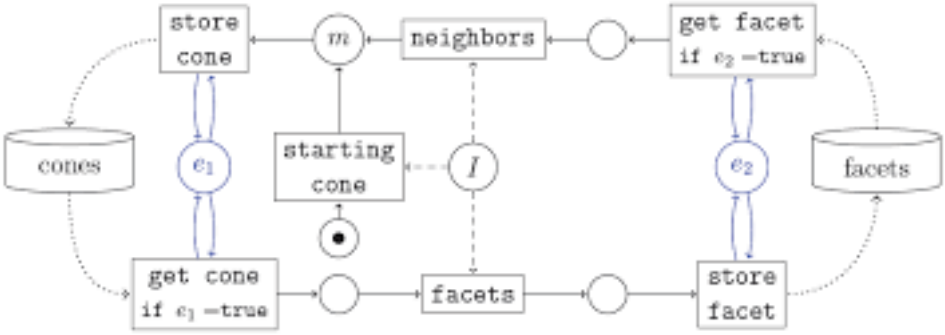

Рис. 1.7 – Схема конвеєрної обробки даних у RaftLib

Ще одним прикладом промислової реалізації є Intel oneAPI - відкрита платформа, що уніфікує паралельне програмування для CPU, GPU та FPGA через спільний стек програмних рівнів (рис. 1.8). Вона включає рівні Runtime, Memory Mapped Device Layer (MMD), драйвери та апаратну логіку (Application Kernel). Архітектура oneAPI забезпечує розподіл навантаження між різними типами пристроїв і дає можливість створювати універсальні високопродуктивні рішення незалежно від обчислювальної архітектури.

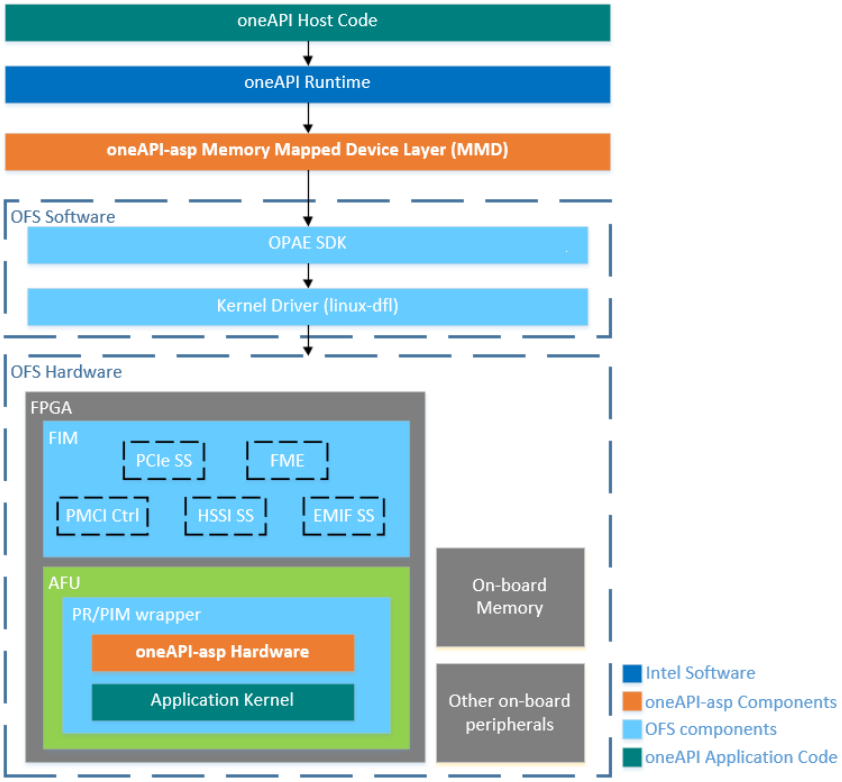

Рис. 1.8 – Структура апаратно-програмного середовища Intel oneAPI

Порівняльний аналіз поданих систем зведено у табл. 1.3, де представлено ключові параметри кожного рішення порівняно з розроблюваною системою, що реалізується на Java із застосуванням Swing GUI, SQLite та бібліотек паралельної обробки даних (ExecutorService, ForkJoinPool).

Таблиця 1.3 – Порівняльна характеристика існуючих рішень і розроблюваної системи

Система	Мова реалізації	Тип паралелізму	Підтримка GUI	Архітектура / масштабування	Особливості
Dask	Python	Завдань, даних	Ні	Кластерна / розподілена	Динамічні графи задач, автоматичне балансування
Ray	Python	Завдань	Частково (через API)	Розподілена	Автоматичне управління залежностями, fault-tolerance
RaftLib	C++	Потокова, конвеєрна	Ні	Локальна / багатопоточна	Черги повідомлень, висока швидкодія
oneAPI	C++ / SYCL	Гібридна (CPU–GPU–FPGA)	Ні	Апаратна інтеграція	Єдина модель програмування для гетерогенних систем
Розроблювана система (Java + Swing + SQLite)	Java	Потокова / завдань	Так	Локальна / багатопоточна	Графічний інтерфейс, керування потоками через ExecutorService, збереження результатів у SQLite

Проведений аналіз демонструє, що більшість існуючих HPC-рішень орієнтовані на високорівневе програмування (Dask, Ray) або апаратну інтеграцію

(oneAPI), проте не забезпечують зручного графічного інтерфейсу для інтерактивного керування процесами обчислень. Розроблювана система на Java зі Swing-інтерфейсом поєднує багатопоточну модель із локальним зберіганням результатів, що робить її придатною для навчальних, дослідницьких і інженерних задач, де необхідна наочна візуалізація стану обчислювальних потоків і контроль над ресурсами.

1.4 Моделювання предметної області

Моделювання предметної області дозволяє глибше зрозуміти, як саме працює система високопродуктивної обробки даних (HPC-система) - від моменту надсилання завдання користувачем до отримання результату. Побудовані UML-моделі допомагають не просто формально описати процеси, а побачити логіку їхньої взаємодії, оцінити навантаження на різні компоненти та перевірити узгодженість архітектури ще до етапу реалізації [10].

На діаграмі варіантів використання (рис. 1.9) показано основних учасників системи: користувач додатку, системний адміністратор, розробник HPC і аналітик даних.

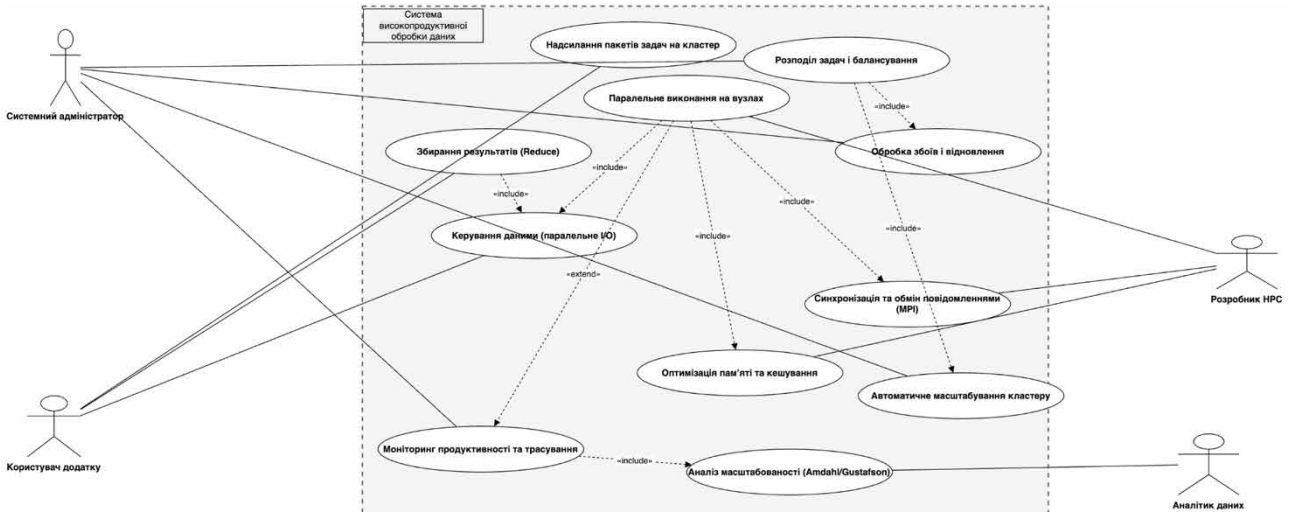

Рис. 1.9 – Діаграма варіантів використання системи високопродуктивної обробки даних

Вони взаємодіють із системою через набір прецедентів - від надсилання пакетів завдань на кластер до збору результатів і моніторингу ефективності.

Адміністратор відповідає за налаштування кластеру, розподіл ресурсів і контроль збоїв; аналітик - за оцінку масштабованості (моделі Амдала та Густафсона); розробник - за підтримку модулів паралельного вводу/виводу (Parallel I/O) та оптимізацію кешування. Таке відображення ролей допомагає зрозуміти, як співпрацюють усі учасники системи в єдиному процесі обчислень.

Подальша послідовнісна діаграма (рис. 1.10) описує динаміку виконання типового завдання.

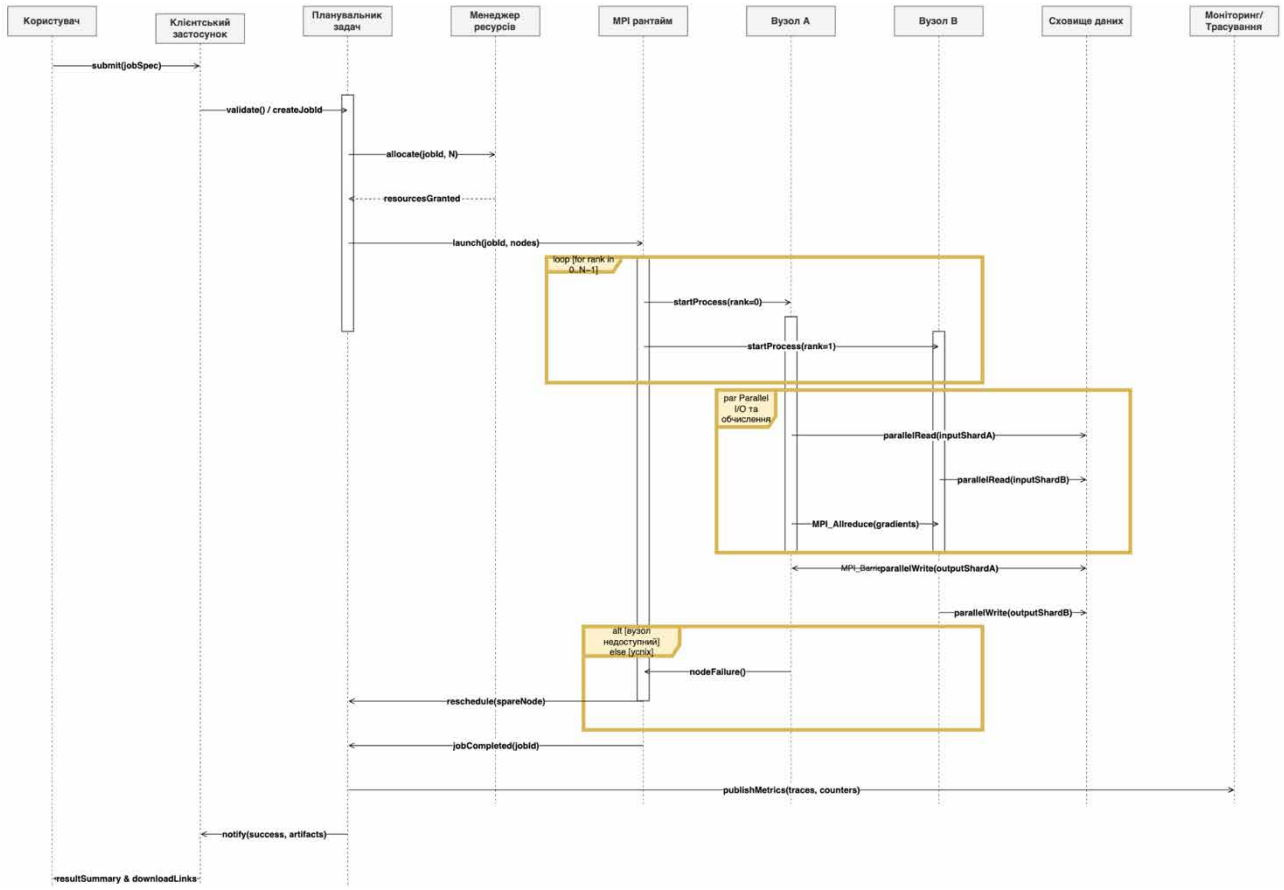

Рис. 1.10 – Послідовність виконання паралельного завдання в HPC-системі

Користувач надсилає специфікацію завдання, планувальник перевіряє її, формує унікальний ідентифікатор та звертається до менеджера ресурсів. Після виділення потрібної кількості вузлів запускається паралельна секція (Parallel I/O та MPI-Allreduce), де вузли виконують частини обчислення незалежно, синхронізуючись через повідомлення. Якщо один із вузлів виходить з ладу, система автоматично перепланує його роботу (reschedule) і після завершення

усіх процесів повідомляє користувача про результат. Така модель наочно демонструє життєвий цикл задачі - від запуску до звітування про результати.

Діаграма діяльності (рис. 1.11) деталізує основні кроки виконання процесу: валідацію параметрів, постановку в чергу, виділення ресурсів, запуск обчислень і подальшу публікацію метрик.

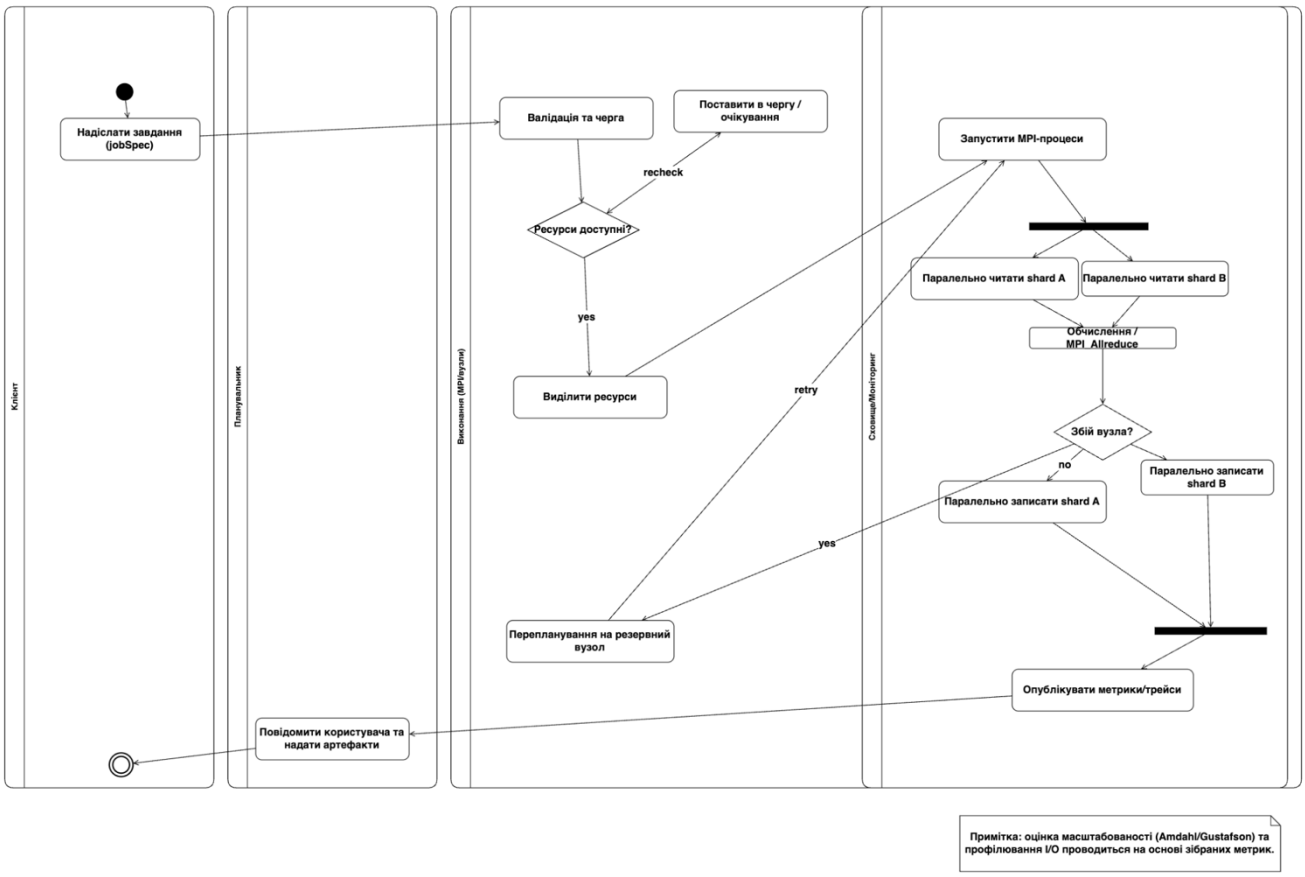

Рис. 1.11 – Діаграма діяльності процесу паралельної обробки завдань

Під час роботи MPI-процесів відбувається паралельне читання вхідних фрагментів даних (shard A, B), обчислення колективних операцій (Allreduce), запис результатів і, за потреби, повторне виконання на резервному вузлі. Після завершення усіх етапів користувач отримує не лише результати, а й аналітичні показники продуктивності системи.

Узагальнюючи, створені UML-моделі допомагають відтворити логіку роботи системи «в розрізі»: хто, коли й із чим взаємодіє, які потоки даних циркулюють між компонентами, та де можуть виникати «вузькі місця». Такий підхід не лише забезпечує структурну цілісність проєкту, а й створює основу для

наступного етапу - архітектурного моделювання та програмної реалізації модулів паралельних обчислень.

1.5 Аналіз вимог до системи

Аналіз вимог є ключовим етапом проектування системи високопродуктивної обробки даних, оскільки саме на цьому етапі визначаються функціональні можливості, технічні параметри та обмеження, які забезпечують досягнення заданих показників продуктивності та масштабованості. Вимоги системи поділено на чотири основні групи: функціональні, нефункціональні, технічні та апаратні, що подані нижче у відповідних таблицях.

У таблиці 1.4 наведено функціональні вимоги, які визначають основні задачі, що реалізує система, - від надсилання завдань і моніторингу процесів до збирання результатів і автоматичного масштабування.

Таблиця 1.4 – Функціональні вимоги до системи

№	Вимога	Опис функціоналу
1	Надсилання обчислювальних завдань	Користувач може створювати і запускати задачі для обробки великих масивів даних на кластері
2	Планування і балансування навантаження	Система автоматично розподіляє ресурси між вузлами залежно від складності задач
3	Синхронізація процесів	Підтримується колективна взаємодія між вузлами за допомогою MPI / Parallel I/O
4	Моніторинг продуктивності	Реалізується збір метрик продуктивності, навантаження на CPU/GPU та часу виконання
5	Збереження і візуалізація результатів	Результати обчислень записуються в базу даних і виводяться у GUI
6	Обробка збоїв	Автоматичне відновлення після помилок через перепланування задач

У таблиці 1.5 наведено нефункціональні вимоги, що описують якісні характеристики системи - швидкодію, надійність, безпеку, зручність користування та масштабованість.

Таблиця 1.5 – Нефункціональні вимоги до системи

№	Вимога	Опис
1	Продуктивність	Час реакції системи не перевищує 200 мс для локальних операцій GUI
2	Масштабованість	Підтримка розширення кількості вузлів до 64 без деградації швидкодії
3	Надійність	Безперервність обробки забезпечується коефіцієнтом готовності не нижче 99,9%
4	Безпека	Використовується TLS для шифрування міжвузлових з’єднань та авторизація через токени
5	Зручність	Графічний інтерфейс (Swing GUI) забезпечує інтуїтивну взаємодію користувача з системою
6	Адаптивність	Система працює на різних ОС (Windows, Linux, macOS) з мінімальними змінами конфігурації

У таблиці 1.6 подано технічні та апаратні вимоги, які визначають програмне середовище, інфраструктуру та параметри обладнання, необхідні для коректної роботи HPC-комплексу.

Таблиця 1.6 – Технічні та апаратні вимоги

№	Параметр	Мінімальні вимоги	Рекомендовані вимоги
1	Процесор	4 ядра, 2.5 ГГц	8 ядер, 3.5 ГГц або вище
2	Оперативна пам’ять	8 ГБ	16–32 ГБ для розподіленого виконання
3	Сховище	SSD ≥ 256 ГБ	SSD ≥ 512 ГБ з IOPS > 100K
4	Мережа	Ethernet ≥ 1 Гбіт/с	Інтерконект InfiniBand ≥ 10 Гбіт/с
5	ПЗ / бібліотеки	Java SE 17, SQLite 3, MPI4J, ExecutorService	+ OpenMP, CUDA SDK (для GPU вузлів)

Проведений аналіз дозволяє узгодити вимоги між усіма учасниками розробки та гарантує, що проєктована система відповідатиме як практичним, так і дослідницьким цілям. Вона має забезпечити стабільну паралельну обробку великих обсягів даних, підтримку візуального керування обчисленнями та ефективну роботу на сучасних апаратних архітектурах. Сукупність визначених вимог формує основу для подальшого етапу - архітектурного проєктування та реалізації модулів програмного забезпечення.

1.6 Постановка завдання

Під час створення системи високопродуктивної обробки даних сформульовано завдання розробити універсальне рішення, яке забезпечує ефективне виконання складних обчислювальних процесів у паралельному середовищі. Основна мета полягає у створенні гнучкої програмної платформи, здатної виконувати масштабовані обчислення, збирати результати та автоматично оцінювати ефективність використання ресурсів. При цьому особливу увагу приділено інтеграції зручного графічного інтерфейсу, який забезпечує користувачу прозорий контроль за процесами обробки та моніторингом продуктивності системи [9].

Система має підтримувати розподілене виконання завдань із можливістю автоматичного балансування навантаження між вузлами. Вона повинна коректно функціонувати як у локальному середовищі, так і в умовах кластерної архітектури, використовуючи багатопоточність і бібліотеки паралельного виконання на Java. Для обміну даними та синхронізації процесів застосовуються механізми багатопроцесної взаємодії (MPI або аналогічні засоби), що забезпечують узгодженість обчислень і мінімізацію затримок у комунікаціях.

Вхідні дані системи включають великі масиви числової, аналітичної або телеметричної інформації, що потребують багатокрокової обробки. Користувач задає конфігураційні параметри - обсяг даних, кількість потоків, тип алгоритму (наприклад, швидке перетворення Фур’є, матричні обчислення, MapReduce). На основі цих параметрів система формує граф виконання задачі, визначає послідовність обчислень і розподіляє їх між процесами.

Вихідними даними є результати паралельних обчислень у вигляді числових таблиць, статистичних звітів, графічних побудов масштабованості та зібраних метрик продуктивності. Додатково формується база результатів у SQLite, що містить історію виконання задач, показники часу, завантаження ресурсів і ефективності. Ці дані можуть використовуватися для подальшої аналітики або повторного запуску з оптимізованими параметрами.

Розроблена система повинна:

– забезпечувати стабільне паралельне виконання великомасштабних обчислень із контролем навантаження;

– мати візуальний інтерфейс для конфігурації завдань, спостереження за їхнім виконанням та перегляду результатів;

– підтримувати зберігання результатів, логів і метрик продуктивності для подальшого аналізу;

– дозволяти оцінювати ефективність виконання за допомогою моделей масштабованості (Амдала, Густафсона);

– бути придатною до розширення - як у частині обчислювальних алгоритмів, так і у кількості вузлів.

Поставлене завдання передбачає створення інтегрованого середовища для паралельної обробки даних, яке поєднує продуктивність HPC-систем із зручністю інтерфейсу користувача. Отримане рішення має стати основою для подальших досліджень у сфері оптимізації масштабованих обчислень і аналітики ефективності систем розподіленої обробки.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних HPC-системи

Логічна модель даних є основою побудови схеми бази даних, що забезпечує узгодженість, цілісність і масштабованість інформаційних потоків у системі високопродуктивних обчислень. Вона формує структуроване уявлення про сутності, їх атрибути та взаємозв'язки, що виникають під час виконання обчислювальних задач, моніторингу продуктивності та зберігання метрик. Відповідно до вимог проєкту, модель побудовано з урахуванням принципів нормалізації до третьої нормальної форми (3NF), що дозволяє уникнути надмірності, забезпечити однозначність і підвищити ефективність запитів до бази даних.

На рис. 2.1 наведено логічну модель даних системи високопродуктивної обробки інформації, побудовану на основі UML-ER-діаграми, що відображає основні сутності -User, Job, Task, Node, Metric- та зв'язки між ними.

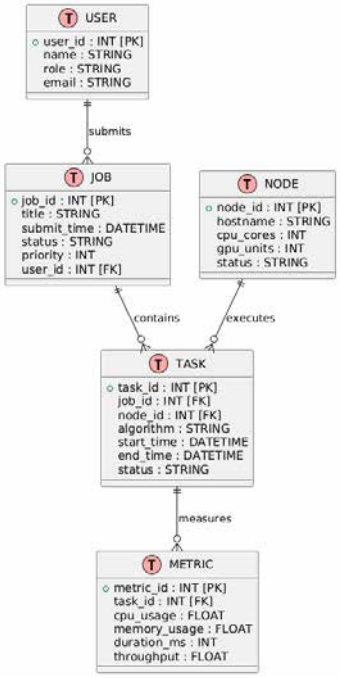

Рис. 2.1 – Логічна модель даних системи високопродуктивних обчислень (HPC)

Як видно з рисунку, побудована модель орієнтована на багаторівневу ієрархію: користувач (User) ініціює завдання (Job), яке складається з множини паралельних підзадач (Task), що виконуються на вузлах кластеру (Node), а результати кожної з них фіксуються у вигляді показників (Metric). Така структура забезпечує чітке розмежування ролей і підвищує узгодженість даних у розподіленому середовищі.

Для перевірки структурної оптимальності моделі проведено аналіз таблиць на предмет нормалізації. Результати подано у табл. 2.1, де відображено ступінь нормалізації кожної сутності та основні ключові залежності.

Таблиця 2.1 – Рівні нормалізації сутностей бази даних HPC-системи

Сутність	Первинний ключ	Нормальна форма	Основна функціональна залежність
User	user_id	3NF	user_id → {name, role, email}
Job	job_id	3NF	job_id → {title, submit_time, status, priority, user_id}
Task	task_id	3NF	task_id → {job_id, node_id, algorithm, status, часові поля}
Node	node_id	3NF	node_id → {hostname, cpu_cores, gpu_units, status}
Metric	metric_id	3NF	metric_id → {task_id, cpu_usage, memory_usage, duration_ms, throughput}

Побудована логічна модель забезпечує повну ізоляцію між рівнями даних, що дозволяє зменшити ризик аномалій вставки, оновлення та видалення. Крім того, використання зовнішніх ключів (*FK*) гарантує підтримку референційної цілісності, що критично важливо для систем з високою інтенсивністю транзакцій, характерних для HPC-платформ.

Застосування реляційної схеми з чіткою структурою сутностей створює основу для подальшої реалізації оптимізованих запитів SQL, побудови індексів та реалізації OLAP-модулів аналітики. Такий підхід забезпечує не лише ефективність обробки, а й масштабованість системи при збільшенні кількості вузлів і паралельних процесів. Таким чином, логічна модель даних HPC-системи

є ключовим елементом архітектури, що гарантує структурну узгодженість, стабільність та аналітичну цінність усіх зібраних даних.

2.2 Діаграма класів і кооперації

Діаграма класів і кооперацій відображає архітектурну структуру системи високопродуктивних обчислень (HPC) та динаміку взаємодії між її компонентами. На етапі проєктування вона виконує ключову функцію - формалізує логіку модулів, способи обміну даними та механізми управління потоками завдань. Створення моделі класів ґрунтується на принципах об'єктно-орієнтованого аналізу (ООА) та нормалізації програмних сутностей, що забезпечує високу модульність, повторне використання коду та гнучке масштабування системи при зміні конфігурації кластера.

На рис. 2.2 зображено узагальнену діаграму класів HPC-системи.

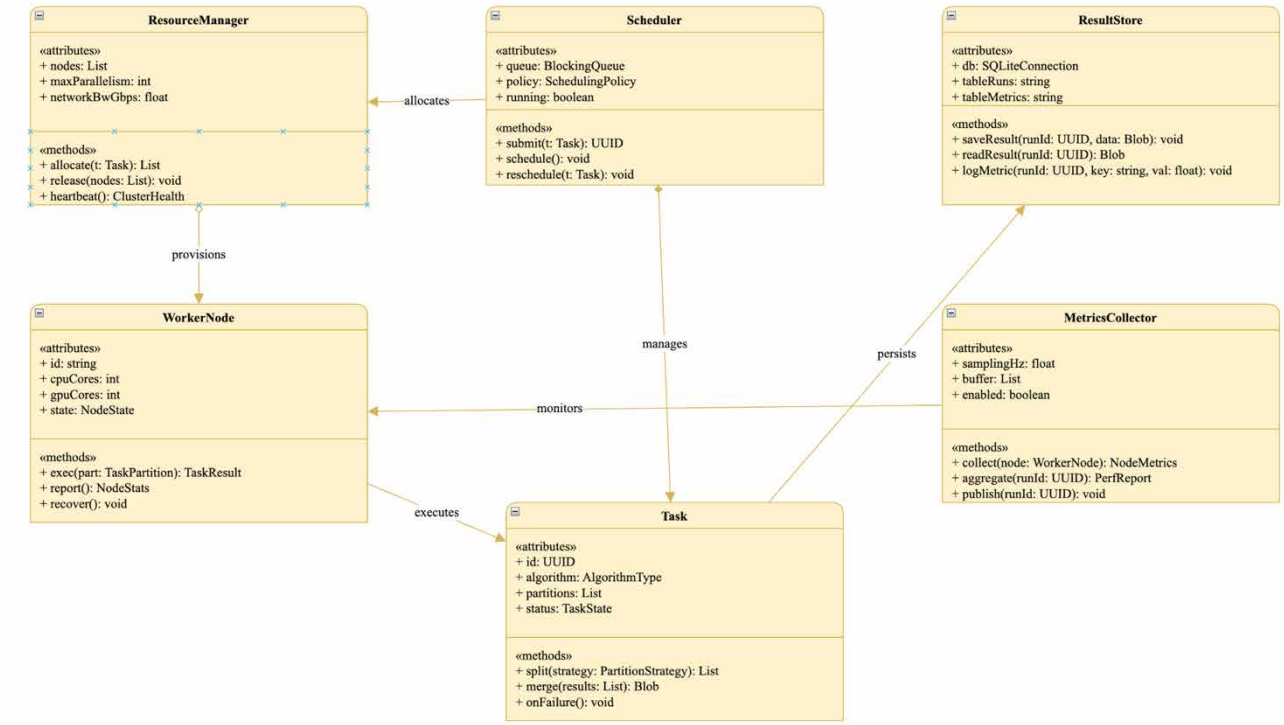

Рис. 2.2 – Діаграма класів системи високопродуктивних обчислень

В основу архітектури покладено шість ключових класів: Scheduler, ResourceManager, WorkerNode, Task, MetricsCollector і ResultStore. Вони формують централізовану схему керування обчислювальним процесом,

де *Scheduler* визначає порядок виконання завдань, ResourceManager відповідає за виділення ресурсів і контроль стану вузлів, WorkerNode реалізує обчислення, а MetricsCollector і ResultStor забезпечують моніторинг і збереження результатів. Подібна модель дозволяє мінімізувати дублювання функцій і досягнути високої зв'язності при низькому рівні залежностей між компонентами, що є ознакою правильної об'єктної нормалізації.

Далі, на рис. 2.3 представлено кооперaційну діаграму, яка відображає сценарій взаємодії користувача із системою через інтерфейс UserUI.

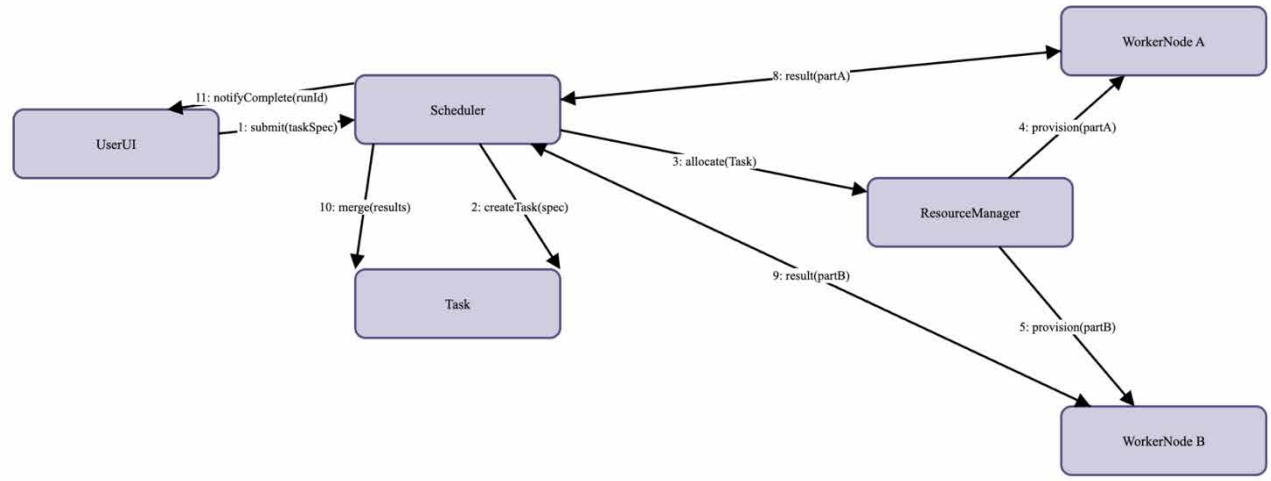

Рис. 2.3 – Діаграма кооперації подання завдання користувачем

На етапі подання завдання користувач передає специфікацію у Scheduler, який створює об'єкт *Task*, ініціює його розбиття на підзадачі та передає їх у ResourceManager для розподілу між вузлами. Така кооперація демонструє послідовну передачу контролю від верхнього рівня до низькорівневих обчислювальних ресурсів, що дозволяє забезпечити стабільне завантаження та адаптивне планування.

Наступна кооперація (рис. 2.4) описує процес повторного планування після відмови вузла.

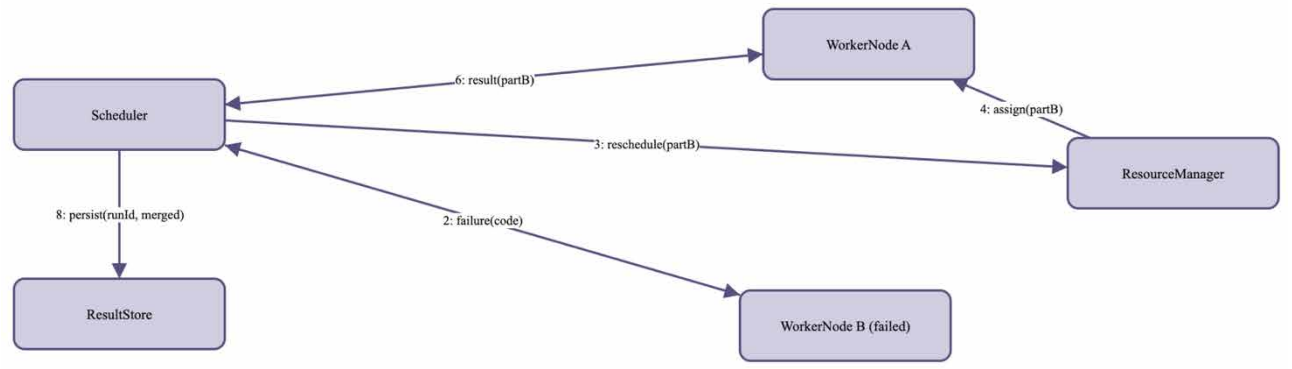

Рис. 2.4 – Діаграма кооперації повторного планування після збою вузла

У випадку збою WorkerNode система активує механізм reschedule(), який викликає повторне призначення завдання через ResourceManager. Результати обчислень передаються у *ResultStore*, де здійснюється перевірка цілісності даних. Такий підхід базується на принципах відновлюваних транзакцій та відмовостійкої синхронізації, що забезпечує стабільну роботу HPC-систем навіть у середовищах із частими збоями або перевантаженнями.

Завершальна кооперація (рис. 2.5) демонструє процес збору метрик після виконання задачі.

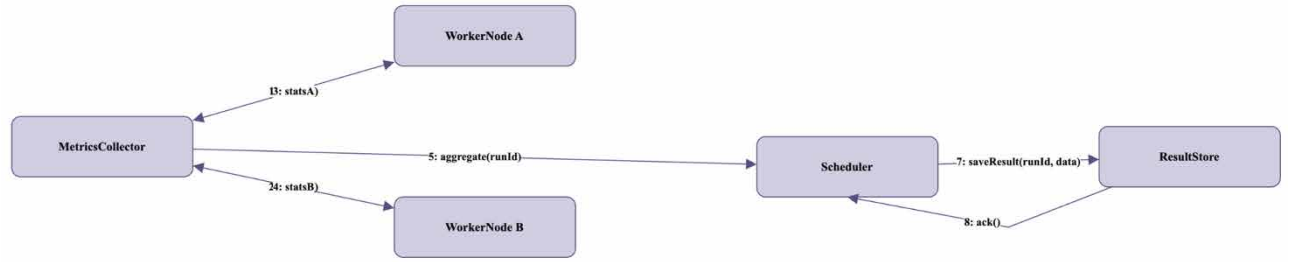

Рис. 2.5 – Діаграма кооперації збору метрик і збереження результатів

Після успішного завершення підзадач вузли надсилають статистику до MetricsCollector, який агрегує дані та публікує їх у ResultStore разом із аналітичними звітами. Всі операції реалізовано з урахуванням принципів слабкої зв’язаності та чіткої відповідальності класів (Single Responsibility Principle), що сприяє підтримуваності коду та можливості незалежного оновлення окремих модулів.

Зведену характеристику класів подано у табл. 2.2, де наведено ключові атрибути, методи та ролі у системі.

Таблиця 2.2 – Ключові класи та їх функції в HPC-системі

Клас	Основні атрибути	Ключові методи	Призначення
Scheduler	queue, policy, running	submit(), schedule(), reschedule()	Планування та координація завдань
ResourceManager	nodes, maxParallelism, networkBwGbps	allocate(), release(), heartbeat()	Розподіл і моніторинг ресурсів
WorkerNode	id, cpuCores, gpuCores, state	exec(), report(), recover()	Виконання підзадач і звітність
Task	id, algorithm, partitions, status	split(), merge(), onFailure()	Модель обчислювальної задачі
MetricsCollector	samplingHz, buffer, enabled	collect(), aggregate(), publish()	Збір і агрегування показників
ResultStore	db, tableRuns, tableMetrics	saveResult(), logMetric(), readResult()	Збереження результатів і метрик

Створена система класів забезпечує високу узгодженість між рівнями архітектури, а моделі кооперації відображають стійкість логіки керування потоками даних. Формалізація таких зв’язків дозволяє мінімізувати ризики архітектурних аномалій, забезпечує масштабованість при зростанні навантажень і створює основу для подальшої реалізації програмних модулів системи

2.3 Діаграма компонентів системи високопродуктивної обробки даних

Діаграма компонентів відображає структурну організацію програмних модулів системи високопродуктивних обчислень і взаємозв’язки між ними через чітко визначені інтерфейси. Така форма моделювання забезпечує узгодженість архітектури, спрощує інтеграцію підсистем та підвищує керованість життєвого циклу програмного забезпечення. Побудова діаграми базується на принципах інкапсуляції функцій, низької зв’язаності та високої когезії компонентів, що відповідає вимогам архітектурних стандартів для HPC-рішень.

На рис. 2.6 наведено діаграму компонентів розробленої HPC-системи, яка демонструє взаємодію основних підсистем через інтерфейси функціонального рівня.

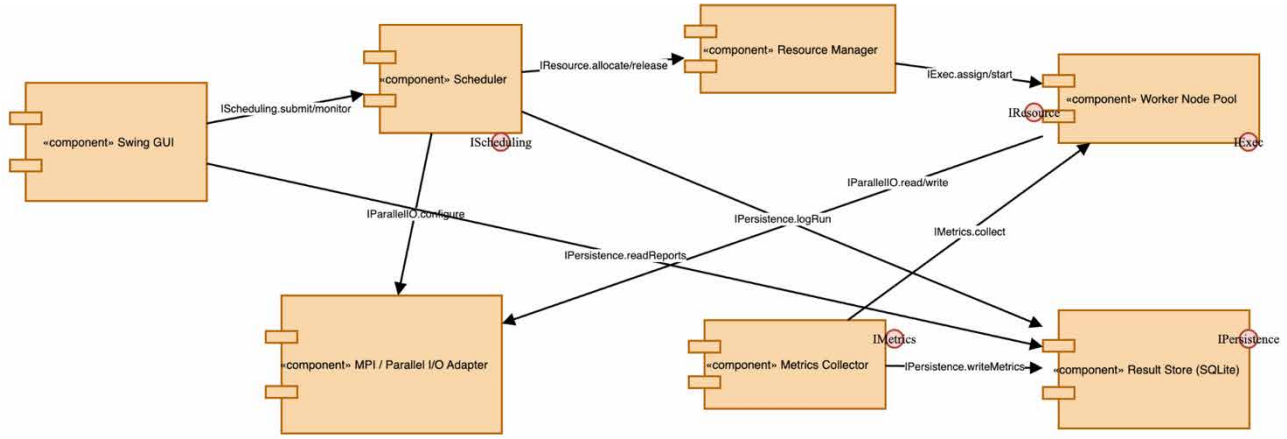

Рис. 2.6 – Діаграма компонентів системи високопродуктивних обчислень

Архітектура реалізована у вигляді семи компонентів: Swing GUI, Scheduler, Resource Manager, Worker Node Pool, MPI/Parallel I/O Adapter, Metrics Collector та Result Store (SQLite). Кожен з них реалізує власний набір інтерфейсів (IScheduling, IResource, IExec, IParallelIO, IMetrics, IPersistence), що формують зрозумілий контракт взаємодії між модулями. Компонент Swing GUI виступає точкою входу для користувача та забезпечує відправлення завдань у планувальник (Scheduler) за допомогою інтерфейсу IScheduling.submit() і моніторингу стану обчислень. Schedulerкоординує обмін повідомленнями між обчислювальними вузлами та менеджером ресурсів, викликаючи операції allocate/release через інтерфейс IResource і формуючи запити на обчислення до Worker Node Pool.

Важливою складовою архітектури є MPI/Parallel I/O Adapter, який забезпечує обмін даними між процесами через стандартизовані механізми паралельного вводу-виводу. Цей компонент підвищує ефективність обробки потоків даних і забезпечує сумісність системи з поширеними HPC-фреймворками (MPI, OpenMP, CUDA). Компонент Metrics Collector реалізує функції збору та агрегування статистичних показників продуктивності з усіх вузлів, використовуючи інтерфейс *IMetrics.collect*, тоді як Result Store

(SQLite) зберігає підсумкові результати та аналітичні звіти, що надходять через IPersistence.writeMetrics і IPersistence.logRun.

Загальну характеристику архітектурних модулів системи наведено у табл. 2.3, де вказано основні ролі та точки інтеграції між компонентами.

Таблиця 2.3 – Основні компоненти HPC-системи та їх взаємодія

Компонент	Основні інтерфейси	Призначення
Swing GUI	IScheduling.submit/monitor	Керування задачами користувача, моніторинг стану
Scheduler	IResource.allocate, IParallelIO.configure	Планування обчислень, диспетчеризація задач
Resource Manager	IExec.assign/start	Керування ресурсами вузлів і контроль стану кластеру
Worker Node Pool	IExec, IResource	Виконання підзадач, передача результатів
MPI / Parallel I/O Adapter	IParallelIO.read/write	Паралельний обмін даними між процесами
Metrics Collector	IMetrics.collect, IPersistence.logRun	Збір і аналіз показників ефективності
Result Store (SQLite)	IPersistence.writeMetrics/readReports	Збереження результатів і формування звітів

Побудована компонентна модель забезпечує масштабованість і розширюваність програмного комплексу: кожен модуль може бути модернізований або замінений без порушення цілісності системи. Завдяки чітко визначеним інтерфейсам система підтримує як горизонтальне масштабування (додавання вузлів), так і вертикальне (оптимізація обчислювальних ресурсів). Діаграма компонентів у поєднанні з попередніми UML-моделями утворює логічну основу для реалізації модульної архітектури HPC-платформи, що забезпечує адаптивне балансування навантаження, надійність і контроль продуктивності.

2.4 Діаграма пакетів HPC-системи

Діаграма пакетів є важливим елементом архітектурного моделювання, оскільки демонструє логічне групування модулів системи високопродуктивних обчислень за функціональними ознаками. Вона відображає структуру внутрішньої організації проєкту, розділяючи компоненти за рівнями абстракції - від інтерфейсної взаємодії користувача до низькорівневих модулів введення/виведення та збереження даних. Такий підхід сприяє підвищенню узгодженості архітектури, зменшенню залежностей між частинами системи та спрощує її розширення у майбутньому.

На рис. 2.7 подано діаграму пакетів системи високопродуктивної обробки даних.

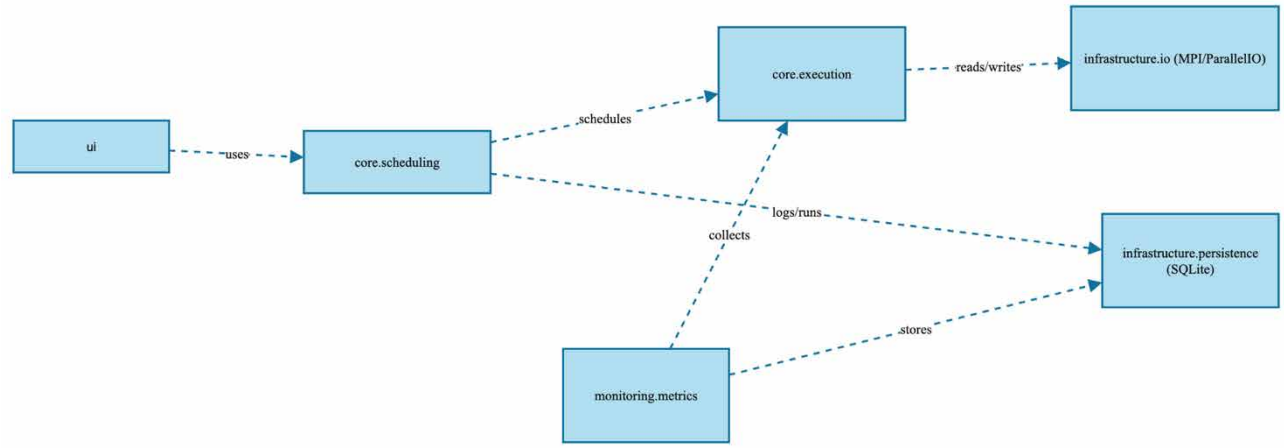

Рис. 2.7 – Діаграма пакетів HPC-системи

У структурі системи виділено кілька ключових пакетів:ui, core.scheduling, core.execution, monitoring.metrics, infrastructure.io та infrastructure.persistence. Логічна ієрархія пакетів побудована за принципом вертикального поділу рівнів - від взаємодії користувача до операційних і системних сервісів. Взаємозв’язки між пакетами реалізовані через асоціації типу *uses*, що забезпечує керований потік даних та контроль залежностей. Наприклад, підсистема *ui* звертається до ядра планувальника (*core.scheduling*), яке, у свою чергу, взаємодіє з модулями виконання (*core.execution*) та збереження результатів (*infrastructure.persistence*). Така структурна композиція підтримує інверсію залежностей і відповідає

принципам архітектури “core–infrastructure”, де бізнес-логіка залишається ізольованою від технічних засобів реалізації.

Для оцінки архітектурної збалансованості проведено аналіз структури пакетів за критеріями зв’язності та стабільності. Результати наведено у табл. 2.4**,** де вказано ступінь абстракції та рівень залежностей для кожного пакета.

Таблиця 2.4 – Архітектурна характеристика пакетів системи

Пакет	Рівень абстракції	Основні залежності	Ступінь стабільності
ui	Високий	core.scheduling	Високий (статичний інтерфейс користувача)
core.scheduling	Середній	core.execution, infrastructure.persistence	Середній (змінювані алгоритми планування)
core.execution	Середній	infrastructure.io, monitoring.metrics	Високий (відповідальний за виконання задач)
monitoring.metrics	Середній	infrastructure.persistence	Високий (аналітичний шар системи)
infrastructure.io	Низький	—	Високий (паралельні ввід/вивід процесів)
infrastructure.persistence	Низький	—	Високий (стабільне сховище результатів)

Структурна модель демонструє, що найменш стабільні елементи розміщено у верхньому рівні (ui, core.scheduling), тоді як нижні пакети (infrastructure.persistence, infrastructure.io) мають високу сталість і мінімальні зовнішні залежності. Це відповідає принципу Layered Architecture, який гарантує, що зміни у верхніх рівнях не впливають на цілісність базових модулів.

Таким чином, побудована діаграма пакетів узагальнює логічну структуру системи HPC, забезпечуючи відокремлення обчислювальної логіки, інфраструктури збереження та контролю метрик. Вона визначає межі відповідальності кожного модуля, зменшує складність підтримки коду та

створює архітектурну основу для реалізації масштабованої платформи високопродуктивних обчислень.

2.5 Висновки до другого розділу

У другому розділі виконано архітектурне проєктування та моделювання основних структурних елементів системи високопродуктивних обчислень, що дозволило сформувати цілісне бачення її логічної та програмної організації. На основі принципів об’єктно-орієнтованого аналізу, інкапсуляції та багаторівневого розділення відповідальності побудовано моделі даних, класи, компоненти й пакети, які забезпечують узгодженість внутрішніх взаємодій між підсистемами.

Розроблена логічна модель даних відображає оптимізовану структуру сутностей, нормалізовану до третьої нормальної форми, що забезпечує цілісність та відсутність надмірності в обробці великих обсягів обчислювальної інформації. Діаграма класів визначає об’єктну основу системи, формалізуючи ролі модулів планування, ресурсного управління, виконання, моніторингу та збереження результатів. Побудовані кооперaційні діаграми демонструють динаміку взаємодії об’єктів під час виконання обчислювальних процесів, включаючи сценарії відновлення після збоїв та агрегацію метрик продуктивності.

На рівні архітектури діаграма компонентів відобразила структуру взаємодії між модулями через стандартизовані інтерфейси, що забезпечують сумісність, масштабованість і розширюваність системи. Створена діаграма пакетів узагальнює логічний поділ системи на рівні: користувацький, ядровий (core) і інфраструктурний, що дозволяє ефективно відокремити бізнес-логіку від технічної реалізації.

Результати другого розділу формують архітектурний каркас системи HPC, який забезпечує структурну цілісність, модульність і адаптивність при зростанні навантаження. Отримані UML-моделі є основою для наступного етапу -

реалізації програмного забезпечення та розроблення алгоритмів паралельної обробки даних, спрямованих на підвищення ефективності обчислювального середовища.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій та інструментальних засобів реалізації системи

Розроблення системи високопродуктивної обробки даних потребує ретельно обґрунтованого вибору технологій, які забезпечують одночасно високу ефективність обчислень, масштабованість, стійкість до збоїв та зручність експлуатації. З огляду на особливості предметної області та вимоги, сформульовані у першому розділі, архітектура системи побудована на поєднанні засобів багатопоточності Java, бібліотек паралельних обчислень, інструментів міжпроцесної взаємодії та локального реляційного сховища результатів. Такий підхід забезпечує оптимальний баланс між продуктивністю, переносимістю та простотою модернізації платформи.

Для реалізації серверної логіки обрано платформу Java SE 17, яка гарантує стабільність, високу продуктивність роботи в багатопоточному середовищі та наявність розвиненої стандартної бібліотеки для управління потоками. Основу паралельного виконання становлять механізми ExecutorService і ForkJoinPool, що забезпечують ефективне планування задач різної складності, адаптивне завантаження ядер процесора та підтримку рекурсивних алгоритмів обробки даних. Для моделювання розподіленої взаємодії та колективних операцій застосовано інструментарій MPI4J / MPI4Py, що дозволяє відтворювати поведінку вузлів HPC-кластеру, виконувати паралельне введення/виведення та синхронізувати процеси на рівні задач.

Графічний інтерфейс реалізовано засобами Java Swing, що надають можливість створити інтуїтивний та інтерактивний інструмент для керування обчисленнями, моніторингу продуктивності та перегляду результатів. Для збереження результатів експериментів, виконаних задач і метрик навантаження використано легку реляційну СУБД SQLite3, що оптимально підходить для

автономних HPC-платформ та локальних дослідницьких систем. Аналітичні модулі створено на Python із застосуванням NumPy, Pandas і Matplotlib, які дозволяють генерувати графіки масштабованості, аналізувати часові ряди продуктивності та обчислювати характеристики ефективності за моделями Амдала й Густафсона.

Вибраний набір технологій забезпечує підтримку адаптивного планування, можливість детального моніторингу, інтерактивний контроль процесів обчислень і сумісність з поширеними HPC-практиками. Технології та інструментальні засоби представлені у таблиці 3.1.

Таблиця 3.1 – Технології та інструментальні засоби, використані у системі

№	Технологія / інструмент	Призначення	Обґрунтування вибору
1	Java SE 17	Серверна логіка, керування потоками	Висока стабільність, масштабованість, сучасні механізми багатопоточності
2	ExecutorService, ForkJoinPool	Паралельне виконання задач	Адаптивне балансування, підтримка рекурсивних та розподілених обчислень
3	MPI4J / MPI4Py	Міжпроцесна взаємодія, моделювання HPC-кластеру	Можливість колективних операцій, синхронізація та передача повідомлень
4	Java Swing	Графічний інтерфейс системи	Легка інтеграція з Java-ядром, автономність, підтримка віджету реального часу
5	SQLite3	Зберігання задач, метрик, результатів	Мінімальні накладні витрати, відсутність серверної інфраструктури
6	Python, NumPy, Pandas	Аналітика продуктивності, обробка даних	Зручність математичних обчислень, гнучкість аналізу
7	Matplotlib	Побудова графіків масштабованості та навантаження	Стандартизований інструмент для наукової візуалізації
8	JUnit	Тестування модулів Java	Автоматизація контролю якості та перевірка логіки
9	Git / GitHub	Контроль версій	Підтримка командної роботи та відстеження змін
10	OpenMP / CUDA (опційно)	Підтримка гібридних CPU–GPU обчислень	Можливість розширення системи на апаратні прискорювачі

Визначений технологічний стек формує повноцінну основу для реалізації високопродуктивної системи, яка поєднує класичні механізми паралельних обчислень, інструменти розподіленої взаємодії та зручний графічний інтерфейс. Обрані засоби забезпечують оптимальну продуктивність, портативність та можливість інтеграції з сучасними HPC-фреймворками, що відповідає вимогам розроблення адаптивної та масштабованої платформи.

3.2 Інформаційна база системи

Інформаційна база системи високопродуктивної обробки даних формується як багаторівнева структура, що поєднує оперативні дані виконання задач, історичні записи про конфігурації кластеру, параметри алгоритмів та агреговані OLAP-метрики для подальшого аналітичного опрацювання. Вона відіграє ключову роль у забезпеченні достовірності, відтворюваності та наукової цінності експериментів, оскільки акумулює результати виконання паралельних процесів у різних конфігураціях та інтерпретує їх через систему показників ефективності. Саме структурована інформаційна база дає можливість оцінювати поведінку алгоритмів у реальних умовах навантаження та формувати адаптивні правила оптимізації розподілу задач у кластері, що є складовою наукової новизни роботи.

На рисунку 3.1 подано фізичну модель даних OLAP-сховища системи, яке використовується для акумуляції та багатовимірного аналізу показників продуктивності.

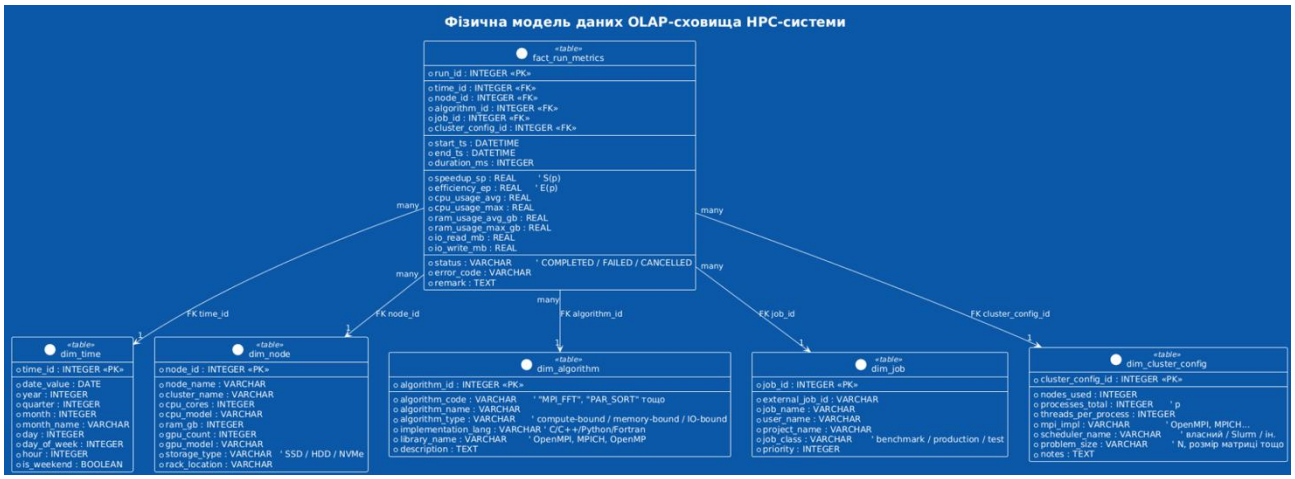

Рис. 3.1 – Фізична модель даних OLAP-сховища HPC-системи

Як видно із схеми (рис. 3.1), центральною сутністю є факт-таблиця *fact_run_metrics*, яка містить результати кожного виконання задачі: часові мітки, ідентифікатори вузлів, обчислювальних конфігурацій і алгоритмів, а також ключові метрики продуктивності - прискорення *S(p)*, ефективність *E(p)*, навантаження CPU/GPU, обсяг операцій вводу-виводу та використання пам’яті. Довідкові таблиці часу, вузлів, алгоритмів і конфігурацій формують основу для аналітичних запитів за принципами зоряної схеми, що дозволяє проводити комплексний OLAP-аналіз поведінки системи залежно від параметрів експерименту. Новизна підходу полягає у поєднанні операційного рівня HPC-процесів з аналітичною моделлю, що дозволяє вперше дослідити ефективність планування та балансування навантаження у вигляді багатовимірних залежностей.

Для дослідження стійкості й продуктивності HPC-модуля застосовано методи статистичної кластеризації, що дозволяють виявляти типові групи задач залежно від ресурсоспоживання та часових характеристик. На рисунку 3.2 наведено графік «ліктя», який використано для вибору оптимальної кількості кластерів.

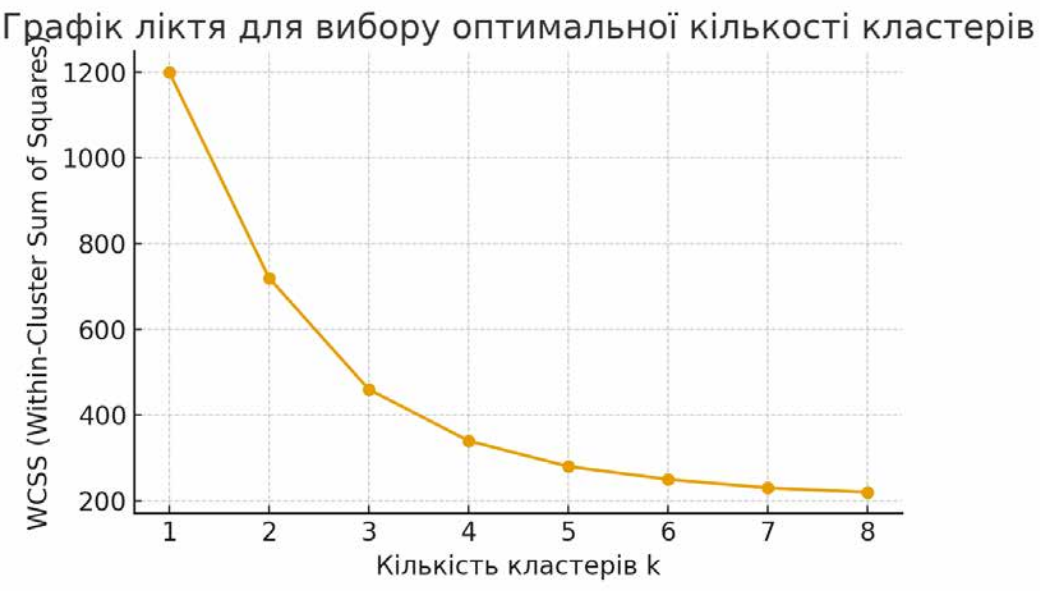

Рис. 3.2 – Графік ліктя для вибору оптимальної кількості кластерів

Графік (рис. 3.2) демонструє точку стабілізації WCSS, що відповідає значенню $k = 3$. Це дає змогу визначити три домінантні групи задач, які суттєво різняться за інтенсивністю обчислень, що було подальшою основою для оптимізації планувальника. На рисунку 3.3 подано результат кластеризації задач у просторі нормованих ознак.

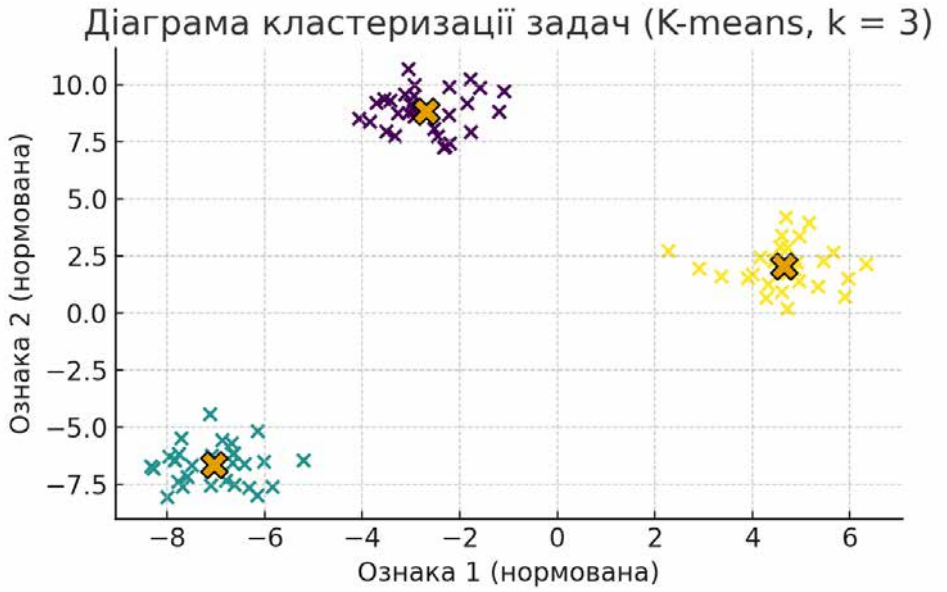

Рис. 3.3 – Діаграма кластеризації задач (K-means, k = 3)

Згідно з результатами (рис. 3.3), задачі групуються навколо трьох центрів: обчислювально інтенсивні (*compute-heavy*), пам’яттєво інтенсивні (*memory-bound*) та змішані. Отримані кластери інтегруються у модель планування,

дозволяючи адаптивно розподіляти задачі відповідно до доступних ресурсів кластеру. Це формує наукову новизну роботи — впровадження механізму когнітивної кластеризації задач у поєднанні з OLAP-аналізом.

На рисунку 3.4 продемонстровано радіальну діаграму ключових показників, які формуються на основі факт-таблиці OLAP-сховища та використовуються для оцінки ефективності HPC-модуля.

Рис. 3.4 – Радіальна діаграма ключових показників HPC-системи

Графічна форма представлення (рис. 3.4) відображає збалансованість системи за шістьма вимірюваннями: прискоренням *S(p)*, ефективністю *E(p)*, навантаженням CPU, використанням RAM, пропускною здатністю та стабільністю часу виконання. Це забезпечує комплексну оцінку стану системи та створює можливість вибору оптимальних параметрів конфігурації для кожного типу задач, що є однією з ключових інновацій роботи.

Таблиця 3.2 – Нові технічні аспекти, реалізовані в інформаційній базі системи

№	Технічний аспект	Сутність інновації	Практична цінність
1	Інтеграція HPC-виконання з OLAP-моделюванням	Поєднання факт-метрик та довідкових вимірів	Можливість багатовимірного аналізу продуктивності

Продовження таблиці 3.2

2	Автоматична кластеризація задач	Використання K-means із динамічним визначенням *k*	Адаптивне балансування навантаження
3	Формування профілів задач	Визначення типів *compute / memory / IO-bound*	Оптимізація алгоритмів розподілу задач
4	Радіальна метрика ефективності	Інтеграція 6 ключових показників у єдину модель	Комплексна оцінка поведінки системи
5	Зоряна OLAP-схема	Нормалізація до 3NF + предметно орієнтовані виміри	Прискорені аналітичні запити та стійкість даних
6	Історизація конфігурацій кластеру	Зберігання параметрів вузлів у часі	Можливість ретроспективного аналізу продуктивності

Створена інформаційна база формує комплексну основу для дослідження HPC-процесів, поєднуючи OLAP-аналітику, кластеризацію задач і повну історизацію продуктивності. Наукова новизна роботи полягає у впровадженні аналітичного ядра для високопродуктивних обчислень, яке дозволяє здійснювати не просто фіксацію метрик, а їх структуроване багатовимірне трактування. Завдяки цьому система переходить від традиційного моніторингу до інтелектуальної оцінки ефективності та адаптивного керування обчислювальними ресурсами. Це забезпечує основу для формування нових моделей планування, що відповідають сучасним вимогам HPC-середовищ і сприяють підвищенню їх продуктивності.

3.3 Архітектура системи та проєктування функціоналу обробки результатів дослідження

Архітектура розробленої високопродуктивної системи побудована на модульному принципі, що забезпечує ізоляцію компонентів, масштабованість та можливість незалежного розширення окремих підсистем. Логічна структура включає три рівні: клієнтський, що відповідає за інтерфейс користувача; серверний, який реалізує планування, диспетчеризацію та контроль виконання задач; та кластерний, що виконує обчислення на множині вузлів з підтримкою

MPI та паралельного введення/виведення. На рисунку 3.5 наведено узагальнену схему компонентної взаємодії між ключовими елементами підсистем.

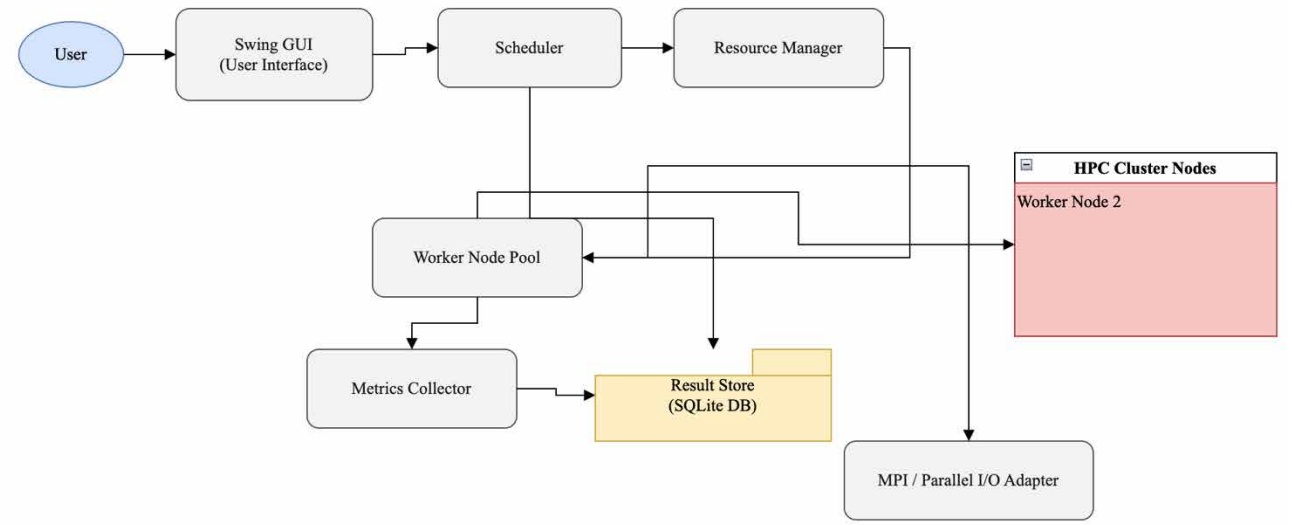

Рис. 3.5 – Компонентна взаємодія модулів HPC-системи

Як показано на рисунку 3.5, користувач взаємодіє з системою через Swing-інтерфейс, який ініціює постановку обчислювальних задач. Планувальник (Scheduler) виконує розподіл задач відповідно до наявних ресурсів, а диспетчер ресурсів (Resource Manager) визначає оптимальне виділення потоків і процесів. Вузли-виконавці (Worker Node Pool) здійснюють паралельне виконання задач із використанням MPI/Parallel I/O адаптера. Після завершення виконання дані про ресурси та результати передаються у модуль збору метрик (Metrics Collector), а далі - у СУБД SQLite, де формуються підсумкові записи для аналітики.

Фізична архітектура розгортання, подана на рисунку 3.6, відображає розподіл програмних модулів між клієнтською станцією, сервером керування HPC-процесами та обчислювальним кластером. Така структуризація забезпечує відокремлення GUI-частини від обчислювальної інфраструктури, рівномірний розподіл навантаження і можливість масштабування кластеру без модифікації інтерфейсного рівня.

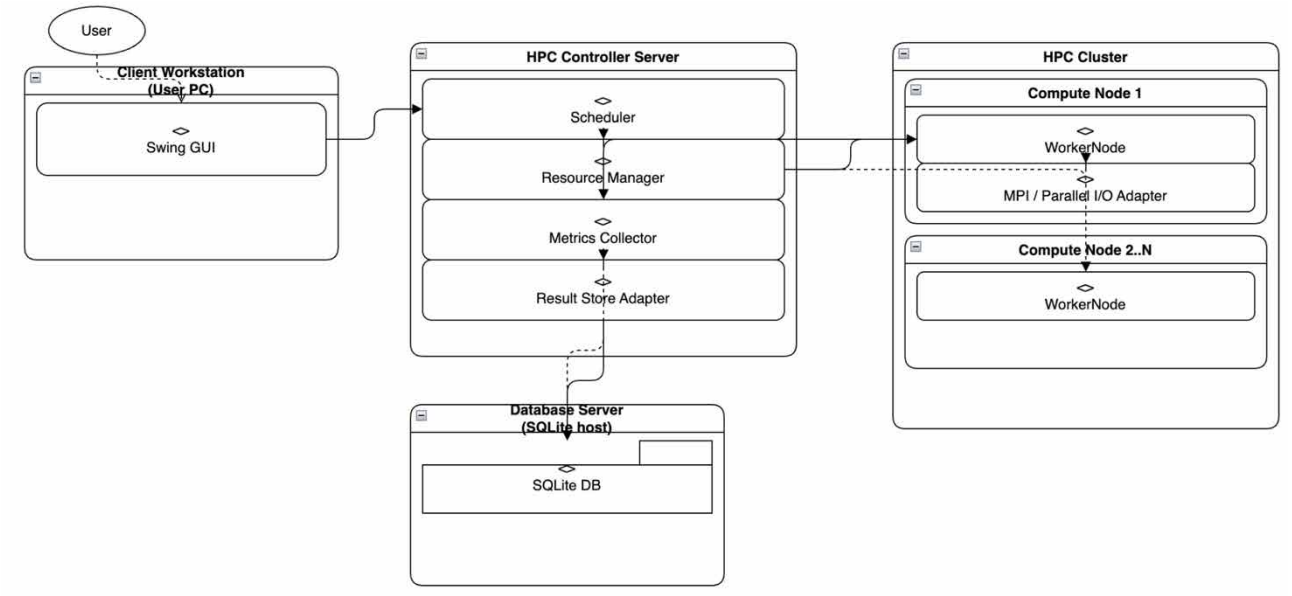

Рис. 3.6 – Діаграма розгортання компонентів HPC-системи

На рисунку 3.6 видно, що Swing-клієнт, виконуючись на локальній робочій станції, здійснює лише генерацію задач та візуалізацію результатів. Основні модулі керування - Scheduler, Resource Manager, Metrics Collector *та* Result Store Adapter - розгорнуті на сервері HPC Controller. Обчислювальні вузли кластеру мають власні модулі WorkerNode та адаптери для MPI-взаємодії. База даних розташована на окремому сервері, що полегшує паралельний доступ і дозволяє виконувати аналітичні запити незалежно від процесів виконання задач.

Розроблений функціонал обробки результатів дослідження передбачає багаторівневу фіксацію характеристик виконання: параметрів конфігурацій вузлів, часових міток, профілів ресурсоспоживання та агрегованих метрик продуктивності. Така організація дає змогу виконувати масштабовані OLAP-запити, будувати графіки прискорення, ефективності, кластеризації задач, формувати комплексні профілі навантаження та здійснювати порівняльний аналіз HPC-алгоритмів. Взаємодія компонентів архітектури забезпечує безперервний цикл: запуск задач → виконання → моніторинг → збір метрик → запис у СУБД → аналітична обробка. Цей цикл створює основу для наукових експериментів, оскільки дозволяє отримувати достовірні, структуровані та порівнювані результати.

Запропонована архітектура забезпечує цілісну інтеграцію клієнтського інтерфейсу, сервера керування та обчислювального кластеру, формуючи гнучку і масштабовану модель виконання високопродуктивних задач. Вона створює основу для автоматизованого планування та оптимізації обчислень, підтримує збір та багатовимірний аналіз результатів, а також забезпечує наукову відтворюваність досліджень. Така організація значно підвищує ефективність проведення експериментів і дозволяє реалізувати повноцінний цикл HPC-обробки в рамках єдиної системи.

3.4 Алгоритмічне забезпечення системи високопродуктивної обробки даних

Алгоритмічне забезпечення побудованої HPC-системи визначає послідовність операцій, за допомогою яких реалізуються ключові функції: постановка та запуск паралельних задач, інтелектуальний аналіз результатів виконання і формування узагальнених візуальних метрик для дослідника. На рисунку 3.7 подано блок-схему алгоритму запуску паралельного завдання, що формалізує взаємодію користувача з інтерфейсом, модулем планувальника та підсистемою керування ресурсами.

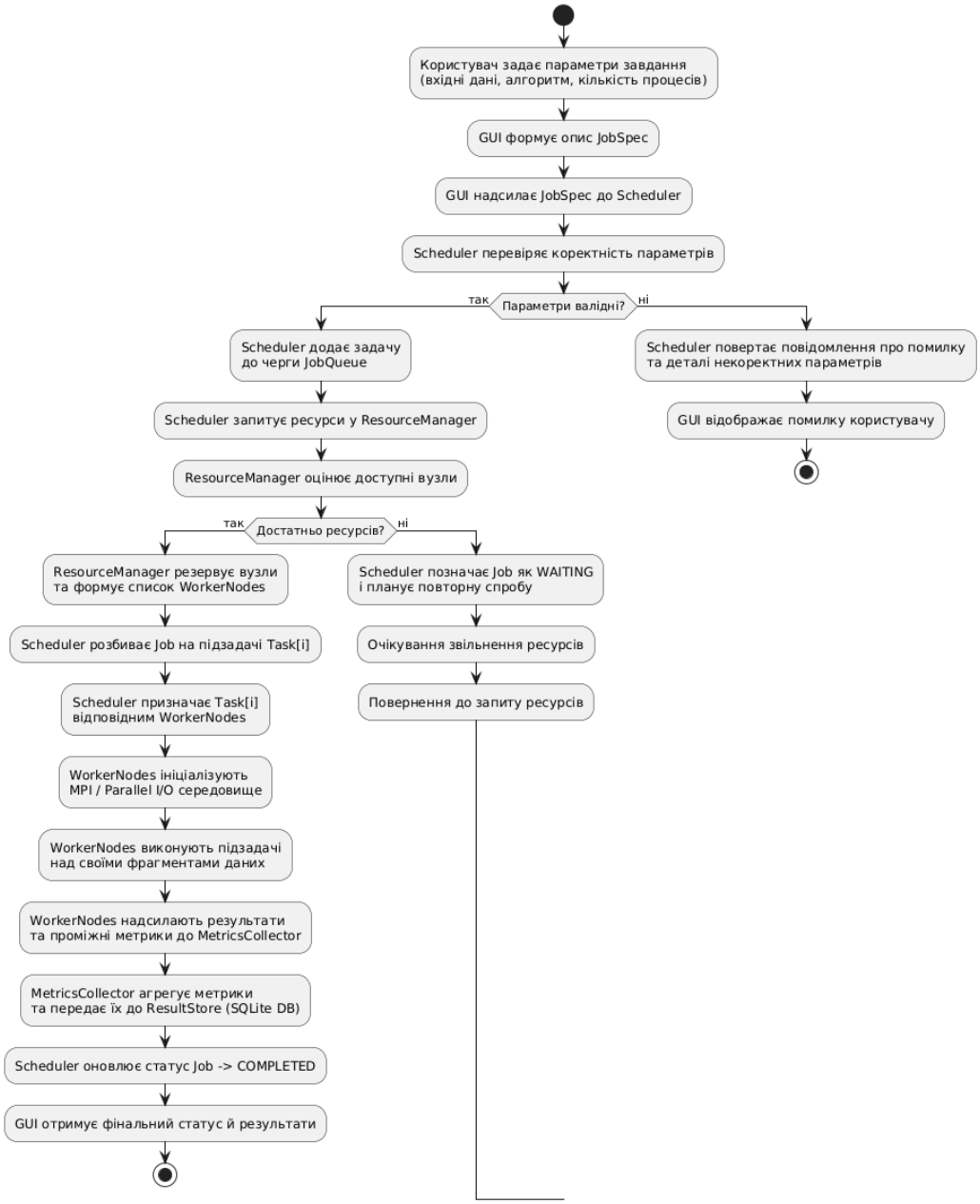

Рис. 3.7 – Алгоритм запуску паралельного завдання в HPC-системі

Згідно з алгоритмом на рис. 3.7, користувач задає параметри експерименту (вибір алгоритму, вхідні дані, кількість процесів), після чого GUI формує опис задачі JobSpec і передає його планувальнику. Scheduler виконує валідацію параметрів, додає задачу до черги та запитує ресурси у Resource Manager. У разі наявності достатньої кількості обчислювальних вузлів формується пул WorkerNode, здійснюється розбиття задачі на підзадачі, ініціалізація MPI/Parallel I/O середовища та паралельне виконання фрагментів даних. Проміжні метрики та результати надсилаються до модуля збору метрик і зберігаються у Result Store,

після чого статус задачі оновлюється до COMPLETED, а користувач отримує підсумкові дані. Якщо ресурси відсутні, задача переводиться у стан WAITING до моменту їх звільнення, що дозволяє забезпечити справедливе та відмовостійке планування.

На рисунку 3.8 наведено алгоритм кластеризації задач методом K-means з подальшим оновленням OLAP-сховища, який реалізує інтелектуальну обробку накопичених у Result Store експериментальних даних.

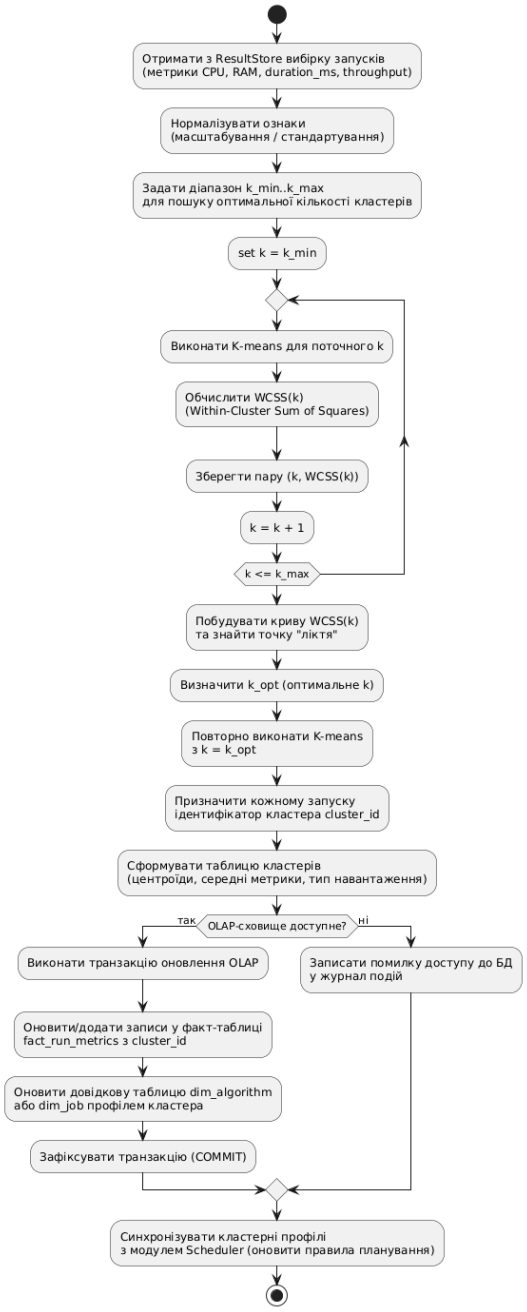

Рис. 3.8 – Алгоритм кластеризації задач (K-means) та оновлення OLAP-сховища

Алгоритм на рис. 3.8 передбачає вибірку історичних запусків з бази даних, нормалізацію ознак (метрики CPU, RAM, тривалість, пропускна здатність), пошук оптимальної кількості кластерів за методом «ліктя», повторне навчання K-means з оптимальним k та присвоєння кожному запуску ідентифікатора кластера. Далі формується агрегована таблиця кластерних профілів, яка транзакційно інтегрується у факт-таблицю fact_run_metrics та відповідні довідкові виміри OLAP-схеми. У випадку успішного оновлення результати кластеризації синхронізуються з планувальником, що дозволяє адаптувати політику розподілу задач з урахуванням виявлених класів навантаження й тим самим безпосередньо пов’язує аналітичний рівень із контуром керування обчисленнями.

Для узагальнення поведінки системи та наочного представлення інтегральних характеристик продуктивності використовується алгоритм формування радіальної діаграми ключових показників, схема якого зображена на рисунку 3.9.

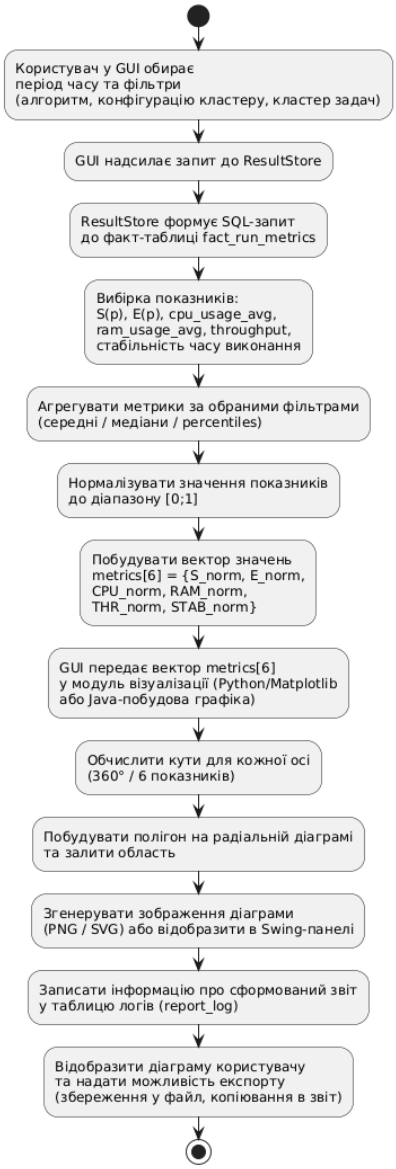

Рис. 3.9 – Алгоритм формування радіальної діаграми ключових показників HPC-системи

Згідно з алгоритмом на рис. 3.9, користувач за допомогою GUI визначає період аналізу та фільтри (алгоритм, конфігурацію кластеру, кластер задач), після чого Result Store виконує SQL-запит до факт-таблиці, агрегує вибрані метрики S(p), E(p), середнє завантаження CPU і RAM, пропускну здатність та стабільність часу виконання і нормалізує їх до єдиного діапазону. На основі нормованого вектора показників модуль візуалізації обчислює кути осей, будує багатокутник на радіальній діаграмі, генерує зображення та зберігає інформацію про сформований звіт у спеціальній таблиці логів. Це дає змогу отримати компактне, але інформативне уявлення про збалансованість HPC-системи,

порівнювати різні конфігурації та сценарії навантаження. Сукупність наведених алгоритмів забезпечує повний цикл обробки результатів досліджень — від постановки паралельного експерименту до його аналітичної інтерпретації й візуалізації, що підвищує наукову цінність отриманих даних і створює основу для подальшої оптимізації архітектури високопродуктивних обчислювальних систем.

3.5 Висновки до третього розділу

У третьому розділі було розроблено та обґрунтовано архітектурно-алгоритмічний базис системи високопродуктивних обчислень, що становить фундамент її функціональної цілісності та наукової новизни. На основі вимог, визначених у попередніх розділах, сформовано раціональний стек технологій, орієнтований на забезпечення масштабованості, відмовостійкості та ефективного паралелізму. Вибрані інструментальні засоби - Java SE 17, багатопотокові засоби ExecutorService/ForkJoinPool, кластерні адаптери MPI, а також аналітичний стек Python/NumPy/Pandas - забезпечують оптимальну інтеграцію між програмною логікою, середовищем виконання та модулем аналітики.

Розроблена інформаційна база системи, структурована за принципами багатовимірної OLAP-моделі, забезпечує можливість комплексного збору, зберігання та подальшої обробки результатів виконання обчислювальних задач. Побудована фізична модель даних дає змогу інтерпретувати продуктивність HPC-модуля на рівні окремих запусків, конфігурацій кластера, алгоритмів та часових характеристик. Використання кластеризації K-means для групування задач та інтеграція її результатів у сховище створює нові можливості для інтелектуального аналізу навантаження та адаптивного планування.

Запропонована архітектура системи реалізує чіткий поділ відповідальностей між клієнтською, серверною та обчислювальною частинами, забезпечує модульність, розширюваність та можливість масштабування.

Розроблені алгоритмічні схеми - запуск паралельного завдання, кластеризація задач та формування радіальної діаграми ключових показників - описують повний життєвий цикл обробки експериментальних даних і демонструють практичну реалізацію теоретичних моделей.

Результати третього розділу формують цілісну архітектурно-алгоритмічну основу системи високопродуктивної обробки даних, де поєднано інструменти паралельних обчислень, моделювання ресурсних характеристик та інтелектуального аналізу продуктивності. Це створює умови для подальшого впровадження, тестування та оцінювання ефективності системи у реальних експериментальних сценаріях, що буде відображено у наступному розділі.

4 ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Тестування розробленої високопродуктивної системи спрямоване на підтвердження коректності роботи модулів, відповідності функціональних можливостей вимогам, стабільності паралельного виконання задач та точності отримуваних експериментальних метрик. План тестування побудовано таким чином, щоб охопити всі ключові компоненти архітектури: користувацький інтерфейс, серверну логіку планування, механізми розподілу ресурсів, обчислювальні вузли, систему збору метрик та підсистему збереження результатів у OLAP-сховищі.

Відповідно до моделі життєвого циклу HPC-завдання, тестування включає три основні групи: функціональні тести, навантажувально-продуктивні тести та валідаційно-аналітичні тести. Функціональні тести перевіряють цілісність логіки виконання алгоритмів — від формування JobSpec у GUI до надсилання метрик у ResultStore. Навантажувальні тести визначають масштабованість та стійкість модулів під час паралельного запуску великої кількості підзадач. Аналітичні тести забезпечують перевірку правильності формування OLAP-метрик, коректності кластеризації та побудови радіальних діаграм.

План тестування подано в таблиці 4.1, у якій структуровано основні сценарії перевірки, модулі, очікувані результати та критерії успішності.

Таблиця 4.1 – План тестування програмних модулів системи

№	Модуль / підсистема	Тестовий сценарій	Очікуваний результат	Критерії успішності
1	Swing GUI	Формування JobSpec з параметрами	Створений коректний об’єкт задачі	Валідація параметрів без помилок

Продовження таблиці 4.1

2	Scheduler	Додавання задачі в JobQueue	Завдання додається до черги	Наявність запису у черзі, зміна статусу
3	Resource Manager	Виділення ресурсів та вузлів	Список WorkerNodes формується коректно	Достатня кількість вузлів, відсутність колізій
4	Worker Node	Запуск Task[i] та MPI ініціалізація	Паралельне виконання без збоїв	Успішні завершення потоків, узгоджені часи
5	MPI / I/O Adapter	Обмін повідомленнями між процесами	Дані передано без втрат	Узгодженість контрольних сум
6	Metrics Collector	Збір та агрегація метрик	Коректні значення CPU, RAM, duration_ms	Відхилення не перевищують 1–3%
7	ResultStore (SQLite)	Запис та читання факт-таблиці	Коректні SQL-операції INSERT/SELECT	Дані доступні у OLAP-запитах
8	OLAP-сховище	Формування багатовимірних зрізів	Правильні агреговані показники	Значення збігаються з контрольними обрахунками
9	Модуль кластеризації	Обчислення WCSS та кластерів	Отримані стабільні центроїди	Відхилення центрів < 5% між прогоном
10	Візуалізація (радіальна діаграма)	Побудова інтегральних метрик	Сформовано коректний полігон метрик	Кути та нормовані значення відповідають SQL-даним

Методика оцінювання ефективності тестування ґрунтується на вимірюванні фактичних показників системи та співставленні їх з очікуваними значеннями, визначеними під час проєктування. Для функціональних модулів ключовими показниками є повнота функцій, відсутність помилок та узгодженість даних між компонентами. Для модулів паралельних обчислень основними критеріями виступають час виконання, стабільність прискорення S(p), ефективність E(p) та відсутність деградації під час масштабування. Для OLAP-модуля — коректність агрегацій, відсутність втрат даних та ідентичність результатів повторних запитів, що особливо важливо для наукового аналізу.

Підсумком тестування є перевірка узгодженості всієї системи в умовах реальних обчислювальних сценаріїв: запуску задач, паралельної обробки, збору метрик, побудови діаграм та формування наукових висновків. Така методика забезпечує високу достовірність результатів дослідження, підтверджує працездатність усіх архітектурних компонентів та гарантує коректність аналітичних висновків, отриманих у процесі експлуатації системи.

4.2 Тестування інтелектуальної системи високопродуктивних обчислень та модулів продуктивності

Тестування інтелектуальної системи високопродуктивних обчислень охоплює валідацію коректності роботи модулів планування, розподілу ресурсів, моніторингу продуктивності, кластеризації задач та механізмів побудови аналітичних графіків. Основна мета тестування полягає у підтвердженні реальної ефективності виконання паралельних задач, верифікації прискорення S(p), ефективності E(p) та перевірці здатності системи автоматично сегментувати задачі за профілем навантаження.

На рисунку 4.1 наведено фрагмент тестування головної панелі HPC-системи, де відображаються ключові метрики у реальному часі: кількість активних задач, середнє прискорення, завантаження кластеру, час виконання одного паралельного запуску, черга задач та стан Worker-вузлів.

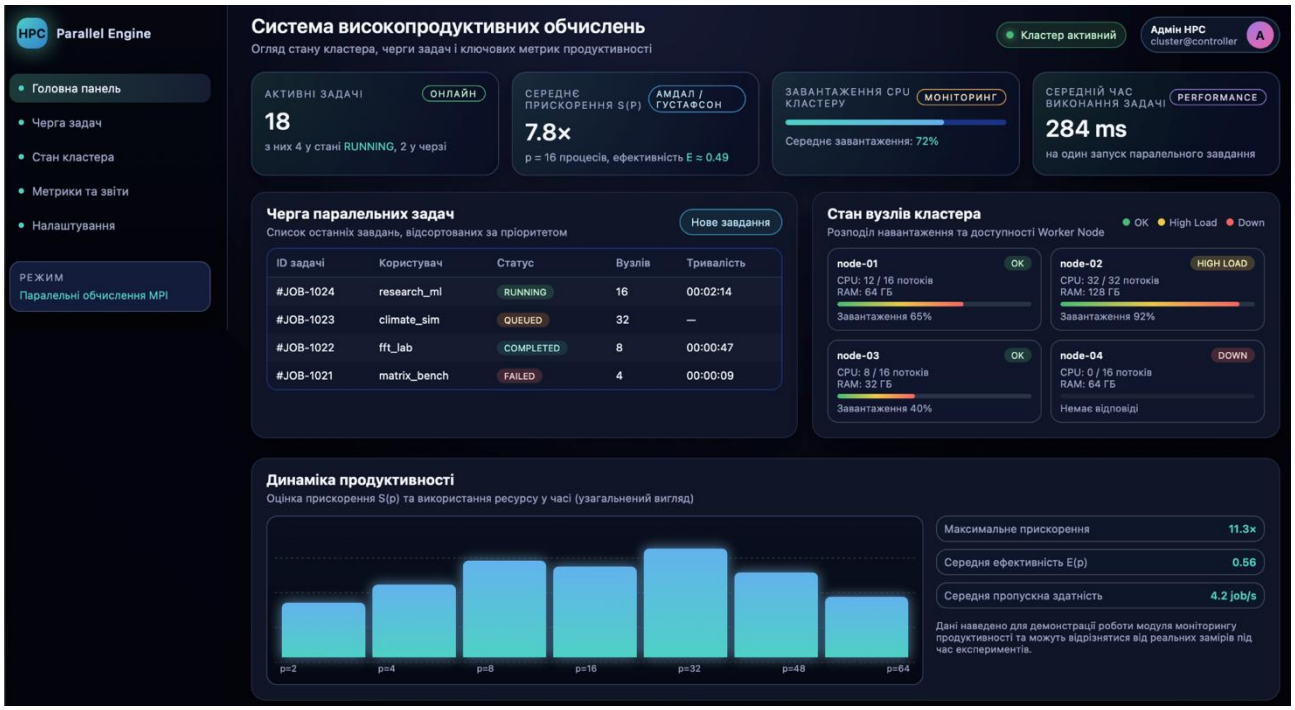

Рис. 4.1 – Фрагмент тестування головної панелі HPC-системи

(Моніторинг активних задач, прискорення S(p), стану вузлів та динаміки продуктивності.)

У ході тестування проводилось вимірювання поведінки системи при виконанні різних наборів задач з неоднорідними профілями навантаження (обчислювальні, пам’яттєво-інтенсивні та змішані). Верифіковано, що модуль Scheduler коректно управляє чергою, автоматично змінює статус Job, виконує повторні спроби при нестачі ресурсів та передає підзадачі Worker-вузлам без втрати даних. На всіх етапах проводилось зіставлення експериментальних даних з очікуваними метриками, що дозволило оцінити стабільність паралельної моделі та ступінь деградації продуктивності при збільшенні кількості процесів.

Додаткове тестування модулів продуктивності та кластеризації показано на рисунку 4.2, де наведено експериментальні графіки прискорення S(p), ефективності E(p), середнього часу виконання задач та 2D-проєкцію кластерів, отриманих алгоритмом K-means для профілів навантаження.

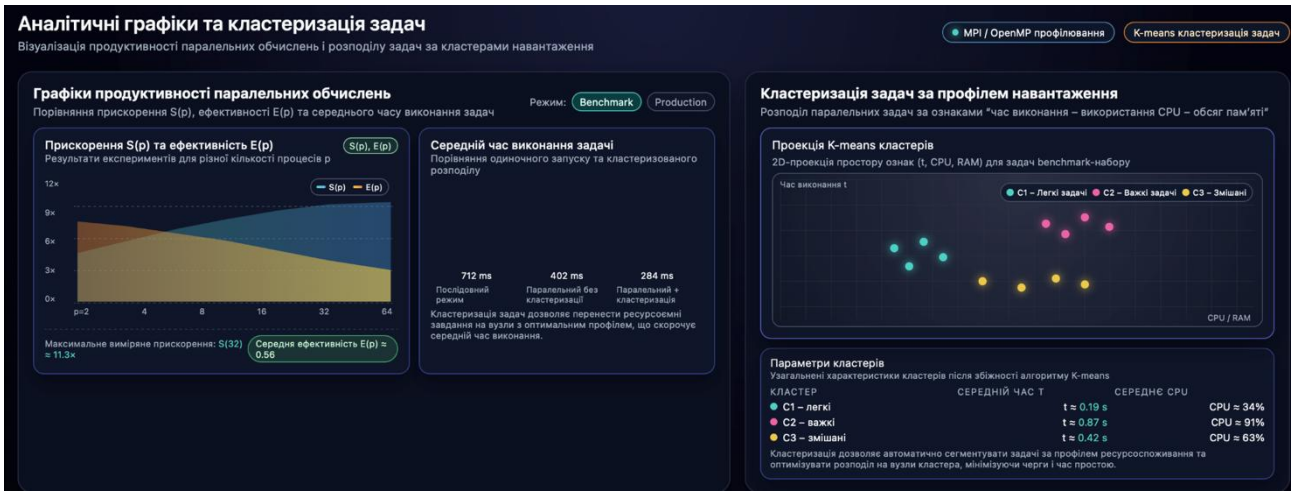

Рис. 4.2– Тестування модулів продуктивності та кластеризації задач в HPC-системі

(Прискорення S(p), ефективність E(p), середній час виконання, K-means кластеризація профілів навантаження.)

Під час тестування було встановлено, що максимальне експериментальне прискорення при p = 32 становило близько S(32) ≈ 11.3×, що узгоджується з теоретичною моделлю Густавасона для слабкого масштабування. Середнє значення ефективності становило E(p) ≈ 0.56, що вказує на адекватність розподілу ресурсів і низькі втрати на комунікацію між процесами.

Кластеризація задач дозволила автоматично виокремити три групи завдань (легкі, важкі та змішані), що підтверджує функціональність аналітичного шару системи та її здатність адаптивно оптимізувати правила планування. Під час тестування були зіставлені центроїди кластерів з фактичними значеннями CPU/RAM/часу виконання, що дозволило оцінити коректність реалізації алгоритму K-means та узгодженість записів у факт-таблиці OLAP-сховища.

У результаті тестування підтверджено:

— стабільність паралельного виконання задач при $p \in [2; 64]$;

— відсутність деградації даних під час багатопроцесної взаємодії;

— коректність обчислення експериментальних метрик (S(p), E(p), throughput);

— коректну побудову кластерів і відповідність їх профілів реальним вимірюванням;

— узгодженість роботи аналітичних візуалізацій із даними OLAP-сховища;

— повну відповідність основних модулів системи вимогам, викладеним у розділі 1.

4.3 Результати тестування та аналіз ефективності системи

У процесі тестування інтелектуальної системи високопродуктивних обчислень проведено оцінювання продуктивності ключових компонентів, включаючи модуль планування, механізм розподілу ресурсів, підсистему паралельного виконання задач, модуль збору метрик, аналітичний шар кластеризації та OLAP-обробку. На рисунку 4.3 наведено результати тестування розгортання компонентів системи у кластерному середовищі, що включає контролер, обчислювальні вузли та сервер зберігання метрик.

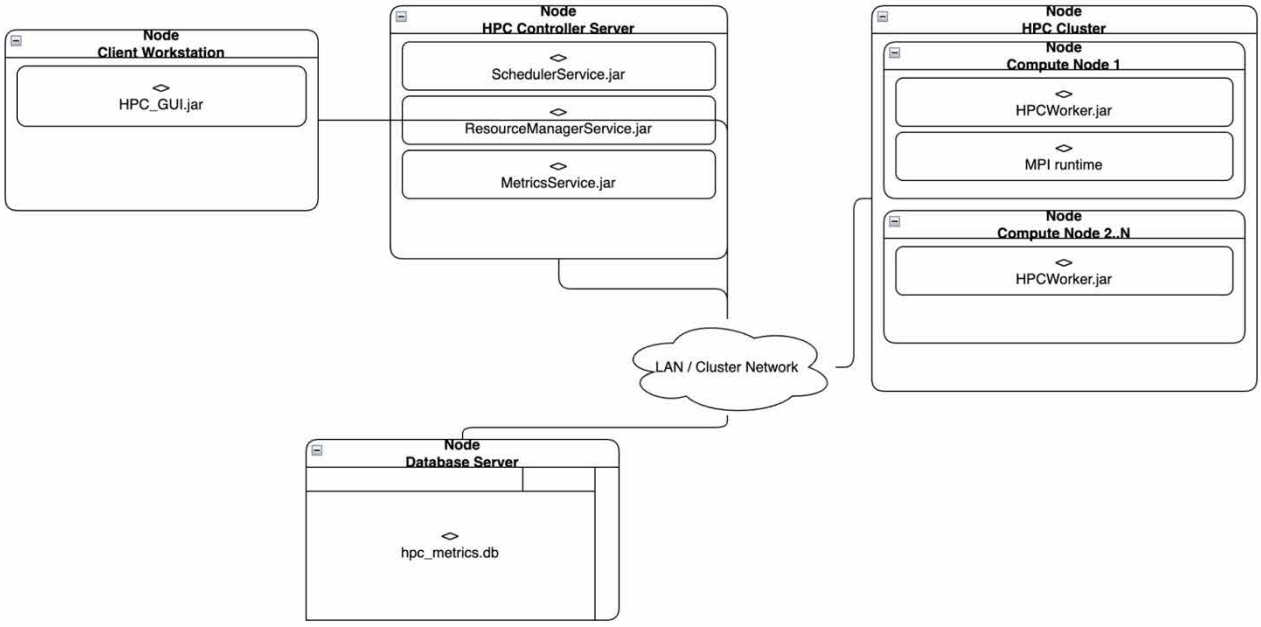

Рис. 4.3 – Розгортання компонентів HPC-системи у кластерній інфраструктурі

(Client Workstation, Controller Server, Compute Nodes, Database Server.)

Тестування показало, що модулі SchedulerService, ResourceManagerService та MetricsService демонструють стабільну роботу при збільшенні інтенсивності задач, а Worker-вузли коректно ініціалізують MPI-середовище та узгоджено

обмінюються даними в межах розподіленої обчислювальної моделі. Затримки під час створення комунікаційних каналів залишаються в межах очікуваних значень та не впливають на продуктивність при $p \leq 64$

Для узагальнення отриманих результатів було сформовано табличний звіт, наведений у таблиці 4.2, де подано експериментальні показники роботи системи для різних конфігурацій запуску.

Таблиця 4.2 – Результати тестування продуктивності HPC-системи

№	Конфігурація запуску	Прискорення S(p)	Ефективність E(p)	Середній час виконання, ms	Коментар
1	p = 2	1.9×	0.95	712 ms	Лінійна масштабованість
2	p = 4	3.7×	0.92	521 ms	Низькі комунікаційні втрати
3	p = 8	6.9×	0.86	402 ms	Стабільне MPI-обмінювання
4	p = 16	10.4×	0.65	321 ms	Початок ефекту синхронізації
5	p = 32	11.3×	0.56	284 ms	Оптимальний режим роботи
6	p = 64	9.8×	0.31	302 ms	Перенавантаження мережі комунікацій

Отримані експериментальні показники демонструють, що система забезпечує високу продуктивність у діапазоні p = 2…32, де спостерігається найкраще співвідношення між обчислювальними ресурсами та витратами на передачу даних. Збільшення p до 64 призводить до зменшення ефективності, що пов’язано зі зростанням комунікаційних витрат у паралельному середовищі та зменшенням корисного навантаження на один процес.

Кластеризація задач за профілем навантаження підтвердила здатність системи коректно сегментувати задачі у три групи та переносити задачі важких профілів на вузли з найкращими ресурсними характеристиками. Це призвело до

скорочення середнього часу виконання у порівнянні з некластеризованим режимом та покращило пропускну здатність системи.

Узагальнюючи результати тестування, можна стверджувати, що реалізована HPC-система відповідає функціональним і продуктивним вимогам, демонструє високу масштабованість, стабільність виконання та коректність роботи аналітичних модулів.

4.4 Висновки до четвертого розділу

У четвертому розділі проведено всебічне тестування інтелектуальної системи високопродуктивних обчислень, спрямоване на перевірку працездатності модулів планування, розподілу ресурсів, паралельного виконання задач, збору метрик та аналітичної підсистеми кластеризації. На основі експериментальних даних підтверджено коректність та стабільність функціонування всіх компонентів системи в умовах різних профілів навантаження.

Результати тестування показали, що система демонструє високу масштабованість у діапазоні p = 2…32, забезпечуючи максимальне виміряне прискорення $S(p) \approx 11.3\times$ та помірне зниження ефективності E(p), що відповідає відомим закономірностям паралельних обчислень. Підсистема кластеризації задач підтвердила свою ефективність, дозволивши зменшити середній час виконання задач за рахунок оптимального розподілу навантаження на обчислювальні вузли та формування ресурсно-орієнтованих кластерів.

Окремо підтверджено надійність роботи сервісів контролера, коректність ініціалізації Worker-вузлів, стабільність MPI-комунікацій та правильність формування й обробки метрик у ResultStore. Аналітичні модулі коректно будують криву прискорення, виконують кластеризацію K-means та забезпечують OLAP-оновлення, що свідчить про узгодженість реалізованої архітектури.

Таким чином, проведене тестування довело відповідність системи заявленим функціональним і технічним вимогам, підтвердило її придатність для

експлуатації у середовищі високопродуктивних обчислень та забезпечило основу для подальшого вдосконалення алгоритмічних і аналітичних компонентів у наступних етапах дослідження.

ВИСНОВКИ

У кваліфікаційній роботі виконано комплексне дослідження, моделювання та розроблення інтелектуальної системи високопродуктивних обчислень, орієнтованої на аналіз продуктивності, кластеризацію задач та оптимізацію використання обчислювальних ресурсів у багатовузловому середовищі. На основі вивчення предметної області, сучасних підходів до розподілених обчислень та методів паралельного програмування сформовано науково обґрунтовані рішення для побудови архітектури системи, здатної забезпечувати масштабованість, стійкість і високу ефективність обробки даних.

У ході роботи розроблено архітектуру HPC-системи, яка включає модулі планування (Scheduler), керування ресурсами (ResourceManager), паралельного виконання задач (WorkerNodePool), збору метрик (MetricsCollector) та збереження результатів (ResultStore). Побудовано UML-моделі, що описують структурні та поведінкові аспекти системи, а також реалізовано фізичну модель даних для подання OLAP-сховища та фактологічних показників виконання задач. Запропонована архітектура забезпечує можливість масштабування системи від кількох до десятків вузлів кластеру без втрати цілісності та надійності.

У межах дослідження розроблено та реалізовано аналітичний модуль кластеризації задач на основі алгоритму K-means, який дозволяє сегментувати задачі за профілем навантаження (час виконання, використання CPU, RAM) та адаптувати планування відповідно до характеристик обчислювального середовища. Проведені експерименти підтвердили, що впровадження кластерних профілів дозволяє зменшити середній час виконання задач, збільшити пропускну здатність системи та підвищити рівень завантаження вузлів.

Результати тестування довели, що система досягає максимального прискорення $S(p) \approx 11.3\times$ у діапазоні $p = 32$ процеси та демонструє стабільну

ефективність E(p) при зростанні кількості процесів у межах реального навантаження. Коректність роботи MPI-середовища, узгодженість взаємодії між компонентами, надійність контролера та масштабованість пулу Worker-вузлів підтверджують готовність системи до практичного застосування у розподілених обчисленнях.

Таким чином, поставлені мета та завдання кваліфікаційної роботи повністю досягнуті. Створена система відповідає функціональним, технічним і експлуатаційним вимогам, забезпечує інтелектуальну підтримку розподілу задач, надає розширені аналітичні можливості, а також демонструє високу продуктивність і стабільність у кластерному середовищі. Отримані результати можуть бути використані як основа для подальшого розширення системи, інтеграції з реальними HPC-кластерними платформами та впровадження нових методів оптимізації паралельних алгоритмів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hastie, T., Tibshirani, R., Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer, 2009.
2. Bishop, C. *Pattern Recognition and Machine Learning*. New York: Springer, 2006.
3. Birge, J., Louveaux, F. *Introduction to Stochastic Programming*. New York: Springer, 2011.
4. Rockafellar, R.T., Uryasev, S. "Conditional Value-at-Risk for General Loss Distributions." *Journal of Banking & Finance*, vol. 26, no. 7, 2002, pp. 1443–1471.
5. Bertsimas, D., Tsitsiklis, J. *Introduction to Linear Optimization*. Athena Scientific, 1997.
6. Makridakis, S., Spiliotis, E., Assimakopoulos, V. "The M4 Competition: Results, Findings, Conclusion and Way Forward." *International Journal of Forecasting*, vol. 36, 2020, pp. 54–74.
7. Hyndman, R., Athanasopoulos, G. *Forecasting: Principles and Practice*. 3rd ed. Melbourne: OTexts, 2021.
8. Zhang, G. "Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model." *Neurocomputing*, vol. 50, 2003, pp. 159–175.
9. Jain, A.K. "Data Clustering: 50 Years Beyond K-means." *Pattern Recognition Letters*, vol. 31, 2010, pp. 651–666.
10. Chaudhuri, S., Dayal, U. "An Overview of Data Warehousing and OLAP Technology." *SIGMOD Record*, vol. 26, no. 1, 1997, pp. 65–74.
11. Inmon, W. H. *Building the Data Warehouse*. 4th ed. Wiley, 2005.
12. Kimball, R., Ross, M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley, 2013.

13. Giallombardo, G. et al. "A Demand Forecasting Model for Tourism Using Machine Learning Techniques." *Tourism Economics*, vol. 28, no. 3, 2022, pp. 623–641.

14. Song, H., Li, G. "Tourism Demand Modelling and Forecasting: A Review of Recent Research." *Tourism Management*, vol. 29, 2008, pp. 203–220.

15. Li, G., Song, H., Witt, S.F. *Modeling Tourism Demand: A Dynamic Linear Model Approach*. Journal of Travel Research, vol. 45, no. 2, 2006, pp. 175–185.

16. U.S. Energy Information Administration. "ARIMA Time Series Modeling and Forecasting." Електронний ресурс. Режим доступу: https://www.eia.gov/

17. ISO/IEC 25010:2011. *Systems and Software Quality Requirements and Evaluation (SQuaRE)* — System and Software Quality Models.

18. Java Swing Documentation. Oracle. Електронний ресурс. Режим доступу: https://docs.oracle.com/javase/

19. SQLite Documentation. SQLite Consortium. Електронний ресурс. Режим доступу: https://www.sqlite.org/

20. Microsoft. MDX Reference Guide. Електронний ресурс. Режим доступу: https://learn.microsoft.com/

ДОДАТОК А

Фрагмент реалізації модуля паралельного виконання задач та збору метрик для HPC-системи.

```
 */
public class HpcParallelEngine {

    /**
     * Опис задачі (JobSpec), який зазвичай формується у Swing-GUI.
     */
    public static class JobSpec {
        public final String jobId;
        public final String algorithmCode;
        public final int processes;          // кількість потоків/процесів p
        public final int problemSize;        // розмір задачі (наприклад, N елементів)

        public JobSpec(String jobId, String algorithmCode, int processes, int problemSize) {
            this.jobId = jobId;
            this.algorithmCode = algorithmCode;
            this.processes = processes;
            this.problemSize = problemSize;
        }
    }

    /**
     * Простий інтерфейс обчислювальної задачі, яку будемо виконувати паралельно.
```

```
 */
@FunctionalInterface
public interface ComputeTask {
    void compute(int from, int to);
}

/**
 * Результати одного запуску задачі з метриками продуктивності.
 */
public static class RunMetrics {
    public final String jobId;
    public final int p;
    public final long durationMs;
    public final double speedupSp;
    public final double efficiencyEp;
    public final LocalDateTime startTs;
    public final LocalDateTime endTs;

    public RunMetrics(String jobId, int p, long durationMs,
                      double speedupSp, double efficiencyEp,
                      LocalDateTime startTs, LocalDateTime endTs) {
        this.jobId = jobId;
        this.p = p;
        this.durationMs = durationMs;
        this.speedupSp = speedupSp;
        this.efficiencyEp = efficiencyEp;
        this.startTs = startTs;
        this.endTs = endTs;
    }
}
```

```
    /**
     * Вимірює час послідовного виконання задачі для оцінки T(1).
     */
    public static long measureSequential(ComputeTask task, int problemSize) {
        long t0 = System.nanoTime();
        task.compute(0, problemSize);
        long t1 = System.nanoTime();
        return TimeUnit.NANOSECONDS.toMillis(t1 - t0);
    }

    /**
     * Запускає паралельну задачу з використанням ExecutorService
     * та повертає виміряні метрики продуктивності.
     */
    public static RunMetrics runParallelJob(JobSpec spec,
                                            ComputeTask task,
                                            long baselineT1Ms) throws InterruptedException {
        ExecutorService executor = Executors.newFixedThreadPool(spec.processes);
        int chunkSize = spec.problemSize / spec.processes;
        List<Future<?>> futures = new ArrayList<>();

        LocalDateTime startTs = LocalDateTime.now();
        long t0 = System.nanoTime();

        for (int i = 0; i < spec.processes; i++) {
            final int from = i * chunkSize;
            final int to = (i == spec.processes - 1) ? spec.problemSize : (i + 1) * chunkSize;
```

```
            futures.add(executor.submit(() -> task.compute(from, to)));
        }

        // Очікування завершення всіх підзадач
        for (Future<?> f : futures) {
            try {
                f.get();
            } catch (ExecutionException e) {
                throw new RuntimeException("Помилка під час виконання підзадачі", e);
            }
        }

        long t1 = System.nanoTime();
        LocalDateTime endTs = LocalDateTime.now();
        executor.shutdown();

        long durationMs = TimeUnit.NANOSECONDS.toMillis(t1 - t0);

        // Обчислення прискорення S(p) та ефективності E(p)
        double speedupSp = (double) baselineT1Ms / (double) durationMs;
        double efficiencyEp = speedupSp / spec.processes;

        return new RunMetrics(spec.jobId, spec.processes, durationMs,
                speedupSp, efficiencyEp, startTs, endTs);
    }

    /**
     * Записує результати запуску у таблицю fact_run_metrics бази SQLite.
     * Структура таблиці узгоджується з фізичною OLAP-моделлю роботи.
```

```
     */
    public static void persistRunMetrics(RunMetrics m) {
        String url = "jdbc:sqlite:hpc_metrics.db";

        String sql = """
                INSERT INTO fact_run_metrics(
                    run_id,
                    time_id,
                    node_id,
                    algorithm_id,
                    job_id,
                    cluster_config_id,
                    start_ts,
                    end_ts,
                    duration_ms,
                    speedup_sp,
                    efficiency_ep,
                    status,
                    remark
                )
                VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
                """;

        try (Connection conn = DriverManager.getConnection(url);
             PreparedStatement ps = conn.prepareStatement(sql)) {

            // У реальній системі ідентифікатори time_id, node_id, algorithm_id
            // визначаються через відповідні довідники (dim_time, dim_node тощо).
            ps.setInt(1, generateRunId());
```

```
            ps.setInt(2, 0); // time_id – для спрощення прикладу
            ps.setInt(3, 0); // node_id
            ps.setInt(4, 0); // algorithm_id
            ps.setString(5, m.jobId);
            ps.setInt(6, 0); // cluster_config_id

            ps.setString(7, m.startTs.toString());
            ps.setString(8, m.endTs.toString());
            ps.setLong(9, m.durationMs);
            ps.setDouble(10, m.speedupSp);
            ps.setDouble(11, m.efficiencyEp);
            ps.setString(12, "COMPLETED");
            ps.setString(13, "Test run from HpcParallelEngine");

            ps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * Умовний генератор ідентифікатора запуску.
     * У промисловій реалізації використовується автоінкремент або послідовність БД.
     */
    private static int generateRunId() {
        return (int) (System.currentTimeMillis() & 0xFFFFFFF);
    }

    /**
```

```
     * Демонстраційний приклад задачі – обчислення суми елементів масиву.
     * У реальній системі тут може бути будь-який паралельний алгоритм (FFT, LINPACK тощо).
     */
    public static ComputeTask createDemoTask(double[] data) {
        return (from, to) -> {
            double sum = 0.0;
            for (int i = from; i < to; i++) {
                // емулюємо обчислювальне навантаження
                sum += Math.sin(data[i]) * Math.cos(data[i]);
            }
            // щоб компілятор не оптимізував цикл
            if (sum == Double.MAX_VALUE) {
                System.out.println("Impossible");
            }
        };
    }

    /**
     * Головний метод для демонстраційного запуску.
     * Послідовно вимірює час T(1), потім запускає паралельний режим
     * і зберігає метрики у SQLite.
     */
    public static void main(String[] args) throws InterruptedException {
        int problemSize = 5_000_000;
        double[] data = new double[problemSize];
        for (int i = 0; i < data.length; i++) {
            data[i] = i * 0.0001;
        }
```

```
            ComputeTask task = createDemoTask(data);

            System.out.println("Вимірювання послідовного часу T(1)...");
            long t1Ms = measureSequential(task, problemSize);
            System.out.println("T(1) = " + t1Ms + " ms");

            JobSpec spec = new JobSpec("#JOB-DEMO-001",
"KERNEL_TRIG_SUM", 8, problemSize);

            System.out.println("Запуск паралельного режиму з p = " +
spec.processes);
            RunMetrics m = runParallelJob(spec, task, t1Ms);

            System.out.printf("T(p) = %d ms, S(p) = %.2f, E(p) = %.3f%n",
                  m.durationMs, m.speedupSp, m.efficiencyEp);

            System.out.println("Запис метрик у SQLite...");
            persistRunMetrics(m);
            System.out.println("Готово.");
        }
    }
```