МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження фреймворків для обробки графів у завданнях аналізу великих даних
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-21-1

Чан С. Т.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник доц. Кравец Н. С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ___________ З. В. Дудар

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри ______________
(підпис)
«____»____________ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Чан Суан Тхангу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження фреймворків для обробки графів у завданнях аналізу великих даних»

затверджена наказом університету від « 03 » березня 2023 р. № 302 Ст

2. Термін подання студентом роботи до екзаменаційної комісії «16» травня 2023 р.

3. Вихідні дані до роботи Наукові роботи та документація до фреймворків Pregel, Giraph++, GiraphX, GPS, X-Stream, Medusa, GraphHP, Chaos, Cassovary, GRACE, TOTEM, Apache Giraph, Apache Spark GiraphX. СУБД для графів Microsoft Azure Cosmos DB, Neo4j, Virtuoso, Ontotext GraphDB, Amazon Neptune.

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної галузі і постановка задачі, дослідження особливостей обробки графів та фреймворки, дослідження баз даних та систем управління базами даних для графів, аналіз отриманих результатів дослідження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.01.2023	виконано
2	Розробка постановки задачі	31.01.2023	виконано
3	Дослідження особливостей графів та їх обробки	15.02.2023	виконано
4	Дослідження роботи фреймворків, їх архітектурних рішень та моделей обробки	05.03.2023	виконано
5	Огляд та аналіз існуючих рішень у сфері фреймворків для обробки великих графів	25.03.2022	виконано
6	Дослідження СУБД для графів як альтернативи фреймворкам	05.04.2023	виконано
7	Огляд та аналіз існуючих СУБД для графів та їх порівняння	20.04.2023	виконано
8	Аналіз результатів дослідження та розробка рекомендацій щодо вибору фреймворку/СУБД	30.04.2023	виконано
9	Підготовка пояснювальної записки	08.05.2023	виконано
10	Підготовка презентації та доповіді	10.05.2023	виконано
11	Нормоконтроль та рецензування	11.05.2023	виконано
12	Попередній захист	13.05.2023	виконано
13	Занесення роботи в електронний архів	15.05.2023	виконано
14	Допуск до захисту	16.05.2023	виконано

Дата видачі завдання 20 січня 2023 р.

Студент ____
(підпис)

Керівник роботи ______________________ доц. Кравець Н. С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 100 с., 29 рис., 10 табл., 52 джер.

АЛГОРИТМИ ОБРОБКИ ГРАФІВ, АРХІТЕКТУРА СИСТЕМИ, БАЗА ДАНИХ, ГРАФ, ГРАФОВА СТРУКТУРА ДАНИХ, ЕТАПИ ОБЧИСЛЕНЬ, ЕФЕКТИВНІСТЬ РОБОТИ, КООРДИНАЦІЯ ОБЧИСЛЕНЬ, КРИТЕРІЇ ВИБОРУ, МЕРЕЖІ, МЕТРИКА ЯКОСТІ, РОЗПОДІЛЕННЯ ДАНИХ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, СТРУКТУРА ДАНИХ, ФРЕЙМВОРК, AMAZON NEPTUNE, APACHE GIRAPH, APACHE HADOOP, APACHE SPARK GRAPHX, BULK SYNCHRONOUS PARALLEL, CASSOVARY, CHAOS, GATHER APPLY SCATTER, GIRAPH++, GIRAPHX, GRACE, GRAPH PROCESSING SYSTEM, GRAPHDB, GRAPHHP, HDFS, MAPREDUCE, MEDUSA, MICROSOFT AZURE COSMOS DB, NEO4J, NO-SQL, PARTITIONING, PREGEL, TOTEM, VIRTUOSO, X-STREAM.

Об’єктом дослідження є фреймворки для обробки великих графів.

Метою цієї роботи є створення структурованого уявлення основних принципів роботи, проведення дослідження властивостей та проблемних областей фреймворків для обробки графів та їх застосування у існуючих програмних системах, що у подальшому надасть основу для прийняття рішень з використання конкретних їх екземплярів в проєктах чи для супутніх досліджень з цієї сфери.

Методами дослідження є загально логічні, теоретичні та емпіричні методи які використовувалися для обробки та аналізу різних джерел даних, таких як книжкові видання, наукові статті, описані приклади використання фреймворків, документація програмних рішень, тощо.

В результаті роботи було встановлено загальне уявлення про сферу обробки та використання великих графів, було розібрано та проаналізовано внутрішню роботу фреймворків, основні проблеми та виклики, різноманіття їх архітектурних рішень, властивостей та особливостей їх роботи, а також розроблені рекомендації по вибору необхідного фреймворку/СУБД для роботи з графами.

AMAZON NEPTUNE, APACHE GIRAPH, APACHE HADOOP, APACHE SPARK GRAPHX, BULK SYNCHRONOUS PARALLEL, CASSOVARY, CHAOS, DATA PARTITIONING, DATA STRUCTURE, DATABASE, DATABASE MANAGEMENT SYSTEM, EFFICIENCY, FRAMEWORK, GATHER APPLY SCATTER, GIRAPH++, GIRAPHX, GRACE, GRAPH, GRAPH DATA STRUCTURE, GRAPH PROCESSING ALGORITHMS, GRAPH PROCESSING SYSTEM, GRAPHDB, GRAPHHP, HDFS, MAPREDUCE, MEDUSA, MICROSOFT AZURE COSMOS DB, NEO4J, NETWORKS, NO-SQL, PARTITIONING, PREGEL, PROCESSING COORDINATION, PROCESSING STAGES, QUALITY METRIC, SELECTION CRITERIA, SYSTEM ARCHITECTURE, TOTEM, VIRTUOSO, X-STREAM.

The object of research is frameworks for processing large graphs.

The aim of the work is to create a structured knowledge about key principles of graph processing framework operations, to conduct a study of the properties and problem areas of graph processing solutions and their applications in existing software systems, which in the future will provide a basis for making decisions about usage of specific framework instances in projects or for further research in this area.

The study methods are logical, theoretical and empirical methods that were used for processing and analyzing various data sources, such as book publications, scientific articles, documented examples of frameworks usage, software solution documentations, etc.

As a result of this work, a general idea of large graph processing field was established. The internal work of frameworks, main problems and challenges, variety of their architectural solutions, properties and features were analyzed. Additionally, by using the obtained data the recommendations for selecting the suitable framework/DBMS for graph processing were developed.

Умови публікації пояснювальної записки

Я, Чан Суан Тханг,

(прізвище, ім’я, по батькові)

студент(ка) групи ІПЗм-21-1, здобувач вищої освіти на другому (магістерському) рівні

кафедра програмної інженерії,

(назва кафедри)

заявляю: моя кваліфікаційна робота на тему «Дослідження фреймворків для обробки графів у завданнях аналізу великих даних»,

(назва роботи)

що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 9
1 Аналіз проблемної області та постановка задачі 12
1.1 Аналіз проблемної області дослідження 12
1.2 Сфери використання графів і фреймворків для роботи з ними 15
1.3 Постановка задачі 21
2 Дослідження особливостей обробки графів та фреймворки 23
2.1 Загальні представлення графу 23
2.2 Цінність графових фреймворків та структур даних 27
2.3 Основні виклики для задач опрацювання графів 28
2.4 Загальна схема обчислень у графі 29
2.5 Класифікація алгоритмів для роботи з графам 30
2.6 Основні проблеми імплементації алгоритмів у фреймворках і програмних системах 35
2.7 Категорії архітектурних моделей обробки 37
2.8 Координація у розподілених систем 42
2.9 Підходи до розподілу/обробки даних графу в фреймворках 47
2.10 Фреймворки для роботи з графами 51
3 Бази даних та системи управління базами даних для графів 59
3.1 СУБД для графів як аналог фреймворкам 59
3.2 Огляд множини альтернатив 59
3.3 Аналіз характеристик обраних СУБД 63
3.3.1 Критерії оцінки характеристик баз даних 64
3.3.2 Розробка шкали оцінок за критеріями 65
3.3.3 Моделювання задачі прийняття рішення для порівняння та вибору найкращої БД 66
3.3.4 Розрахунок задачі прийняття рішення 71
4 Аналіз результатів дослідження 75
4.1 Основні зібрані дані під час дослідження фреймворків та СУБД для роботи з графами 75

4.2 Аналіз отриманих даних 77
4.3 Рекомендації з вибору фреймворків та СУБД для використання 78
Висновки 80
Перелік джерел посилання 81
Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 86
Додаток А. Звіт з результатів перевірки на унікальність тексту 87
Додаток Б. Презентація 88
Додаток В. Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам дсту 3008:2015 98
Додаток Г. Апробація кваліфікаційної роботи 99

ВСТУП

У сучасному світі майже в усіх процесах застосовують технічні і програмні засоби чи рішення. Цими процесами можуть бути будь-які налаштовані алгоритми роботи у бізнесі, на підприємствах та виробництвах, в інституціях та координованих групах, навіть у звичайному повсякденному житті. Вони допомагають уніфікувати та оптимізувати кроки досягнення цілей у процесах і під час цього також збирають дані по результатам своєї роботи і реального процесу. Таким чином будь-яка програмна система, яка має «пам'ять» для функціонування та виконання своїх задач збирає велику кількість даних пов'язаних зі своєю роботою.

З розвитком технічних засобів, а саме підвищенням швидкості опрацювання та можливостей обробки великих обсягів задач та даних, програмні системи почали ставати більш складними і комплексними. А розвиток технологій зберігання даних і їх здешевлення дало можливість системам зберігати неймовірно великі обсяги даних. Складні системи з безліччю даних, що зберігаються становяться все більш складними, зв'язки між різними системами розширюються через інтеграції – це призводить до ще більшого розширення даних, що зберігаються і зв'язку між ними. Разом з тим розвивається і сфера аналізу даних – через збільшення можливостей обробки і об'єму та характеру даних. Ця сфера є невід'ємною частиною великих та важливих бізнес процесів (у розрізі бізнес логіки програмних систем та самих реальних процесів які супроводжують програмні рішення) бо надає велику кількість корисної інформації щодо самих процесів, їх слабких сторін і можливостей для удосконалення чи оптимізації. Обробка великих даних становиться повсякденною задачею для будь-яких складних чи великих систем. Обробка даних нерозривно пов'язана із характеристиками даних – їх кількості, структури, способу збереження, зв'язку, методів обробки та самого програмного забезпечення, що їх зберігає. Також для обробки великих даних є важливими самі умови середи виконання цієї задачі – умови виконання, цілі обробки, особливості самих даних і т. п. Ці потреби привели до появи множини різноманітних структур збереження та обробки даних. Однією з найвідоміших альтернативних структур

збереження даних, до реляційних та об’єктно орієнтованих, є графові структури. Граф – це структура даних яка представляє собою набір вершин та зв’язків між ними (ребра). У контексті збереження даних вершини графу можуть представляти собою певний набір даних, а ребра їх зв’язки (між іншими вершинами), які в свою чергу можуть містити додаткову інформацію. Для збереження даних у подібних структурах і їх обробки існують різноманітні програмні рішення, такі як фреймворки та бази даних. Інша структура даних дозволяє по іншому обробляти та виконувати задачі, а також змінити сам підхід до самого використання даних та процесу їх обробки. Вона ставить у центр уваги не самі дані, а їх зв’язок та властивості цих зв’язків. Таким чином вона надає нові можливості для вирішення різноманітних задач з даними, являється оптимальною для застосування в вирішенні ряду завдань для яких використовуються алгоритми з теорії графів. Але їх використання та загальна інформація про них не є достатньо розповсюдженою відносно фреймворків для роботи з реляційними та об’єктовими структурами. В навчальних закладах, при створенні нових проєктів, у сфері про збереження та обробки даних у першу чергу розглядають саме реляційні структури. Об’єктові, NoSQL також стабільно набирають популярність та розголос у спільноті розробників, але графи та інструменти для роботи з ними ні (винятком можна виділити сферу штучного інтелекту та машинного навчання де вони широко застосовуються).

Метою цієї роботи є створення структурованого уявлення основних принципів роботи, проведення дослідження властивостей та проблемних областей фреймворків для обробки графів та їх застосування у існуючих програмних системах, що у подальшому надасть основу для прийняття рішень з використання конкретних їх екземплярів в проєктах чи для супутніх досліджень з цієї сфери.

Об’єктом дослідження є фреймворки для обробки великих графів.

Предметом дослідження є властивості графових фреймворків, їх характеристики, виклики/проблеми, що постають перед ними, архітектурні рішення та варіація стратегій/моделей обробки і розділення даних, що застосовуються в них.

Під час виконання атестаційної роботи були використані наступні методи дослідження:

– загально логічні (за допомогою аналізу, синтезу, індукції, абстрагування і узагальнення був проведений аналіз проблемної області дослідження, виявлені основні проблемні напрями в роботі об'єкту дослідження та проаналізовані загальні архітектурні рішення, моделі обробки, стратегії розподілення графів та їх взаємозв'язки в комплексі цілого фреймворку);

– теоретичні (був застосований метод сходження від абстрактного до конкретного під час аналізу існуючих програмних рішень та прикладів їх застосування);

– емпіричні (було використано порівняння для відображення та аналізу різноманіття програмних/системних рішень у сфері обробки графів).

Для виконання цілей дослідження були використані різноманітні джерела інформації (книжкові видання, наукові статті, описані приклади використання фреймворків, документація програмних рішень, тощо), що висвітлюють внутрішній устрій фреймворків, їх мотивації і середовище використання. Результати опрацювання були агреговані та структуровані в роботі. Вони допоможуть отримати базові знання та уявлення про фреймворки і супутні інструменти для обробки великих графів, що в свою чергу, можуть бути використані як основа для рішення вибору використання графових фреймворків у проєктах чи створенні власних з урахуванням перелічених особливостей, в встановленні основних напрямків чи бути однією з інформаційних основ/джерел для подальших досліджень на супутні теми обробки графів, фреймворків, методів та рішень роботи з великими даними та графами. Кваліфікаційна робота була апробована на 27-му Міжнародному молодіжному форумі «Радіоелектроніка і молодь у XXI столітті», що проводилася у Харківському національному університеті 10 – 12 травня 2023 року.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області дослідження

Структура даних, що використовує програмне рішення чи інструменти для для обробки / опрацювання, має критично важливу роль у її функціонуванні. Через їх властивості (структур даних) відкриваються різні можливості для їх застосування, де кожен аспект може як підвищувати ефективність та функціональність застосунків, так і погіршити ці показники.

Збережені дані можуть використовуватися програмними рішеннями різними методами та для різних цілей. Більшість застосунків використовують даний функціонал для зберігання операційних даних бізнес логіки процесів, Це може бути як зберігання текстів книг у сервісі онлайн бібліотеки (які в свою чергу використовують для простого вилучення / читання, так і для пошуку певних слів, фраз чи закономірностей у них); зберігання профілю користувача у онлайн сервісах чи соціальних мережах; зберігання бізнес даних компанії, для звітності та побудови різноманітних стратегій на основі аналітичних звітів і т.п. Лідером у світі серед типів структур для зберігання даних є реляційна схема баз даних через її відносну простоту та ряду можливостей, що вона надає [1]. Але для специфічних задач альтернативні структури збереження даних можуть позитивно вплинути на ефективність і на саму побудову застосунку – різноманітний функціонал, що включає у себе роботу з цими даними. Наприклад реляційна схема структури збереження даних є логічною, простою для розуміння, відносно ефективною у плані внесення змін в наявні дані та зменшення розміру самого набору даних (все завдяки нормалізації, що дозволяє зменшити цей обсяг і надає єдине джерело знань для кожної сутності у БД) (див. рис. 1.1). Також через її популярність та відрізку часу, що пройшло з моменту застосування такого типу структури, існують безліч рішень і функціональних можливостей для роботи з ними: різноманітні бази даних, драйвери та бібліотеки під них, рішення інтеграції роботи з різними системами, можливості використовувати майже будь які мови програмування для взаємодії і множина різноманітних фреймворків для створення власних програмних рішень, що з ними працюють. Також протягом часу з’явились уніфіковані стандарти для

такої структури даних, тому ми можемо казати про схожу концепцію роботи бази даних, єдину мову запитів до БД, що в свою чергу полегшує роботу з ними для розробників. Але такий підхід має і свої недоліки – через нормалізацію, сутності в БД розподілені до різних таблиць, і зв'язки між ними використовують додаткові таблиці, через що, запити із використанням великої кількості зв'язків і сутностей вимагають великих витрат у обчислювальному плані і часу. Через ці та інші недоліки такого типу, у окремих задачах використовують інші структури даних, і графова є одною з них.

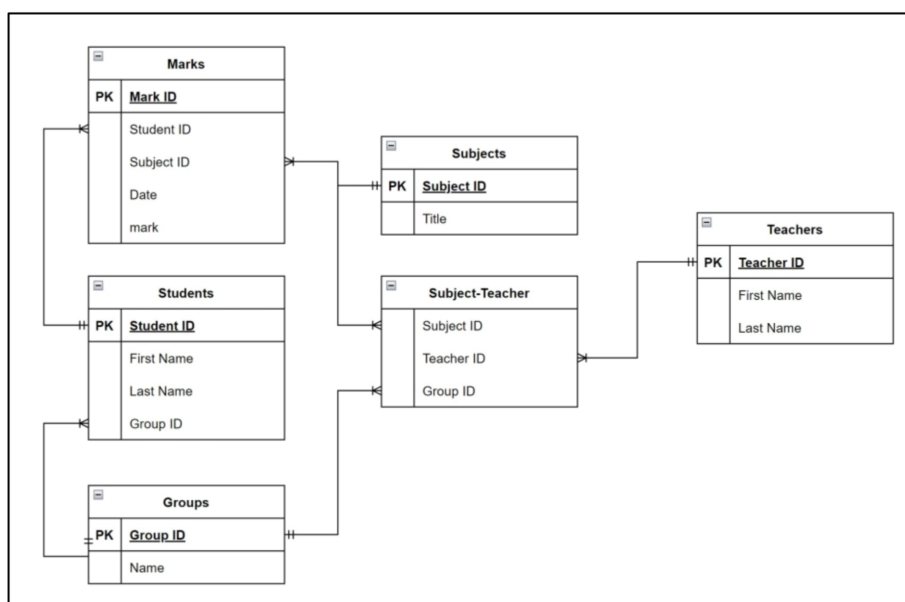

Рисунок 1.1 – ER діаграма. Приклад структури реляційної БД [Рисунок виконаний самостійно*]

Графова структура даних [2] побудована на основі моделі графу, де структура будується через відносини між елементами – вершинами, і характеризується за допомогою ребра (відносин) (див. рис. 1.2). Також у таких структурах використовується елемент «властивості», що може описувати ці відносини. Такий спосіб зберігання даних теоретично надає можливості більш вільно та ефективно встановлювати зв'язки між сутностями, що зберігаються у БД і, відповідно, більш швидко та ефективно виконувати запити пов'язані зі групою зв'язаних об'єктів. Ця особливість також дозволяє природнім чином (без застосування додаткових рішень)

встановлювати між об'єктами даних зв'язок та відстань, що корисно для завдань пошуку та оптимізації шляхів, аналізу наближеності, тощо. Так як граф за своєю абстрактною будовою та суттю моделює оточення з реального життя, то він відносно легко може вмістити себе ті самі моделі даних: дорожні карти, мережеві структури, соціальні зв'язки і інші подібні елементи. Графові бази даних можуть мати різні механізми зберігання таких структур. Можливо використовувати звичайну реляційну модель для збереження елементів графу у таблицях, і використовуючи абстракцію на більш високому рівні (рівні доступу до даних) надавати дані саме у вигляді графу (чи приймати специфічні запити). Інша, більш популярна схема роботи, включає в себе використання сховищ ключ-значення чи докуметно-орієнтовну схему, що робить такі рішення використанням NoSQL структур [3].

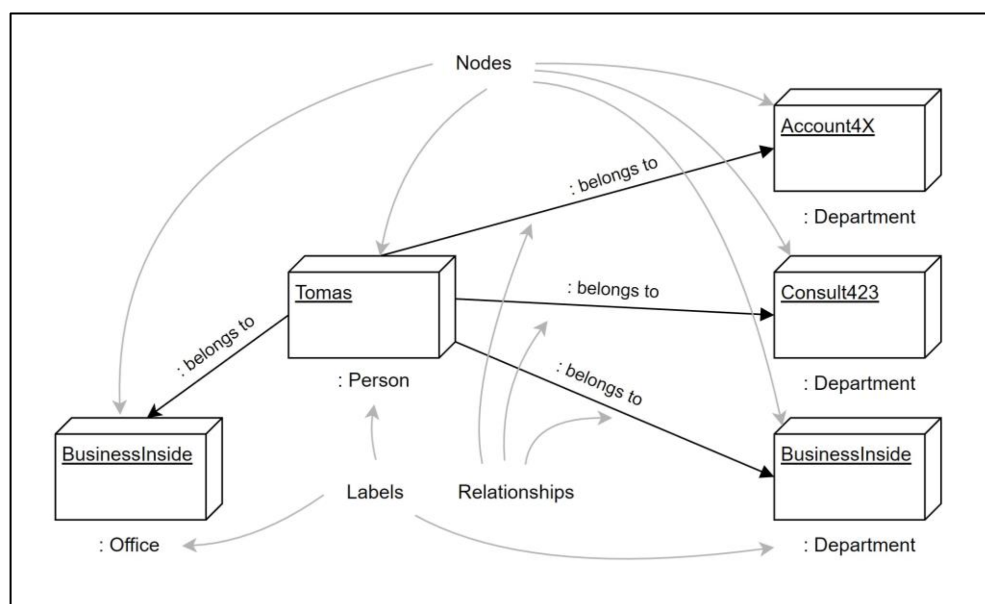

Рисунок 1.2 – Схематичне представлення структури графу з даними [Рисунок виконаний самостійно*]

Альтернативні звичайним SQL структури даних не є широко розповсюдженими і використовуються у особливих умовах, де стандартне популярне рішення не дає можливості швидко та легко виконувати поставлені задачі. Графові структури і фреймворки є одними з них – вони не є настільки

широко розповсюдженими і зазвичай виконують специфічні задачі. Але через свою унікальність та відмінність вони надають більше можливостей у роботі з даними, що мають велику кількість зв’язків і відносин. При використанні даних для типових задач обробки даних виникає питання, чи є варіант використання графових фреймворків більш доцільним ніж використання поширених та перевірених методів з фреймворками під реляційні дані. Теоретично графові структури надають перевагу у обробці великої кількості пов’язаних даних через можливості використовувати різноманітні алгоритми для обробки графів, а також у вирішенні специфічних задач, таких як визначення близькості даних, пошуки найкоротших або альтернативних шляхів, визначення популярних об’єктів, побудова баз знань з можливістю визначень супутніх/пов’язаних/контекстних знань, тощо. Але через специфіку графів та їх непопулярність, інформація про програмні інструменти, фреймворки, бази даних, застосування не є широко розповсюдженими, тому з’являється проблема у визначенні привабливості цих структур для використання, інструментів (фреймворків) для роботи з ними, самих їх можливостей. Через це виникає недостатність обізнаності у цій сфері, що сама по собі є одною з причин маловідомості існування, характеристик, особливостей роботи фреймворків для обробки графів.

1.2 Сфери використання графів і фреймворків для роботи з ними

Використання даних та структур основаних на графах є досить поширеним, приклади їх використання можна знайти у багатьох сферах пов’язаних з програмним забезпеченням – побудова фізичної структури зв’язку між серверами та базами даних, представлення зв’язку між сайтами/ресурсами у мережі інтернет, розподілення та консолідація даних у задачах розподіленої обробки даних, створення моделей структур з реального світу (наприклад молекулярні/атомні зв’язки), зв’язки даних та акаунтів у соціальних мережах, структурування даних у машинному навчанні і, власне, внутрішня побудова вузлів обробки у штучному інтелекті. Це досить обмежений список прикладів їх використання – з розвитком технологій штучного інтелекту і машинного навчання та збільшенням об’єму

інформації для опрацювання в різноманітних системах, фреймворки для обробки графів стають більш відомими та популярними. Також до розвитку технологій пов'язаних з графами причетні покращення технічних можливостей (потужність обробки даних, розвиток засобів збереження даних), поширення відносно нових концепцій як хмарні обчислення та архітектура, розвиток сфери розумних пристроїв IoT та постійне збільшення об'єму самих даних (інформації) у системах. Розглянемо більш докладно відомі сфери, що використовують графові структури даних та фреймворки для них для виконання своїх завдань.

Мережа Інтернет – кожен пристрій у комп'ютерній мережі, будь то роутер, сервер чи персональний комп'ютер, є вузлами/вершинами у мережі, а фізичні та інформаційні зв'язки між ними – ребрами цього «мережевого» графу. Маленькі «локальні» мережі об'єднані між собою у більші кластери, які в свою чергу об'єднані у ще більші – в кінцевому результаті створюючи неймовірно великий граф який називають мережею Інтернет (див. рис. 1.3). Через свою побудову вона є «реальним» представником графової структури, тому для роботи з нею доцільно використовувати теорію графів та різноманітні інструменти для роботи з графами. Типовою задачею у цій сфері є оптимізація трафіку – пошук найкращого шляху для доставки інформації, балансування навантаження між шляхами, аналіз топології для виявлення зв'язків та глухих кутів у мережах, оптимізація маршрутів. Такі завдання зазвичай повинні виконуватися у режимі реального часу, так як мережа і трафік в ній постійно змінюється.

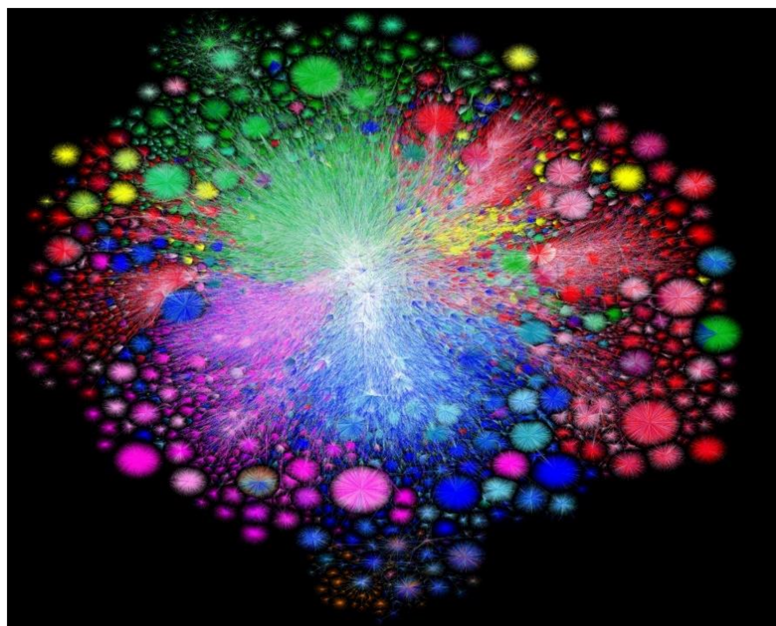

Рисунок 1.3 – Ілюстративне зображення графу Інтернет мережі [4]

World Wide Web (Всесвітня мережа) – відомий та важливий елемент сучасної цивілізації, роль якої важко описати та переоцінити. Станом на 2023 рік існує близько 1.13 мільярдів сайтів (не всі з них активні), і у 2016 році Google відзвітувала, що їх система індексувала близько 130 трильйонів веб-сторінок. Увесь спектр сучасної людської діяльності представлений у «всесвітній павутині», від навчальних та розважальних матеріалів, до особистих фотографій та висловлювань думок. Цю мережу сайтів часто представляють у вигляді графів, через зв’язки між ними, а також через особливості фізичного розташування і передачі даних між клієнтами/серверами (див. рис. 1.4). Різноманітні сервіси, що працюють у даній сфері, активно використовують теорію графів у програмному забезпеченні для виконання власних завдань з аналізу таких масивних мереж (наприклад пошукові сервіси Google, Yahoo, Bing).

Рисунок 1.4 – Ілюстративне зображення частини мережі World Wide Web з центральним вузлом Wikipedia [5]

Соціальні мережі – одні з найпопулярніших сервісів, що використовуються в Інтернеті – для спілкування, знайомств, поширення та споживання новин, ідей, заходів. Вони цілком можуть бути єдиним ресурсом, що використовує людина в мережі Інтернет, так як охоплює великий спектр усієї діяльності яка можлива в Інтернеті. Найвідоміші з них: Facebook (2,96 мільярдів активних користувачів), Twitter (450 мільйонів активних користувачів), Instagram (1,318 мільярда активних користувачів)[6][7][8]. В таких мережах акаунти користувачів можна представити вершинами графу, тоді зв’язки між ними будуть відображати їх взаємодію (підписки, стосунки чи статус між ними) (див. рис. 1.5). У системах вершинами можливо також представити інші компоненти: інформація, новини, категорії/тематики даних, медіа, тощо. Ребра ж формуються на основі різноманітних зв’язків. Наприклад: тематика, належність до певної географічної локації, лайки користувачів, підписки, цитування та посилання, авторство інформації, рейтинг і інше. Завдяки представленню такої системи і даних у графовому вигляді досягаються можливості

з їх аналізу для подальшого використання. Типовими прикладами використання такої системи є формування стрічки новин (показувати актуальні та найбільш цікаві для даного користувача), функція пошуку (збільшувати рівень релевантності їх результатів), показ реклами (на основі інтересів та інших характеристик користувача). Більш специфічними, але теж важливими є задачі соціального аналізу – виділення груп по інтересам/властивостям, пошук трендів і їх характеристик, OSINT задачі. Маючи великі обсяги даних, компанії власники соціальних мереж звертають увагу та розвивають розробку програмних рішень для роботи з великими графами/даними, щоб покращити свої сервіси і діставати більше цінної інформації з них.

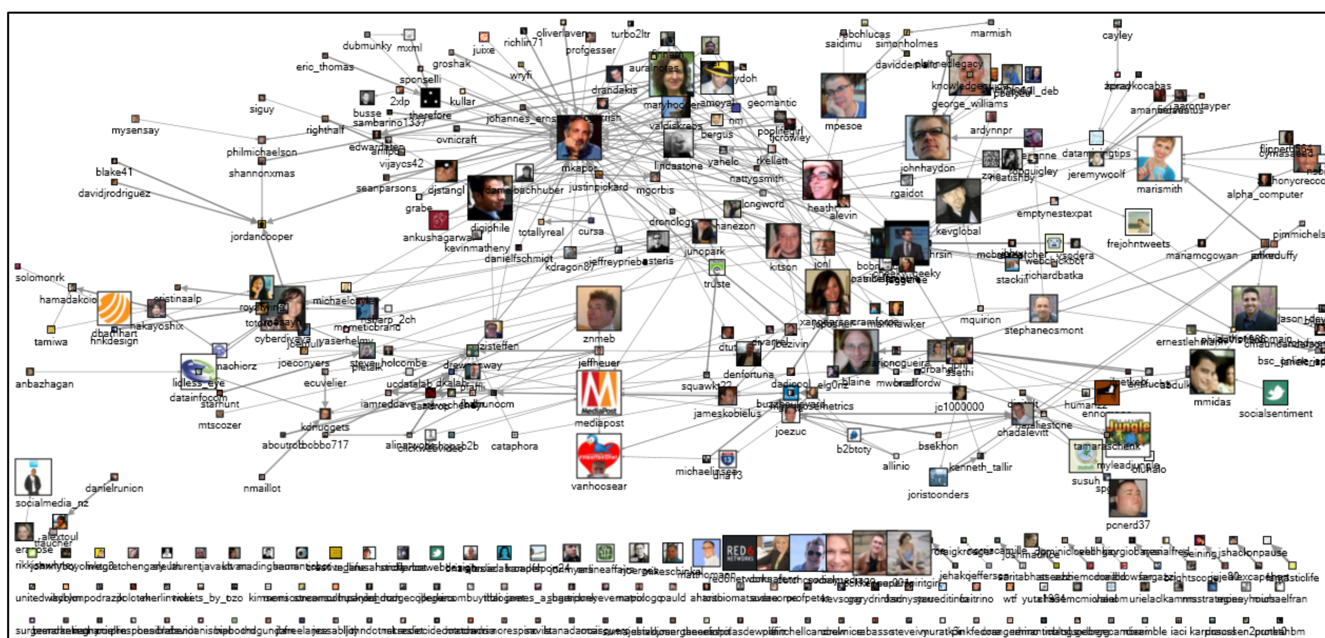

Рисунок 1.5 – Ілюстративне зображення соціального графу у мережі Twitter [9]

Мережі розумних пристроїв, мережеві системи - до цієї категорії ми можемо віднести не тільки IoT системи, але і системи розумних міст, розумного дому, різноманітні сервісні системи (у залізничних шляхах, логістичних компаніях), тощо (див. рис. 1.6) [10]. Ці програмні засоби збирають велику кількість даних, і також оперують графовими структурами для вирішення задач своєї проблемної області [11]. Наприклад логістичні компанії мають змогу використовувати графи для представлення географії де вони оперують: шукати найкоротші можливі шляхи

доставки, прогнозувати час перевезення, розраховувати витрати під час доставки чи змінювати маршрути у непередбачених ситуаціях. IoT та розумні міста/дома – область схожа за структурою з мережею Інтернет у плані задач з об’єднання великої кількості пристроїв у єдину мережу. Окрім представлення самої мережі за граф, такі системи збирають значну кількість даних, що формують графові структури (за місцезнаходженням, приналежності до певного об’єкта чи категорії даних, за пов’язаністю до інших пристроїв/вузлів) і які використовуються для задоволення потреб проблемної області. Графові алгоритми можуть бути використані для аналізу ситуації/обстановки за отриманими даними і створенні відповідних планів дій до них. Наприклад масштабування і балансування електроенергії в великій мережі на основі метрик її генерування і споживання у реальному часі. Або керування трафіком доріг через мережу світлофорів на основі даних про кількість машин на вулицях.

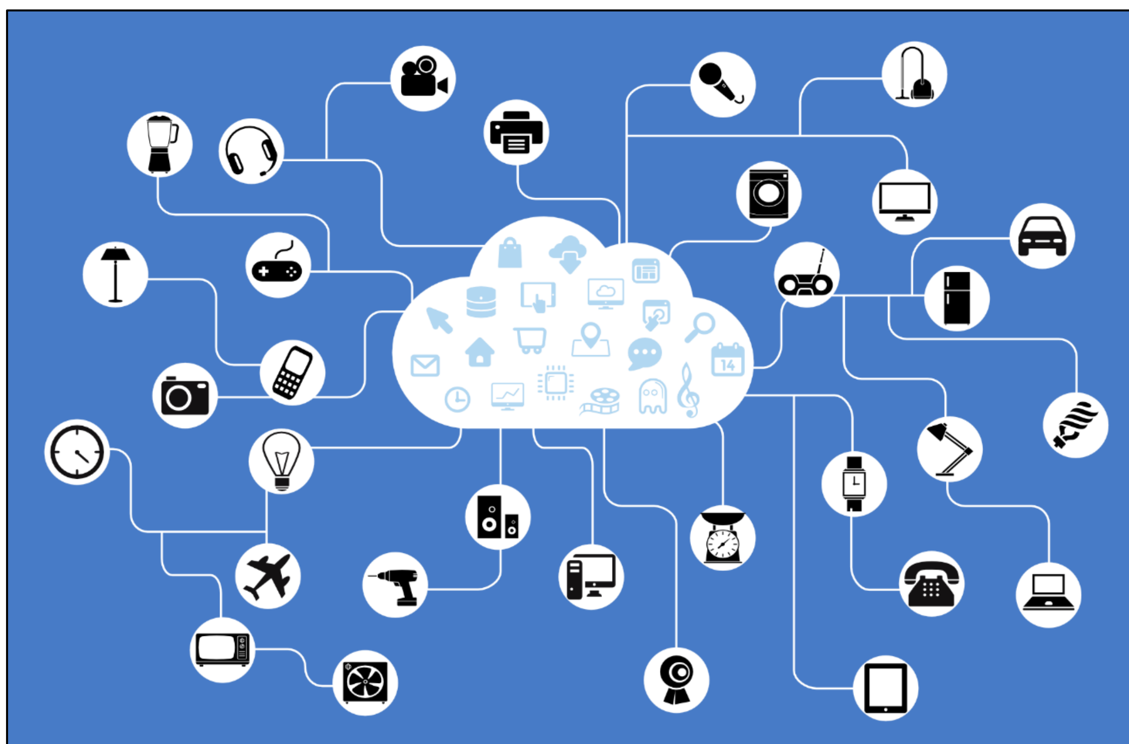

Рисунок 1.6 – Ілюстративне зображення мережі IoT пристроїв [12]

У описаних вище випадках, структури даних, з якими оперують системи, нагадують та відповідають ознакам графу, тому використання саме цього виду

структур даних і алгоритмів для їх обробки є доцільним і ефективним. Фреймворки для обробки графів є ідеальним інструментом для створення подібних систем, так як вони надають готові компоненти для впровадження власних алгоритмів та рішень роботи з даними без необхідності налаштування та створення низькорівневих методик маніпулювання даними, ретельного врахування особливостей програмної та апаратної середи роботи, тощо. Різноманітні фреймворки надають особливі можливості для роботи, вони можуть відрізнятися умовами свого застосування, переліком наданого функціоналу, методиками роботи та обробки, спроможністю працювати на апаратних архітектурах різного типу і багато чим іншим. Ці аспекти особливо важливі при плануванні використання фреймворків у високонавантажених системах з великою кількістю даних.

1.3 Постановка задачі

У ході аналізу та огляду предметної області були виявлені аспекти які можливо позначити як проблемні чи які потребують дослідження і на їх основі ми можемо сформулюємо мету дослідження. Об'єктом дослідження є фреймворки для обробки графів великого розміру, і метою дослідження є встановлення структурованої інформації про їх властивості, середи використання, методи роботи, програмні рішення, інші особливості та приклади самих фреймворків для побудови структурного розуміння сфери фреймворків обробки графів, проблем та викликів які постають перед нею і архітектурні рішення прийняті ними.

Під час дослідження заплановано розглянути різні фреймворки для обробки графів, ознайомитись з основними їх характеристиками та підходами до роботи з даними, розглянути особливості використання та можливі обмеження чи недоліки пов'язані з цим, супутні програмні рішення, а також можливі приклади їх практичного використання у відомих системах.

Впродовж дослідження будуть розглянуті наступні теми:

– представлення графів як абстрактної структури, їх властивості і вигляд;

– основні проблеми та виклики, що постають перед фреймворками для обробки графів;

– процес роботи фреймворків та загальні етапи в їх обчисленнях;

– перегляд наявних алгоритмів, що використовуються для обробки графів в фреймворках, їх класифікація;

– апаратні виклики та обмеження перед фреймворками для обробки великих графів;

– системні архітектурні рішення які використовуються для забезпечення можливостей обробки великих даних, їх використання фреймворками;

– варіація та особливості стратегій/моделей обробки задач та розділення даних, що використовуються у фреймворках;

– приклади існуючих фреймворків та їх опис і класифікація згідно раніше переглянутих властивостей;

– огляд супутніх програмних рішень у сфері – графові БД та СУБД;

– оцінка та аналіз характеристик прикладів графових БД;

– оцінка отриманих результатів дослідження та вироблення висновків щодо сфери програмних рішень обробки графів.

У процесі буде встановлено загальні уявлення про сферу обробки великих графів, розуміння внутрішньої роботи фреймворків і різноманіття їх архітектурних рішень, що роблять їх унікальними та конкурентоспроможними серед інших. Ці дані можуть бути використані для вирішення проблеми вибору фреймворків/БД для реалізації нових систем або для відкриття подальших досліджень проблем обробки великих графів і їх рішень.

2 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ ОБРОБКИ ГРАФІВ ТА ФРЕЙМВОРКИ

2.1 Загальні представлення графу

В дискретній математиці граф є представленням структури, що складається з множини об'єктів у якій ці об'єкти можуть бути між собою пов'язаними певними зв'язками. В абстракції ці об'єкти являються «вершинами» і їх зв'язки між собою – «ребра». Також можна зустріти інші назви цих елементів: вершини можуть називатися «вузлом» чи «точкою», а ребра «зв'язком», «лінією» чи «грань». Часто ви можете побачити графічне відображення графів у формі діаграми де показані набори вершин та зв'язків між ними (ребра), що зазвичай позначаються кругами (для вершин) та лініями між ними (для зв'язків). Як сказано вище, графи є одним з об'єктів вивчення в дискретній математиці, а конкретніше ці дослідження і знання відносяться до «теорії графів» [13]. Графи бувають різних типів, далі розберемо детальніше лише найпростіші з них.

Неорієнтований простий граф – граф, що складається з вершин та звичайних ребер (див. рис. 2.1). Представлений у вигляді:

$$G = (V, E),$$

де V – це набір вершин (вузлів, точок), $E \subseteq \{\{x, y\} \mid x, y \in V \text{ та } x \neq y\}$ – набір ребер, що представляють неорієнтовані пари вершин (зв'язки між двома різними вершинами).

У цьому позначенні $\{x, y\}$ є ребром, що поєднує дві вершини x та y, іншими словами показує інцидентність (пов'язаність) x та y. Також у графі вершина може не мати зв'язків з іншими вершинами, тобто не бути у складі ребра. За визначенням даного графа, для двох вершин не може бути більше одного ребра.

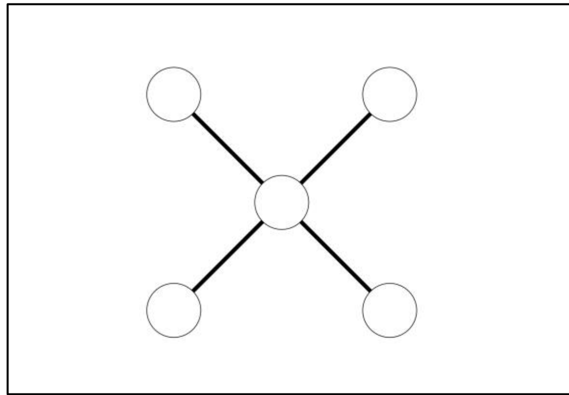

Рисунок 2.1 – Приклад неорієнтованого простого графу [Рисунок виконаний самостійно*]

Неорієнтований мультиграф – граф, що складається з вершин та звичайних ребер і може містити у себе більше одного ребра для двох тих самих вершин (див. рис. 2.2). Позначається він наступним чином:

$$G = (V, E, \phi),$$

де V – це набір вершин (вузлів, точок), E – набір ребер, $\phi: E \rightarrow \{\{x, y\} \mid x, y \in V \text{ та } x \neq y\}$ – інцидентна функція, що відносить кожне ребро до пари вершин.

Таким чином через додаткове позначення можна виразити більшу кількість ребер для пари вершин (встановити їх «паралельно»).

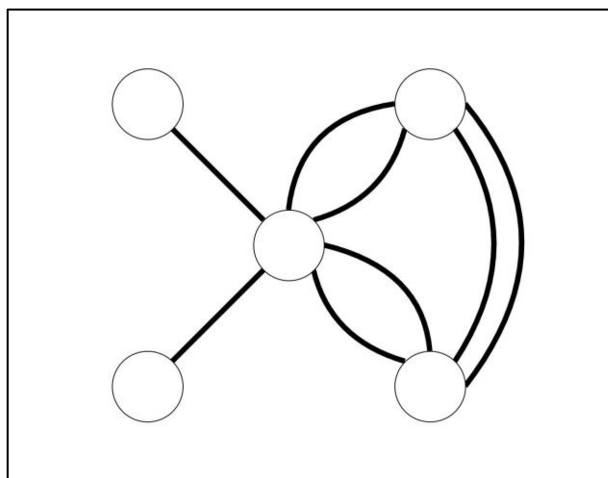

Рисунок 2.2 – Приклад неорієнтованого мультиграфу [Рисунок виконаний самостійно*]

Орієнтований граф – граф, у якому ребра мають орієнтацію (напрям). Схожий по структурі на неорієнтований простий граф, але ребра мають напрям з однієї вершини до іншої (див. рис. 2.3). Представлений у вигляді:

$$G = (V, E),$$

де V – це набір вершин (вузлів, точок), $E \subseteq \{(x, y) | (x, y) \in V^2$ та $x \neq y\}$ – набір ребер, що представляють направлений зв'язок між парою вершин (також можуть називатися направлені ребра, направлені лінії, стрілки та дуги).

V^2 – позначає набір n-кортежів елементів V, що відповідають впорядкованій послідовності з n елементів (не обов'язково різних). Грань (x, y) направлена з x до y, які ще називаються кінцевими точками ребра. x є хвостом ребра, а y є головою ребра. Ребро (y, x) є інвертованим до (x, y). Як і в неорієнтованому відповіднику, граф може мати вершину не пов'язану з жодним ребром, а також не може мати більше одного ребра, що має у себе однакові вершини у хвості і однакові вершини для голови ребра.

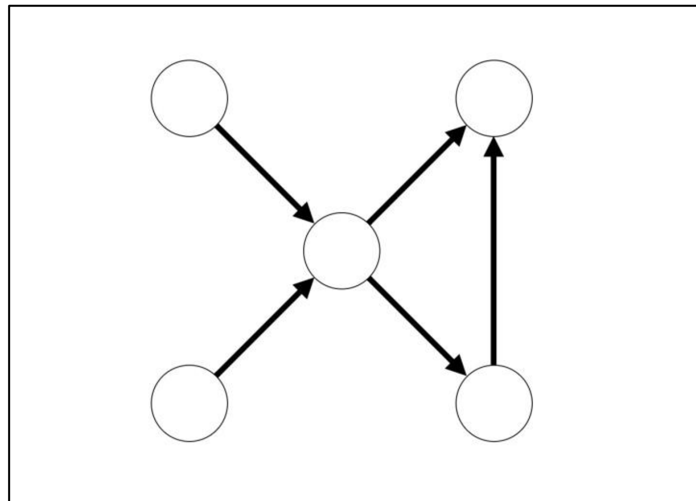

Рисунок 2.3 – Приклад орієнтованого графу [Рисунок виконаний самостійно*]

Орієнтований мультиграф є відповідником неорієнтованого мультиграфа – це граф, з направленими ребрами, що може містити 2 або більше ребра де є однакові хвости і однакові голови (паралельні ребра) (див. рис. 2.4).

Позначається він наступним чином:

$$G = (V, E, \phi),$$

де V – це набір вершин (вузлів, точок), E – набір ребер, $\phi: E \rightarrow \{(x, y) | (x, y) \in V^2$ та $x \neq y$ – інцидентна функція, що відносить кожне направлене ребро до пари вершин.

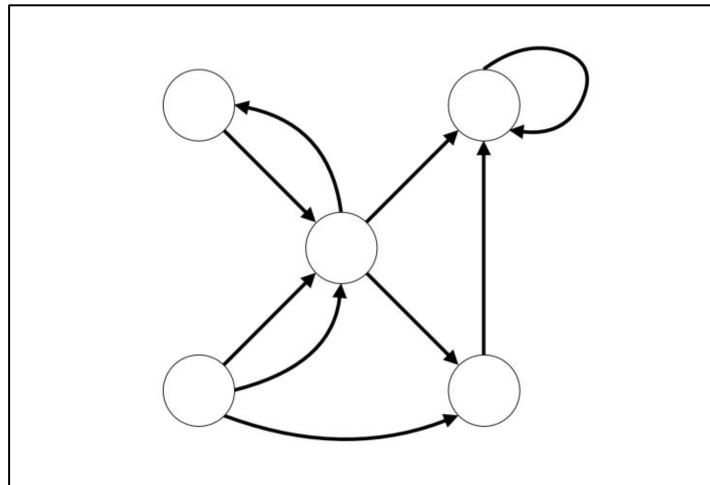

Рисунок 2.4 – Приклад орієнтованого мультиграфу [Рисунок виконаний самостійно*]

Розглянемо основні характеристики, що властиві графам. Кількість V та E зазвичай прийнято вважати скінченним, де V представляється як не пуста множина (якщо $V = 0$ то граф називають порожнім), а E може бути пустою множиною. Кількість вершин графу називають порядком графу і позначають як $|V|$. У свою чергу кількість ребер графу визначає його розмір і позначається як $|E|$. Ступінь або валентність вершини – це кількість ребер у графі, що інцидентні до неї, тобто кількість ребер, що включають у себе цю вершину (цикли у цьому випадку рахуються двічі – як два зв'язки). Ступінь графу є максимальна ступінь вершини, що зустрічається у неї.

Тож виділимо загальні властивості графу – граф $G = (V, E)$ складається з набору вершин $V = \{v_1, v_2, v_3, \ldots, v_n\}$ та набору ребер $E = \{e_1, e_2, e_3, \ldots, e_n\}$, що

позначають відносини між вершинами $E = V \times V$. Вершини називають зв'язаними чи сусідніми якщо $(v_i, v_j) \in E$. Самі ребра можуть бути направленими (орієнтованими) чи ненаправленими (неорієнтованими) та містити додаткові атрибути (вагу чи позначку).

2.2 Цінність графових фреймворків та структур даних

Як було вказано вище, графи складаються з набору вершин та набору ребер, що їх з'єднують. Більшість графових фреймворків реалізують у собі саме цю структуру даних (елементи з вершин та ребер) і підтримують додаткові їх атрибути – позначки для вершин та вагу для ребер. Окрім позначок/ваги деякі фреймворки підтримують використання власних типів як атрибутів у своїй моделі даних. Таким чином у графі зберігаються самі дані та їх зв'язки між ними (схематичний приклад на рис. 2.5).

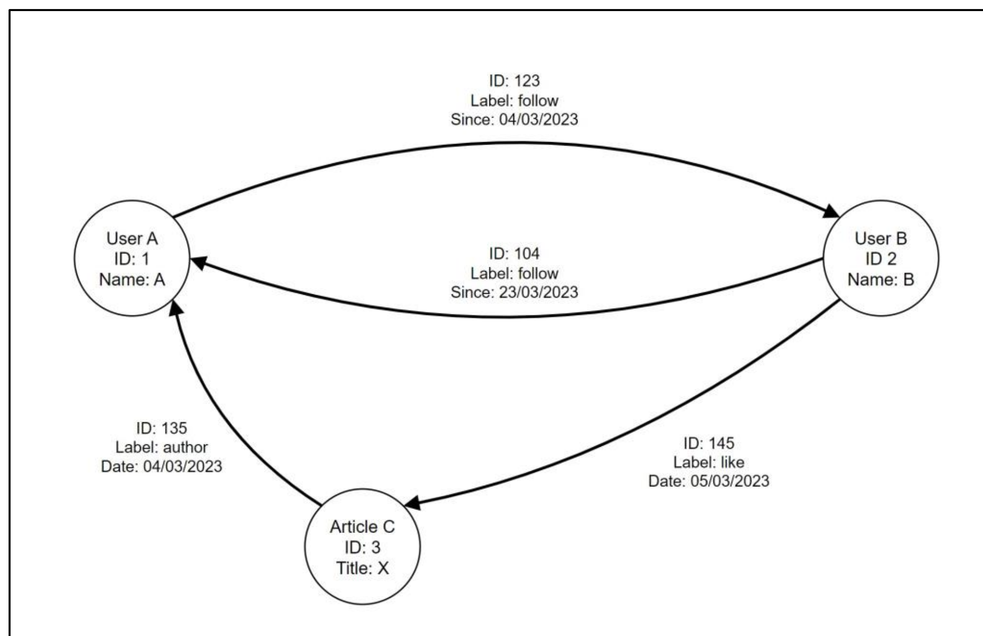

Рисунок 2.5 – Приклад збереження даних у графі [Рисунок виконаний самостійно*]

Велика кількість предметних галузей, структурованих систем та отриманих даних з реального світу природнім чином вписується у формат графових структур даних. Це є однією з причин появлення та використання їх у галузі IT технологій.

На відміну від традиційних структур даних, що використовуються у роботі програмних систем, таких як реляційні БД чи об'єктно-орієнтовні, графові структури акцентують увагу саме на зв'язкові відношення між даними, що вони зберігають. Відповідно графові фреймворки та СУБД реалізують різні типи обчислювальних алгоритмів основаних на теорії графів і властивості графів, що вирішують задачі з акцентом на зв'язки в даних, їх зближеність та відносні характеристики між ними. Таким чином, опрацьовуючи дані з урахуванням їх відносин, ряд специфічних задач може бути опрацьована більш ефективно. До прикладів таких завдань можна віднести: пошук можливих зв'язків між певними вершинами (даними); розрахунок кількості пов'язаних вершин до конкретно обраної; пошук найкоротшого шляху між вузлами мережі; відносна важливість певної вершини у графі; пошук схожих чи наближених даних, тощо.

2.3 Основні виклики для задач опрацювання графів

Обробка обчислень у графах має свої особливості через унікальність характеристик самого графа як абстракції/структури. Тому, зазвичай, конвенційні підходи до обробки даних, що застосовуються до реляційних та об'єктно-орієнтованих структур, погано працюють у випадку роботи з графами. Розглянемо основні пункти, що характеризують процес обробки графу і які можуть являтися труднощами для фреймворків обробки графів [14].

Процеси обчислень у графах мають у своїй основі дані і акцентовані на них. Графові алгоритми опираються на саму структуру даних, тобто самі обчислення керуються даними на яких вони проводяться. Так як граф це набір з вершин і ребер, то даними для опрацювання являються не тільки інформація, що міститься у вершинах та ребрах, а і сама побудова (топологія) графу. Через це шлях обчислення у графових алгоритмів змінний в залежності від змін побудови графа, на відміну від алгоритмів обробки статичних структур, таких як таблиця, де змінюються лише дані в ній.

Змінність графу може викликати різні проблеми в його обробці. Це наслідки попередньої тези – через те, що обробка графів залежить від самих даних, то на

різних графах можуть виникати різні проблеми. Гнучкість графу, зміна кількості вершин та ребер, зміни в топології і асиметричність графу не дозволяють точно прогнозувати роботу алгоритмів, створювати оптимальну схему розділення даних та викликають проблеми у оптимізації паралельних обчислень на графах.

Також через нерегулярність характеристик графу наявні методи оптимізації обчислень, що використовуються у більш низьких рівнях роботи програми (оптимізація на рівні процесора, компілятора/інтерпретатора, тощо) можуть не працювати, тому може бути складно досягти високої ефективності для алгоритмів обробки графів.

Особливість залежності алгоритмів від даних породжує додаткову проблемну область – залежність від швидкості читання та завантаження даних. Через високе співвідношення операцій над даними до самих операцій обчислень з даними, швидкість та ефективність роботи програмних рішень дуже сильно залежить від характеристик процесів читання/запису даних. Саме завантаження даних з диску для опрацювання може займати набагато більше часу ніж саме опрацювання. Ця проблема особливо торкається ситуацій де використовуються великі об’єми даних, які ще можуть бути розділені на частини (не тільки логічно, але і фізично на різних серверах).

2.4 Загальна схема обчислень у графі

Можливо розділити процес обробки графа програмною системою на основні узагальнені кроки які вона виконує (див. рис. 2.6). Цими кроками є:

– читання даних;
– підготовка даних;
– розділення даних;
– обчислення;
– запис даних.

Алгоритми обробки графів опираються на структуру самого графу, тому процес читання даних може зайняти велику частку від усього часу виконання задачі. Тому алгоритми роботи з даними та технічна складова обчислювальної машини

грають велику роль у роботі графових фреймворків. Також окрім читання, задачі запису даних теж опираються на цей фактор – результатом обчислень може бути не тільки звіт про аналіз чи пошук, але і змінений граф, що потрібно буде записати назад до сховища даних.

Рисунок 2.6 – Загальні етапи процесу обробки графу [Рисунок виконаний самостійно*]

Підготовка даних є звичним етапом роботи різноманітних систем, що працюють з даними. На цій фазі можуть відбуватися різні процеси, що готують дані до подальшого опрацювання, це можуть бути фільтрація, розділення їх на окремі частини, перетворення у необхідний для опрацювання вигляд, тощо.

Розділення даних це можлива фаза в якій виконується розділення отриманих даних для подальшого його почергового чи паралельного обчислення. Виділений як окремий етап, через можливість того, що дані будуть поділені при читанні та подачі їх до оброблюючого процесу (наприклад подача даних частинами зі сховища даних), і також будуть поділені усередині процесу перед самим його опрацюванням (наприклад поділення для паралельного обчислення).

Обчислення – власне сама робота яку виконує програмна система Використовуючи алгоритми для роботи з графами, різноманітні фреймворки обробляють граф і отримують необхідний користувачу результат. Різні системи можуть використовувати різні підходи та алгоритми.

2.5 Класифікація алгоритмів для роботи з графам

Існують різноманітні алгоритми для роботи з графами, частина з яких навіть не мала на меті працювати з ними, тобто не розроблені спеціально для графових структур, але були модифіковані для цього. На таблиці 2.1 можна побачити

теоретичні категорії алгоритмів для роботи з графами та деякі їх приклади згідно Dominguez-Sal [15].

Таблиця 2.1 – Класифікація алгоритмів обробки графів [15]

Category	Example
Traversal	Breadth first search (BFS) Single source shortest path (SSSP)
Graph Analysis	Diameter Density Degree distribution
Components	Connected components Bridges Triangle counting
Communities	Max-flow min-cut K-means, semi clustering
Centrality Measures	PageRank Degree centrality Betweenness centrality
Pattern Matching	Path/subgraph matching
Graph Anonymization	K-degree anonym K-neighborhood anonym
Other Operations	Structural equivalence, similarity, ranking, and other

Traversal algorithms (алгоритми обходу) – алгоритми для обходу графів через його вершини за допомогою ребер для виконання задач, таких як отримання інформації, чи для оновлення значень вершини/ребра. Breadth first search (BFS) (пошук у ширину, див. рис. 2.7) та Depth first search (DFS) (пошук у глибину) є класичними алгоритмами обходу графу (чи дерева) різними шляхами для пошуку заданої вершини [16]. По суті ці алгоритми задають порядок обробки вершин у графі, а під час проходження можливо виконувати інші дії над вершинами, окрім самого пошуку. Single source shortest path (SSSP) (пошук найкоротшого шляху) – це ціла категорія алгоритмів для знаходження найкоротшого шляху між двома заданими вершинами. Найвідомішими з них є алгоритми Дейкстри (Dijkstra's algorithm) та Беллмана-Форда (Bellman–Ford algorithm).

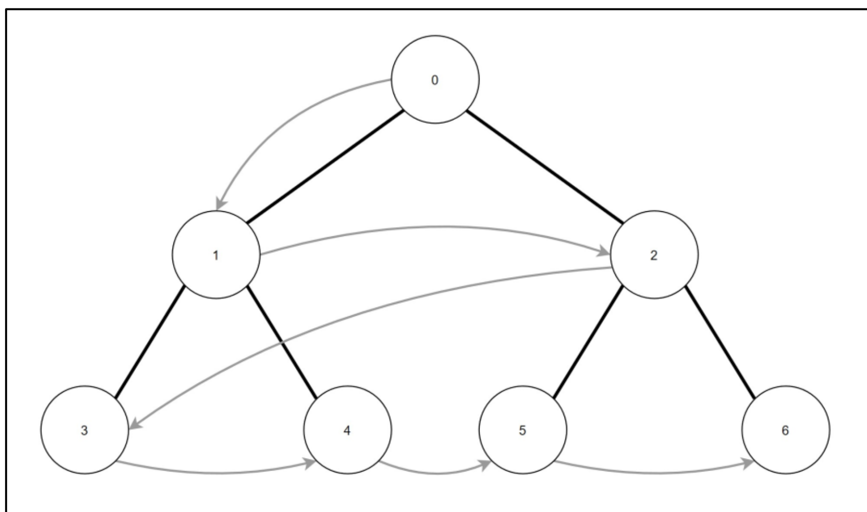

Рисунок 2.7 – Приклад схеми обходу за Breadth first search алгоритмом [Рисунок виконаний самостійно*]

Graph Analysis algorithms (аналітичні алгоритми) – алгоритми, що використовуються для визначення характеристик графа, його топології та аналізу його об'єктів. Завдяки алгоритмам з цієї категорії можливо отримувати інформацію, що випливають із структури графу. Наприклад пошук діаметру графу, його щільності, розподіл ступенів в ньому чи у підграфі. І на основі отриманих даних робити висновки по певним його об'єктам, наприклад схожість/ближність чи інші характеристики (залежить від специфіки даних, що розглядаються).

Components (компоненти) – алгоритми в цій категорії мають на меті пошук пов'язаних компонент в графі, тобто пошук підграфів, що не мають зв'язку з іншими підграфами (кожен компонент складається з вершин, що мають між собою шляхи але жодна з якої не має зв'язку з іншими компонентами, див. рис. 2.8). Окрім пошуку підграфів до цієї категорії також відноситься задача пошуку «трикутників» – пошук наборів з трьох вершин, де кожна вершина має зв'язок з двома іншими.

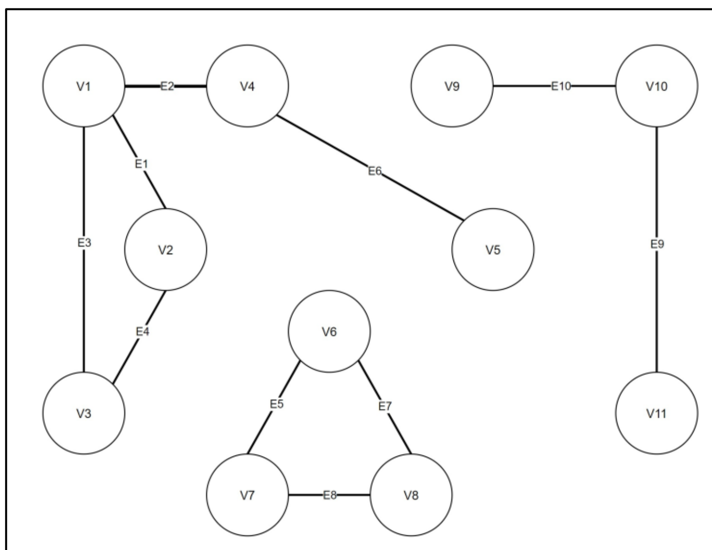

Рисунок 2.8 – Приклад компонентів графу [Рисунок виконаний самостійно*]

Communities (спільноти) – спільнота це набір вершин які знаходяться відносно близько один до одного від інших вершин/спільнот (див. рис. 2.9). Для визначення спільнот беруть в увагу різноманітні топологічні характеристики та атрибути, що використовуються у встановленні ступеню близькості і якості зв'язків. Це є варіацією кластеризації/пошуку кластерів тільки для графових структур.

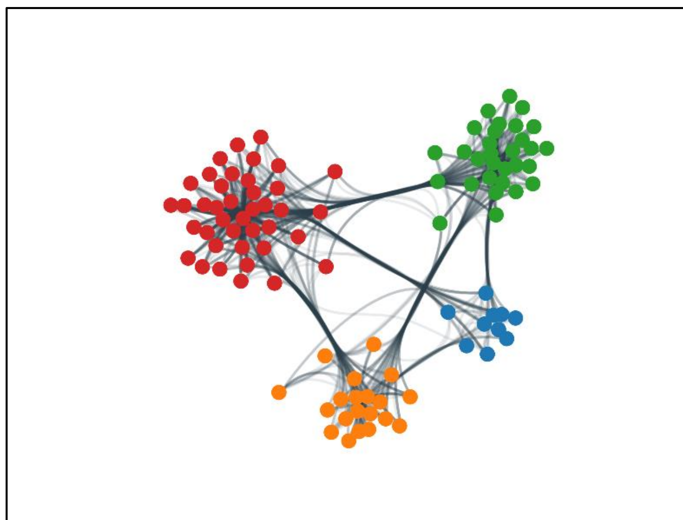

Рисунок 2.9 – Приклад виділення спільнот (communities) [17]

Centrality Measures (міра центральності) – алгоритми, що мають на меті розрахувати відносну важливість обраної вершини у спільноті/графі. Зазвичай для цього перевіряється кількість та якість зв'язків (їх поширення серед графу), що має

вершина. Найвідомішим алгоритмом цієї категорії є PageRank, що використовувався у пошуковому сервісі Google для визначення рейтингу сайту (див. рис. 2.10).

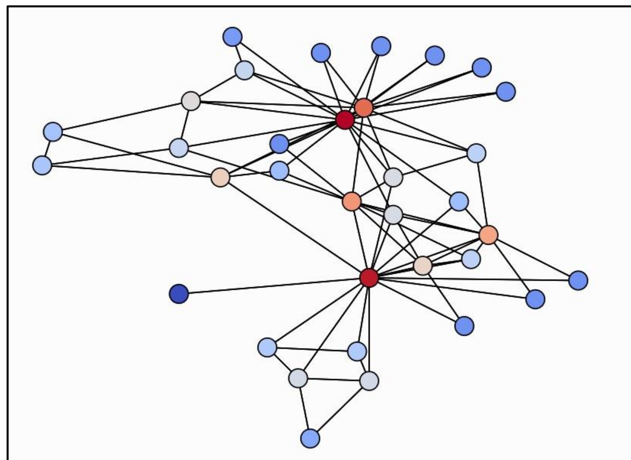

Рисунок 2.10 – Приклад візуалізації результату роботи PageRank [18]

Pattern Matching (відповідність шаблону) – категорія алгоритмів, що шукає збіги в графі по шаблону. Маючи шаблон та параметри, що задають міру схожості алгоритм шукає збіги підграфу у вхідному графі.

Graph anonymization (анонімізація графу) – категорія алгоритмів, що створюють подібну оригінальному графу структуру беручи за основу його топологічні властивості чи атрибути (див. рис. 2.11). Ці алгоритми використовують для приховування оригінальної структури даних при цьому даючи можливість працювати з його аналогом.

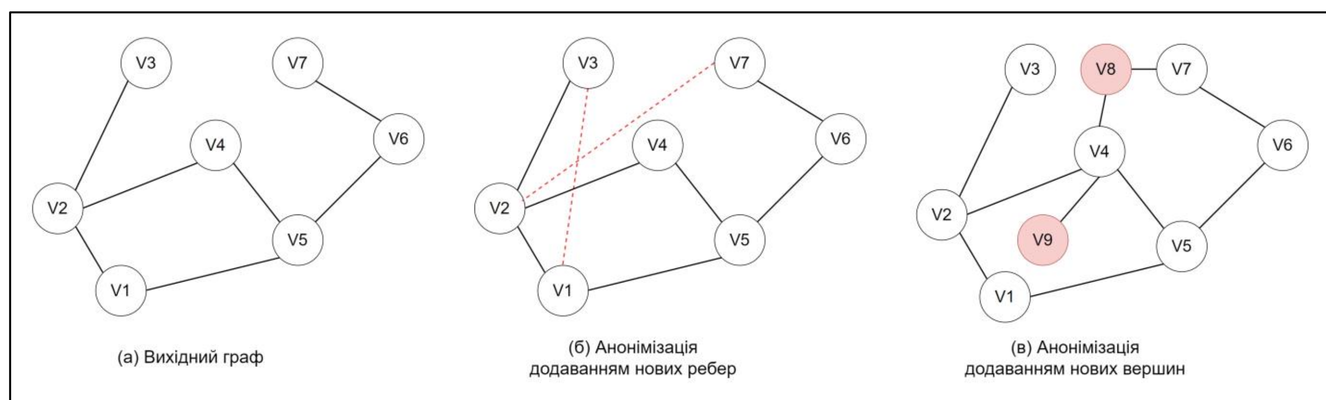

Рисунок 2.11 – Приклад анонімізації графу [Рисунок виконаний самостійно*]

До категорії інших відносяться решта специфічних алгоритмів, наприклад такі, що виконують задачі обчислення рейтингу вершин, розрахунок міри схожості вершин чи підграфів, тощо.

2.6 Основні проблеми імплементації алгоритмів у фреймворках і програмних системах

Алгоритми для роботи з графами використовуються у відповідних програмних рішеннях, але їх абстрактна схематична реалізація стикається з багатьма проблемами пов’язаними з програмними і технічними обмеженнями. Так програмні системи мають обмеження на кількість пам’яті, що може використовуватися для обробки інформації, також на об’єми інформації, що можуть бути надані до обробки, кількість одночасних виконань обчислень та інші, що не враховуються в звичайному теоретичному описі алгоритмів. Зі збільшенням об’єму даних у системах ці проблеми стають більш критичними і актуальними. Ми можемо виділити декілька основних проблем з якими стикаються при реалізації систем обробки графів чи інших систем, що працюють з великими об’ємами даних.

Об’єм оперативної пам’яті. У цьому пункті мається на увазі об’єм швидкої пам’яті яку процес може використовувати для збереження даних, що обробляються, проміжних результатів, необхідних даних для роботи програми чи алгоритму. Розмір цього виду пам’яті відносно обмежений у порівнянні з пам’яттю постійною/для зберігання. Це насамперед пов’язано з технічними особливостями пристроїв на яких виконуються обчислення. Типовий ліміт об’єму оперативної пам’яті для персонального комп’ютера на даний час сягає 128 ГБ, для серверів – більше (в залежності від архітектури материнської плати та процесора, наприклад для AMD Ryzen Threadripper PRO у парі з Asus Pro WS WRX80E-Sage SE – 2048 ГБ) [19]. Якщо обчислення проводяться не на центральному процесорі, а на графічному (GPU), то тут об’єм оперативної пам’яті досягає позначки 24 ГБ для користувацьких варіантів і більше для корпоративних/спеціалізованих. На рівні оперативної системи існують інструменти, що дозволяють використовувати постійну пам’ять для розширення доступної оперативної, але це значно впливає на

роботу системи, так як швидкість постійної пам’яті значно менша за оперативну. Тому для обробки великої кількості даних фреймворки використовують різноманітні шляхи їх розподілення як на рівні даних так і на рівні обчислень.

Особливості використання постійної пам’яті. Цей пункт стосується пам’яті, що використовується для зберігання самих вихідних даних. При виконанні задач обробки даних, дані в першу чергу «дістаються» зі сховища даних, технічні характеристики яких можуть бути завадою для досягнення високої швидкості обчислень. Окрім обмежень швидкості, є обмеження доступу до пам’яті – в залежності до типу операції та фізичного розміщення даних у сховищі швидкість доступу може різнитися від теоретично максимальної. Також одночасні операції над даними зменшують можливу пропускну здатність для кожної з них. Для набору великих даних додатково є проблемою їх зберігання у єдиному фізичному сховищі, тому використовуються методи розділення даних на частини, що можуть зберігатися на різних носіях – як на різних фізичних сховищах в рамці єдиної «машини», так і у кластері з множиною «машин», що мають логічний рівень спільної пам’яті.

Вище були перелічені проблеми, що пов’язані з аспектами використання пам’яті в фреймворках для виконання обчислень. Для зменшення впливу цих чинників, особливо для великих об’ємів даних, частина фреймворків використовує парадигму розподілених обчислень та методів збереження даних. Завдяки цьому алгоритми можуть працювати на даних у кластері машин зі «спільною пам’яттю», що розширює фізичні обмеження пам’яті. Самі розподілені обчислення відкривають можливості виконувати операції окремо на кожному з вузлів кластеру і над своєю частиною даних (що зберігаються безпосередньо на цьому вузлі). Чинниками проблем у цій ситуації може бути особливості алгоритму обробки графу, що потребує послідовного виконання операцій, проблеми зв’язку між вузлами кластеру машин та керування розподіленим процесом обробки.

Тож у підсумку можемо виділити основні характеристики/властивості фреймворку для роботи з графами, щоб досягти більшої ефективності при використанні в великих графах. По перше, бажано щоб програмна система мала

можливість працювати з розподіленими даними, тобто з даними, що частинами знаходяться на різних фізичних вузлах. Алгоритми роботи фреймворку повинні мати змогу працювати і розділяти роботу на частини, що виконуються окремо на різних машинах (підтримка парадигми розподілених обчислень) над частинами даних. Підтримка паралелізму в алгоритмах також сприятиме більш ефективній роботі системи.

2.7 Категорії архітектурних моделей обробки

Процес обробки даних у різних видах його здійснення, можливо поділити на категорії (див. рис. 2.12), що відображають архітектурну модель виконання цього процесу:

- розподілена архітектура;
- архітектура на основі однієї машини;
- гетерогенна архітектура.

Це розділення стосується не тільки систем, що обробляють графи, а і усіх інших. В залежності від необхідних результатів та наявних засобів, одні типи архітектури можуть бути більш доцільними ніж інші.

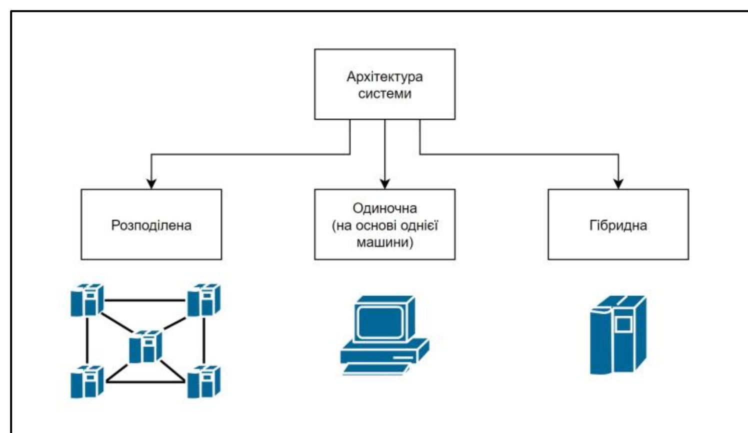

Рисунок 2.12 – Ілюстративна схема типів архітектурних систем [Рисунок виконаний самостійно*]

Розподілена архітектура – тип архітектури яка описує розподілену систему, що включає у себе множину однотипних обчислювальних компонент (машин), що

в свою чергу мають доступ лише до своєї локальної/приватної пам’яті. Між кожною компонентою (машиною) системи мається зв’язок який дозволяє передавати інформацію між собою (наприклад деталі виконання задачі, результати виконання задачі, координація тощо). Такі системи зазвичай мають головну, керуючу машину, що відповідальна за роботу усієї мережі: керує даними, що зберігаються на вузлах, дає завдання і координує роботу вузлів, обробляє помилки та інше (архітектура майстер – робітник, master-workers architecture).

У фреймворках для роботи з графами використовують цю структуру для розподілення великої кількості даних на більш менші частини і для виконання операцій паралельно на мережі машин, що в теорії повинно прискорити та оптимізувати обробку. Усі дані системи (велика графова структура) поділяється на частини, кожна з яких може зберігатися на окремому вузлі мережі машин. Передбачено, що масштабування системи, у разі необхідності зберігання більшої кількості даних відбувається горизонтально, додаючи нові вузли до розподіленої мережі машин (але вертикальне масштабування також можливе збільшуючи розмір сховища даних і підвищуючи обчислювальні можливості для кожного вузла мережі). Така архітектура обчислень відносно складна для використання фреймворками, так як потребує врахування багатьох її аспектів. Алгоритми та програмні системи повинні бути адаптовані для використання розподіленої архітектури: керувати розділенням даних між вузлами та враховувати це при обробці, мати рішення для запобігання ситуацій «гонки» (races; при дефектах координації паралельні обчислення можуть записувати некоректні дані) та «тупиків» (deadlocks; ситуація в якій паралельні обчислення блокують роботу одне одного взаємними умовами які не можуть бути виконаними у даний час), що властиві розподіленим та паралельним системам.

Одним за прикладів методів обробки, що використовує цей тип архітектури є MapReduce [20]. MapReduce – це методологія для обробки великих об’ємів даних за допомогою використання розподіленої мережі машин і паралельних алгоритмів, розроблений і представлений Google у 2004 році. Ця модель має простий інтерфейс для розробника, що надає можливість йому задати функції Map та Reduce, які

будуть виконуватися на самих даних [21]. Цей запит за зазначеною схемою роботи автоматично виконується у паралельному форматі. Розподілення даних, групування, координація та планування задач для обробки на вузлах повинна брати на себе система та інші підсистеми розподіленої мережі. Але в плані використання цієї методології для виконання задач обробки графових структур, вона має суттєві недоліки. За своєю логікою MapReduce має лише дві фази у моделі обробки (Map та Reduce відповідно), коли більшість алгоритмів, що працюють з графами, використовують ітерації для проходження графу. Також ці фази не мають пам’яті, що зберігають стан даних які вони обробляють – через це їх використання в графових алгоритмах (і не тільки) приводить до частих звернень до пам’яті – які в свою чергу сповільнюють обробку. MapReduce має низьку ефективність в задачах обробки графів через час операцій читання/запису, а також затримки при перевірках координації та результатів завершених задач, передачі даних по мережі розподілених машин. Більш розвинутим аналогом цього фреймворку, що також використовує підхід MapReduce є Apache Hadoop (Apache Software Foundation 2011) [22]. Apache Hadoop є популярним фреймворком з відкритим кодом (open-source), що підтримує NoSQL платформи обробки запитів, пакетні обробки і є більш гнучким у виконанні самих запитів. Частиною Hadoop є механізм розподіленої файлової системи (HDFS) і сам механізм MapReduce для Hadoop, що є популярним серед розподілених систем і є основою для багатьох інших фреймворків та платформ, включаючи в себе фреймворки обробки графів.

Архітектура на основі однієї машини – тип архітектури яка описує систему, що складається з одного обчислювального вузла – комп’ютера/сервера. Вузол може мати один або декілька процесорів та має загальну оперативну пам’ять для них, тобто усі обчислювальні ядра мають доступ до однієї фізичної пам’яті. Цей тип архітектури також називають архітектурою зі спільною пам’яттю (у випадку використання двох або більше процесорів). Цей тип більш відомий та знайомий, так як у цю класифікацію входять користувацькі персональні комп’ютери, ноутбуки, смартфони і інші пристрої, тому ми можемо назвати її «класичною» архітектурою серед цієї класифікації систем. Зазвичай цю архітектуру розглядають для обробки

відносно невеликих графів, бо на відміну від неї розподілена архітектура надає більше можливостей для масштабування процесів, паралельної обробки, спроможності зберігати великі об'єми даних. Основними недоліками використання однієї машини для фреймворків обробки графів є технічні обмеження, що накладає такий тип архітектури. Один вузол має фізичний ліміт розширення операційної пам'яті та кількості обробляючих ядер/процесорів. Це ускладнює обробку великих графів у випадках коли потрібно оперувати усією або великою її частиною одночасно. Також ці ліміти сповільнюють процес запису/читання даних (які використовуються не тільки у процесах обробки, але і у актуалізації/оновленні наявних даних), так як цим займається лише один операційний вузол. Але обробка на одній машині також має свої переваги, головними з яких є простота системи та відсутність затримок обробки через процеси передачі даних між вузлами та координації. Простота конфігурації спрощує написання програм та систем, що в свою чергу спрощує використання різних алгоритмів обробки (в яких вже не потрібно враховувати властивості роботи в розподілених системах). Привабливість такого підходу також підвищує розвиток технологій – процесори стають більш продуктивними і ефективними, а пам'ять стає більш швидкою, дешевою і об'ємною.

Рішення використання певних фреймворків та типів архітектури систем ґрунтується на особливостях використання та вимог до неї, і вибір розподіленої архітектури у випадку обробки великих даних не є завжди оптимальним варіантом. Наприклад дослідження Microsoft Research показало, що використання кластерних систем для аналізу великих даних не є оптимальним для багатьох задач – єдина машина з великим об'ємом пам'яті показала себе більш ефективною ніж кластер [23]. Для опрацювання об'ємів даних, що вимірюються у гігабайтах, така конструкція забезпечує кращу продуктивність за одиницю валюти (витраченої на роботу системи).

Одною з найвідоміших систем, що була створена з урахуванням роботи в такій конфігурації на одному комп'ютері є Cassovary – система з відкритим кодом для обробки графів, що знаходяться у пам'яті вузла, розроблена Twitter [24]. Ця

система використовувалась для роботи WTF (who to follow), рекомендаційного алгоритму, що пропонував сторінки у Twitter які були б цікаві користувачам.

Гетерогенна архітектура – тип архітектури, що описує систему зв’язаних обробляючих вузлів, що не є однаково потужними. Цей тип схожий за структурою з розподіленими системами, але на відміну від неї, кожен обробляючий вузол не є конфігураційно (технічно/програмно) однаковим. Також до цього класу може входити і одна обробляюча машина – комп’ютер, що має у своїй базі множину блоків обробки/прискорювачів, на яких і відбуваються розрахунки. Прикладами таких машин можуть бути апаратні системи з множиною блоків GPU, чи спеціалізованих прискорювачів обчислень (такі як карти серії NVIDIA A, H, V чи гібридні системи NVIDIA DGX, HGX) [25]. У випадку конфігурації гетерогенної структури у вигляді розподіленої системи, виникають питання до сумісності вузлів між собою (наприклад у протоколах передачі даних чи їх індивідуальних обмеженнях) а також до роботи самої програмної системи у різних середовищах (вузлах системи), але кардинально принципи роботи від звичайних гомогенних розподілених систем не відрізняються. Гетерогенні структури на основі однієї обчислювальної машини мають високий потенціал для використання у різних розрахункових задачах, у тому числі і в роботі з графами. Цей підхід дозволяє скомбінувати попередньо розглянуті архітектури – розподілену та на основі однієї машини, та в певній мірі позбавитись недоліків один одного. Усі операції виконуються на одній машині, що полегшує завдання розробки алгоритму обробки для розробників, зменшує час виконання завдань передачі даних між процесами, зменшує час на читання/запис даних. При цьому маючи множину обробляючих блоків можливо прискорювати обробку даних виконуючи її паралельно на цих блоках.

Використання GPU у фреймворках стало поширеним методом у обробці даних і не тільки, цей елемент використовується у різнорідних задачах, що колись виконувались тільки на центральних процесорах. Завдяки розвитку GPU карт цей компонент став багатофункціональним елементом сучасного комп’ютеру, що виконує не тільки задачі пов’язані з графікою, але і з математичними обчисленнями,

моделюванням, задачами машинного навчання, штучного інтелекту, тощо [26]. Тому системи обробки графів також не оминають можливість використання їх для спрощення та прискорення власних обчислень. Деякі фреймворки були розроблені для використання GPU як обробляючого центру, наприклад Medusa чи TOTEM.

Але у гетерогенних/гібридних системах на основі однієї машини, попри комбінуванні переваг, все ще залишаються притаманні їй недоліки: обмеженість масштабування можливостей обробки та доступної пам’яті, обмеженість фізичного розташування, обмежена пропускна здібність як для запитів обробки так і для задач читання/запису даних у сховищі даних.

2.8 Координація у розподілених систем

У розподілених системах, а також багатопоточних, координація обчислень є критично важливою частиною роботи системи. Керування задачами, що виконує кожен оброблювальний вузол, синхронізація та передача даних між ними, збір результатів та остаточне оформлення виконаної мети – є одними з частин, що вкладається у визначенні координації в таких системах, і фреймворки для обробки графів не є виключенням. При використанні розподілених систем для обробки графів, стратегії координації виконання запиту/задачі не мають кардинальних відмінностей від стратегій паралельної обробки даних. Їх можна розділити на 3 групи:

– синхронний тип;
– асинхронний тип;
– гібридний тип.

Синхронний тип описує системи, що виконують роботу на множині обробляючих вузлів синхронно – кожен з вузлів виконує свою частку роботи паралельно, але одночасно розподіл нових задач/даних, збір результатів виконується в рамках ітерації обробки для всіх вузлів (див. рис. 2.13) [27]. Послідовність таких ітерацій є чітко визначеною і координується на усій мережі розподіленої системи.

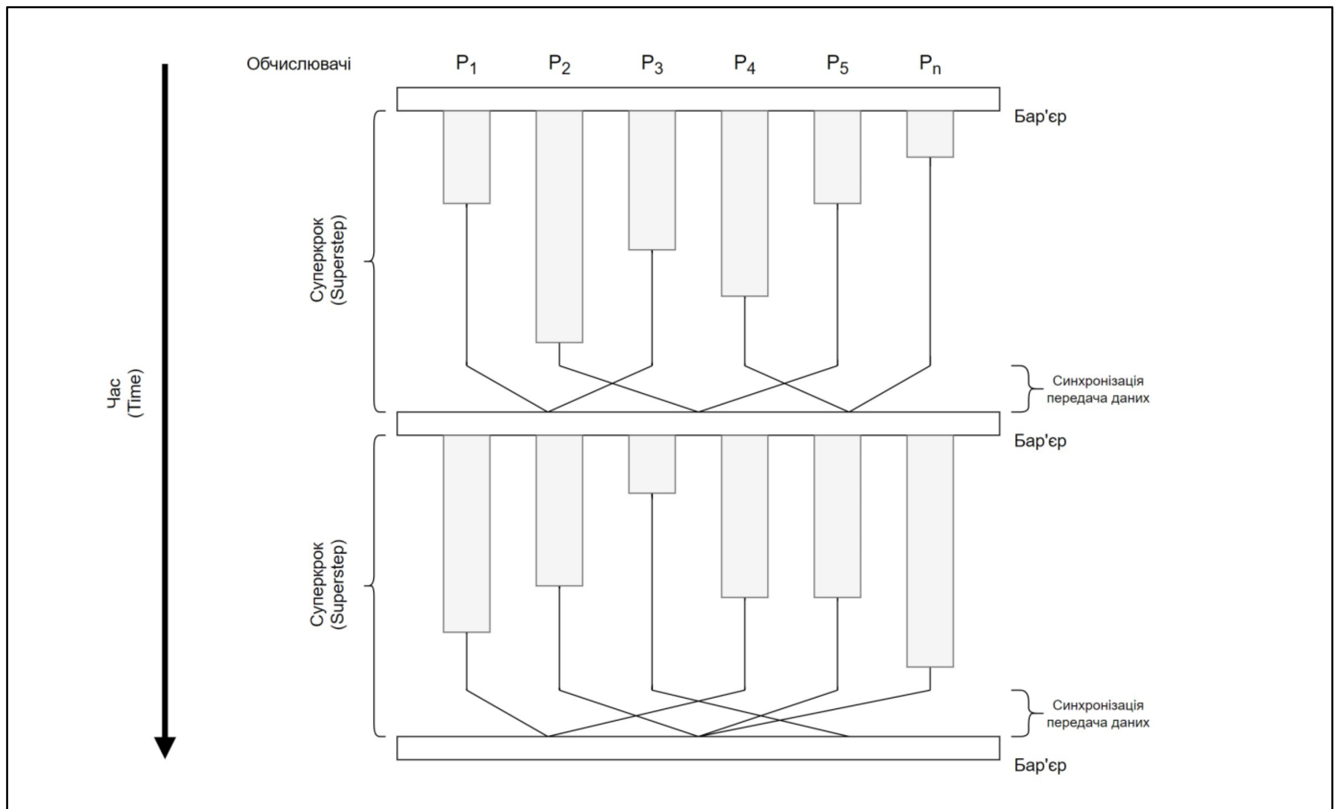

Рисунок 2.13 – Приклад схеми роботи за синхронною моделлю BSP (Bulk Synchronous Parallel) [Рисунок виконаний самостійно*]

Таким чином система має управління над кроками обробки, що надає змогу виконувати алгоритм послідовно крок за кроком, не дивлячись на те, що вони розподілені між вузлами і задачі всередині кроків (ітерацій) виконуються паралельно. Така стратегія обробки можлива для використання не тільки в розподілених системах але і в звичайних, на основі одного комп’ютера чи з гібридною архітектурою, при цьому не потребує значних змін у логіки обробки (паралельні задачі в рамках кроку можуть бути виконані на паралельних потоках чи на апаратних прискорювачах в гібридних системах чи взагалі послідовно). Одними з прикладів використання такого типу координації є системи Pregel та йому подібні/похідні (Apache Giraph, Apache Spark GraphX, і т.п.), GPS, Medusa. В них використовуються засоби координації паралельних обчислень, один з яких називають бар’єр. Він використовується як синхронізуюча точка, що агрегує результати паралельної обробки в рамках однієї ітерації загального виконання задачі. Кожна така ітерація називається суперкрок і містить у собі підзадачі, що

виконуються паралельно на окремих вузлах. Бар'єр контролює виконання суперкроку – усі вузли повинні завершити опрацювання суперкроку перед тим як синхронізувати між собою данні, отримати нові вхідні дані і приступити до нового суперкроку. Абстрактна схема роботи цього методу виглядає так: головний вузол розділяє та розподіляє дані між вузлами і надає задачу для обробки; кожен вузол працює з отриманими даними і по завершенню надає сигнал бар'єру і чекає відповіді від нього; коли усі вузли завершили обробку своєї частини, бар'єр дозволяє продовжити обробку; усі отримані результати суперкроку агрегуються, вузли отримують вхідні дані для подальшого суперкроку і починається наступна ітерація. Самі задачі всередині суперкроку вже залежать від алгоритму, що виконується.

Такий спосіб керування паралельною обробкою надає різноманітні можливості, що з'явилися завдяки особливостям цієї стратегії. По перше, через ітераційність роботи системи у проміжках між виконанням суперкроків можливо виконувати додаткові обчислення, якщо алгоритм того потребує. Також це розділення задач дає вікно для оптимізації, наприклад можливо передавати дані цілими групами, а не поодинці (це впливає не тільки на можливу швидкість передачі даних, але і на саме навантаження на систему). Через покрокове виконання обчислень, можливо отримувати/моніторити стан роботи в режимі реального часу, так як усі вузли виконують лише свою частку роботи за ітерацію і між ними є час синхронізації результатів і подальших інструкцій. Модель синхронного виконання задач також має перевагу в можливостях балансування навантаження на вузли системи. При оптимальному розподілені даних між вузлами можна добитись меншого часу виконання суперкроку. Давши більше даних для обробки до більш потужного/вільного вузла системи ми можемо скоротити час як виконання завдання так і очікування сигналу від бар'єру (коли більш потужний вузол чекає на завершення обробки інших вузлів). Перевагами синхронної обробки також є відносна простота розробки алгоритмів під неї, налагодження таких систем і низька схильність до проблем умови перегонів (race conditions) та тупиків у обробці (deadlocks).

Синхронний тип обробки має свої недоліки та моменти, що необхідно враховувати. Можливості балансування навантаження можуть навпаки зменшити ефективність роботи системи, якщо це балансування було неефективно відтворене і навантаження на вузли виявилось асиметричним і не відповідним їх потужності. У такому випадку час суперкроку буде залежати від найповільнішого вузла – він буде змушувати інших чекати на завершення його кроку на бар'єрі. Час синхронізації між суперкроками і час очікування на єтапі бар'єру не використовує обчислювальні можливості вузлів на повну, тому можна сказати, що така система може бути неефективною, бо має багато проміжків часу коли вузли очікують команд (простоюють).

Асинхронний тип описує модель обробки, що не використовує бар'єри і проміжні етапи обробки для синхронізації даних між вузлами. Вузол може виконувати наступні кроки відразу після завершення попереднього не чекаючи на інші вузли, тобто у цій моделі відсутні суперкроки як це передбачено у синхронній моделі (див. рис. 2.14). Таким чином ця стратегія роботи не має найбільшого недоліку попередньої моделі – неефективність роботи при неправильному балансуванні навантаження. Швидкість обробки не упирається у найповільніший вузол системи, вільні вузли не мають чекати на нього і можуть брати у обробку наступні пакети даних. Через відсутність синхронізуючих етапів балансування навантаження може бути більш динамічним – при синхронній обробці балансування виконується на етапі розділення задач між ітераціями, коли с асинхронному варіанті це можна робити відразу як тільки вузол звільняється. Отже, у такий спосіб асинхронний підхід пропонує використання неблокуючих моделей обробки, де кожен вузол мережі, де відбувається обчислення, бере на себе обробку даних відразу як тільки звільняється від попереднього, що підвищує саму ефективність роботи системи (через зменшення часу простою).

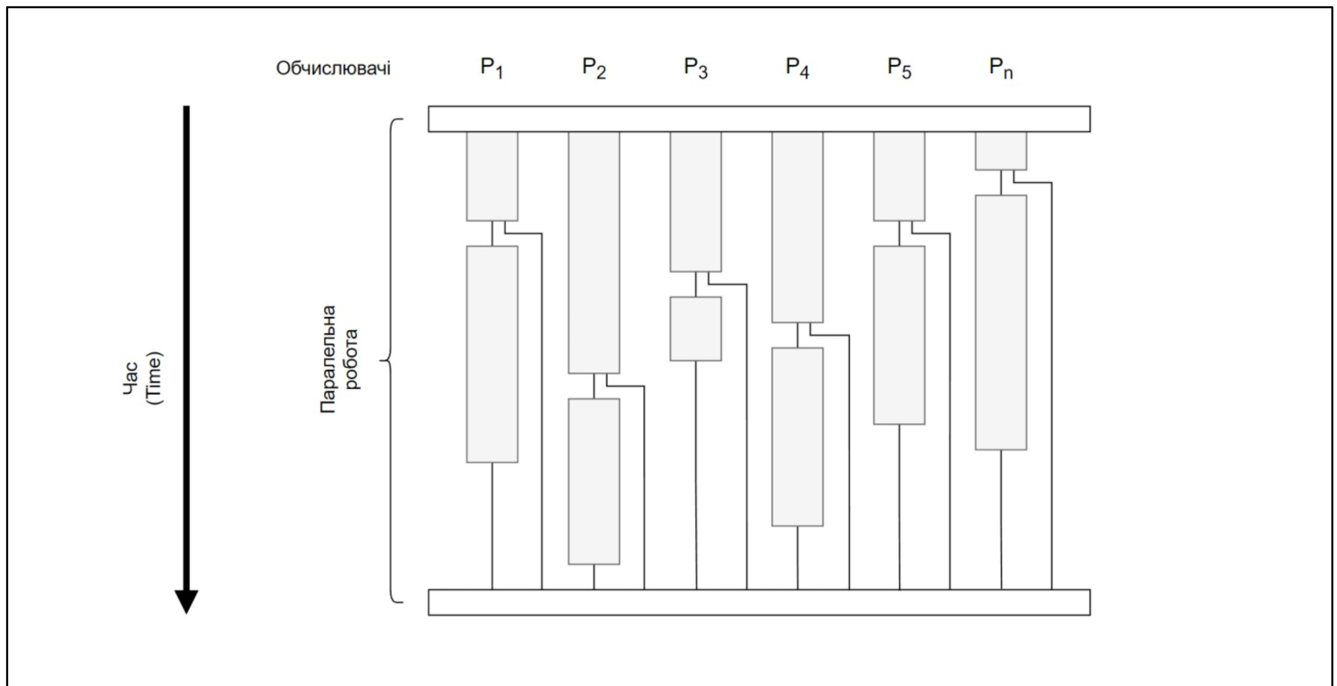

Рисунок 2.14 – Приклад схеми роботи за асинхронною моделлю [Рисунок виконаний самостійно*]

Але такий підхід приносить свої недоліки, найголовнішим з яких є складність його використання. Такі системи мають складну логіку розподілу задач, яка має враховувати множину нюансів, таких як розділення даних, нерегулярні етапи синхронізації даних, непередбачуваний час виконання задач, складність планування обробки, можливі стани гонки (race conditions) чи тупики (deadlocks) у обробці. Саме тому вони складні у розробці, налагоджені і розвертанні в обчислювальній мережі. Але не дивлячись на всю складність системи і її недоліки, переваги асинхронної моделі в фреймворках для обробки графів є значними – ефективність роботи системи, можливе прискорення роботи у порівнянні з синхронним методом, можливість отримання проміжних результатів обробки (наприклад окремо для вузлів до моменту агрегації усіх результатів). Прикладами фреймворків, що використовують асинхронну модель обробки є GiraphX, RASP, GraphHP.

Гібридний тип включає в себе підходи які комбінують частини попередніх видів у різноманітних варіаціях. Використовуючи підходи з різних моделей обробки гібридна модель намагається отримати можливості і переваги обох попередніх,

синхронних і асинхронних, та зменшити вплив їх недоліків. Одним з прикладів такого підходу є фреймворк Giraph++. Він використовує асинхронний підхід для обробки локальних даних чи наданих компонент (графу), що знаходяться на обробляючому вузлі мережі, і синхронний підхід для координації та агрегації результатів у масштабі усієї мережі (на етапі агрегування результатів та координації роботи між вузлами). Цей підхід дозволяє прискорити обробку на вузлі і при цьому мати синхронні етапи обробки, що підходить для мереж/систем які зберігають дані частинами розподілено (на різних фізичних вузлах). Іншим прикладом гібридного підходу є GRACE. Це фреймворк розроблений для роботи на одному вузлі (архітектура на основі однієї машини) і використовує комбінований підхід для обробки – асинхронно оброблюючи дані на різних потоках в рамках синхронного суперкроку. Під час суперкроку він динамічно розподіляє дані і задачі між наявними обробниками та використовує BSP (bulk synchronous parallel) модель паралельної обробки, що відноситься до синхронних типів. Також цікавим гібридним підходом є система PowerSwitch, що використовує обидва окремі синхронний та асинхронний підходи перемикаючись між ними під час роботи в залежності від поточної задачі. Така схема роботи нагадує branch predictor, що використовується у комп’ютерній архітектурі для прискорення обчислень. В залежності від поточної задачі, об’єму наступних запитів чи на основі інформації про наступний суперкрок, система здатна перемикатись між синхронним та асинхронним виконанням задачі.

2.9 Підходи до розподілу/обробки даних графу в фреймворках

Графові фреймворки опрацьовують графи (дані) на різноманітних інфраструктурах, таких як розподілені мережі (кластери комп’ютерів), на одному комп’ютері, в хмарних середовищах тощо. Через множину обмежень по пам’яті, потужності, топології системи, розміру даних і т.п., зазвичай фреймворки не мають можливості оброблювати увесь граф відразу, тому для цього вони використовують різні підходи для розділення даних перед обробкою і для самого їх обчислення. Виділяють основні абстракції за якими фреймворки зберігають та розділяють дані,

за якими встановлюються принципи обробки графу і як побудована модель представлення топології графу.

Вершино-центричні фреймворки. Вершино-центричний концепт є одним з найрозповсюджених серед графових фреймворків, у таких системах основний акцент зроблено на вершинах (дані у вершині) графу, тому дані обробляються і розподіляються на основі розташуванні вершин, що входять до графу. Дані розподіляються по окремих частинах (partitions) у пам'яті у вигляді цілих вершин, не зважаючи на зв'язки між ними (ребра) (див. рис. 2.15). Тобто окрема частина даних містить множину вершин, а ребра графу можуть бути як всередині однієї частини, так і бути загальними для різних частин (якщо зв'язані вершини опинились у різних частинах).

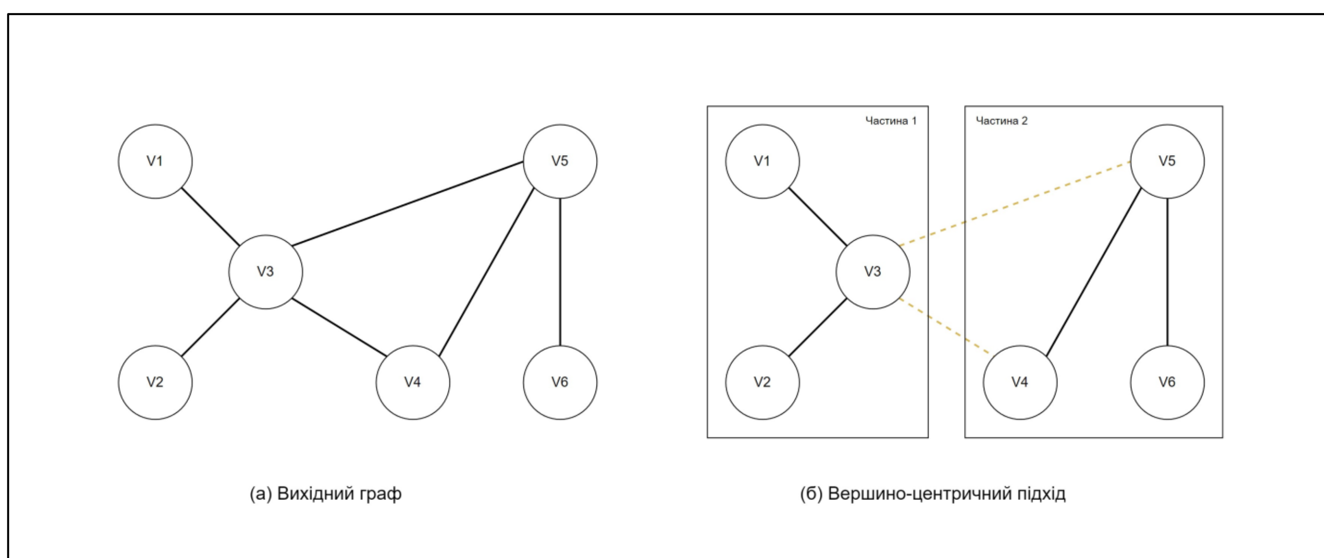

Рисунок 2.15 – Схематична ілюстрація вершино-центричного підходу до розділення [Рисунок виконаний самостійно*]

Ребра, що пов'язують дві частини даних можуть зберігатись як двома об'єктами окремо (тоді вони дублюються для кожної з них), так і в одній з них (тоді це ребро зберігається лише в одній частині і вказує на зв'язок з вершиною, що знаходиться в іншій). Одним з прикладів фреймворків, що використовують таку абстракцію є Pregel. Логіка обчислень побудована навколо вершини (як елемента графу) та ребер, що виходять з неї – процес рухається від вершини до інших вершин

використовуючи ребра. Таким чином це можливо використовувати у паралелізації самого процесу – паралельно оброблювати окремі вершини з їх зв'язками.

Ребро-центричні фреймворки. У фреймворках, що використовують таку абстракцію, центром уваги в обчисленнях є ребро (як елемент графу). По аналогії з вершино-центричним підходом, обчислення виконуються навколо ребер, і рухаються вздовж вершин (від ребра до ребра). Розділення даних виконується на основі ребер – множини ребер розділяються по різним частинам набору даних і до них вже приєднуються вершини (див. рис. 2.16). Вершина, що відноситься до різних ребер у різних частинах даних може зберігатися як в обох наборах (дублікат), так і лише в одному (з посиланням на ребро в іншому наборі).

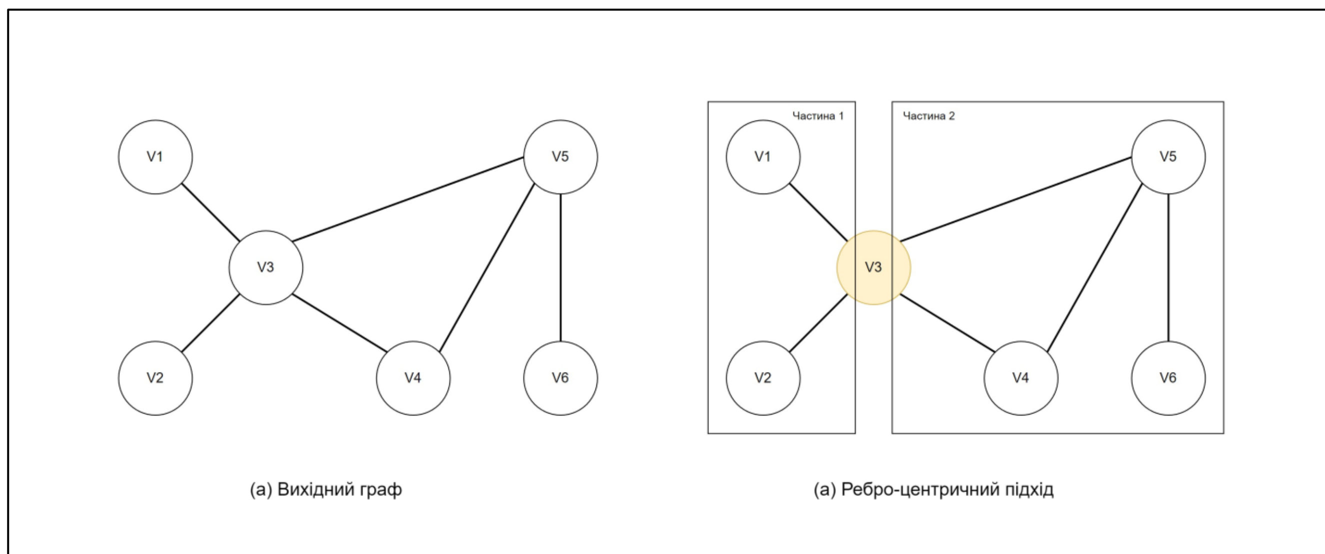

Рисунок 2.16 – Схематична ілюстрація ребро-центричного підходу розділення [Рисунок виконаний самостійно*]

Такий підхід менш інтуїтивний і більш складний в реалізації. Ефективність фреймворку, що використовує дану абстракцію сильно залежить від правильності розподілення даних – незбалансовані частки даних (partitions) можуть вплинути на швидкість паралельного виконання задач над графом. З інформаційної точки зору ребро та вершина структурно схожі, тому теоретично можливо замінити «вершину» та «ребро» між собою в абстракції, що використовує фреймворк – це один запропонованих методів використання вершино-центричних фреймворків у ролі

ребро-центричних і навпаки. Не дивлячись на те, що вершино-центрична абстракція більш інтуїтивно зрозуміла і проста, існування та використання ребро-центричного підходу є логічним і мотивованим. Графові дані у реальних випадках використання доволі часто мають незбалансовану кількість елементів вершин та ребер, де ребер може бути набагато більше ніж самих вершин. Якщо взяти у приклад соціальні мережі, то за вершину зазвичай беруть об'єкт, будь то аккаунт, група, новина, тощо. Ці вершини є сталими, але зв'язки (ребра) між нею та іншими вершинами (об'єктами) постійно оновлюються та є множиною – саме тому виходить, що граф має високу кількість ребер по відношенню до вершин. Одним з прикладів ребро-центричного фреймворку є X-Stream – створений та оптимізований під використання в архітектурі на основі однієї машини де дані розподілені у рамках одного фізичного носія даних (від цього і назва, він бере до уваги, що послідовний запис даних виконується швидше ніж рандомний/запис малих файлів). Ще одним прикладом є фреймворк Chaos, що використовується у розподілених системах з розподіленими даними. Він розділяє увесь граф на частини які зберігає на різних вузлах і використовує підхід GAS (gather-apply-scatter) для виконання обчислень [28]. Цей підхід описує абстрактну модель обчислень, що мають етапи збору інформації з різних вузлів, її обробку, «повернення» оновлених даних у розподілену систему.

Компонентно-центричні фреймворки. Компонент – є частиною графа, підграф. Цей підхід до обчислень бере в увагу цілі частини графу, а не тільки вершину чи ребро. Тому компоненти є одиницями якими оперує фреймворк, тобто до опрацювання подається відразу набір вершин/ребер (див. рис. 2.17).

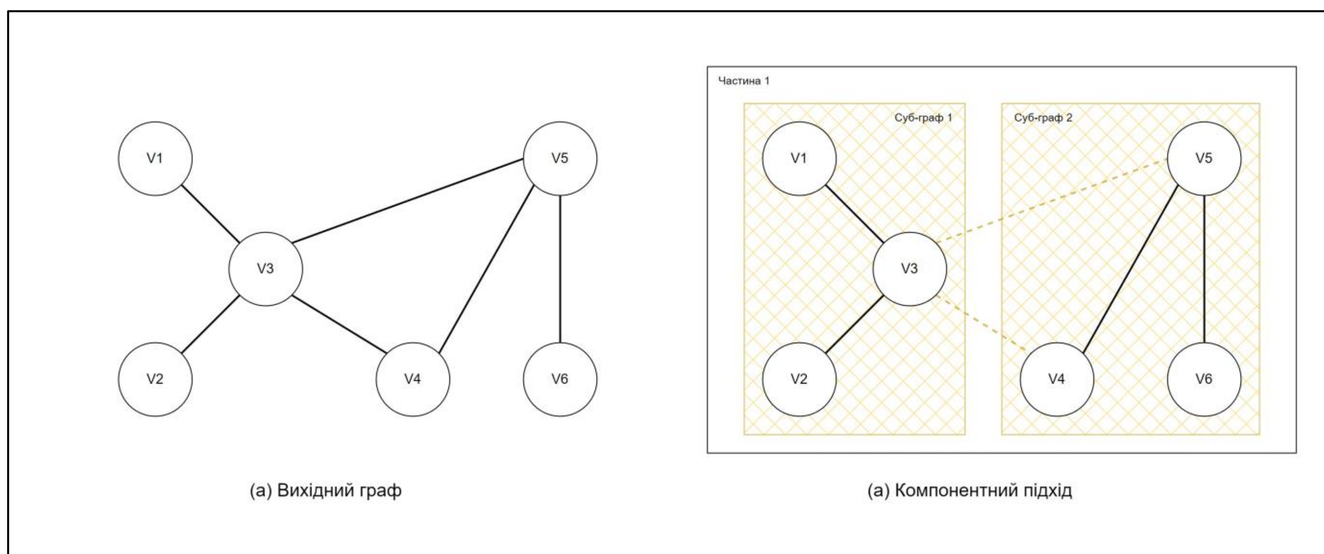

Рисунок 2.17 – Схематична ілюстрація компонентного підходу розділення [Рисунок виконаний самостійно*]

Одним з прикладів використання такого підходу є Giraph++. В ньому використовуються методи розбиття графу на компоненти (від цього залежить ефективність роботи) і обробка графу виконується саме над цими частинами. Таким чином опрацювання проходить не від вершини до вершини, чи від ребра до ребра, а цілими частинами графу – це зменшує навантаження на мережу передачі даних і теоретично зменшує час обробки в рамках ітерації (так як дані для роботи розподіляються відразу множиною, а не по одному елементу).

2.10 Фреймворки для роботи з графами

Розглянувши основні характеристики та методи роботи фреймворків перейдемо до огляду самих представників фреймворків для обробки графів, які розраховані на роботу з великим об’ємом даних. В першу чергу коротко повторно розглянемо MapReduce – програмна модель, абстракція методу обробки даних, що була розроблена компанією Google. Вона була створена як модель для паралельної обробки розподілених даних у розподілених системах (кластерах) для власних потреб компанії, але потім вийшла у світ публікацією з її описом. З тих пір це є одною з найвідоміших моделей, що використовується для обробки в розподілених системах для паралелізації обчислень, в тому числі і для обробки графів. Він сам

по собі не являється фреймворком, що призначений для обробки саме графових даних, але є основою для багатьох фреймворків та систем, що працюють з ними. Основними складовими цієї моделі є функції Map та Reduce, які виконуються в відповідних етапах роботи системи (див. рис. 2.18).

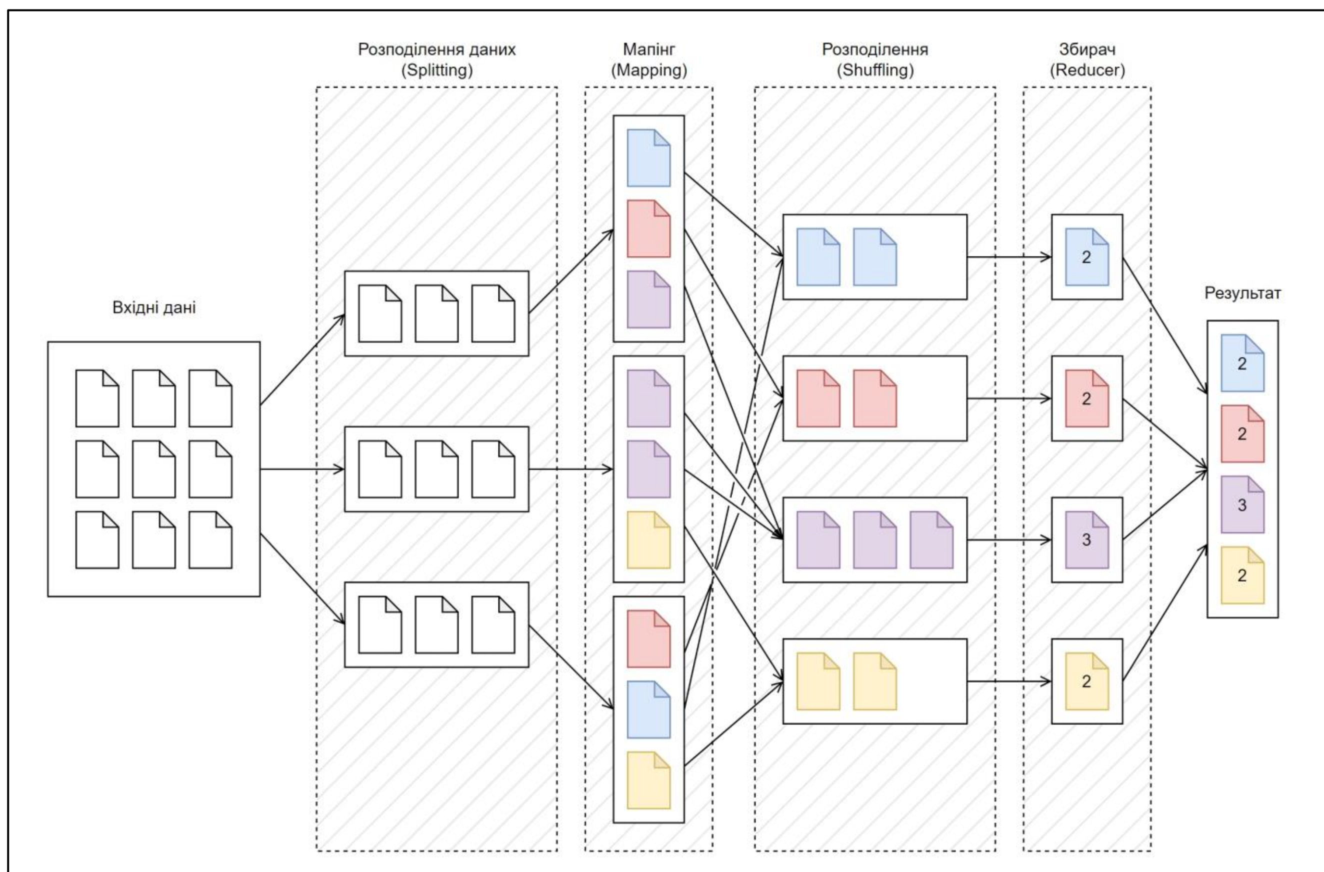

Рисунок 2.18 – Ілюстративна схема процесу роботи MapReduce [Рисунок виконаний самостійно*]

Функція Map виконується на першому етапі обробки, вона передається усім операційним вузлам з даними (у випадку роботи в розподіленій системі) та застосовується до самих даних локально на вузлах. Наступний етап, Reduce, також виконується паралельно і відповідає за агрегування даних, що були отримані на етапі Map. При агрегації відбувається обробка та з’єднання усіх отриманих результатів з попереднього кроку і отримані дані передаються на головний вузол. Це основні етапи цієї моделі обробки, але у реалізаціях також можуть бути інші додаткові, наприклад етап Shuffling, що виконується між Map та Reduce і відповідає

за збір суміжних даних в одному обробляючому вузлі (наприклад якщо дані позначені «мітками»/ключами).

Pregel – фреймворк/система розроблена в Google для завдань оброки великих графів [29]. Він розрахований на використання у розподілених системах (кластерах) для ефективної, паралельної обробки даних. Це вершино-центричний фреймворк, що використовує BSP (Bulk Synchronous Parallel) модель розподіленої обробки даних [30]. Pregel при обчисленні даних робить це послідовними ітераціями, що називаються суперкроком. Під час суперкроку система виконує користувацьку функцію для кожної вершини графу, паралельно. Функція може прочитати дані з попереднього суперкроку для поточної вершини, надіслати нові дані (повідомлення) для обробки в наступному суперкроці іншим вершинам (чи тій самій), та змінювати дані/стан поточної вершини. Зазвичай повідомлення надсилаються пов’язаним вершинам (за ребрами) але можливо надсилати їх зазначеній вершині (якщо мається її ідентифікатор). Сам фреймворк використовувався в Google для використання в розрахунку PageRank, а також став основою для інших систем обробки графових даних.

Apache Hadoop – це фреймворк/платформа, що надає інструменти та рішення для створення систем які виконують завдання з розподіленого зберігання даних та їх обробки [31]. Ця платформа є одною з тих, що використовує модель MapReduce для маніпулювання та обробки даних. Основними компонентами системи являються розподілена файлова система HDFS (Hadoop Distributed Filesystem), Hadoop YARN (платформа керування ресурсами в розподіленій мережі/кластері комп’ютерів), Hadoop MapReduce (модель обробки MapReduce для роботи в HDFS). Ця платформа являється основою для багатьох інших платформ та фреймворків, як в сімействі програмних продуктів Apache так і в інших. Запропоновані нею можливості відмінно підходять для використання у великих системах з багатьма машинами, зберігання та розподілення даних по частинам по всій мережі вузлів системи, створення програмних засобів паралельної обробки великих масивів розподілених даних. HDFS є одним з найголовніших компонентів Apache Hadoop. Ця файлова система використовується у багатьох програмних рішеннях, що

націлені на зберігання великих об’ємів даних у розподіленій системі. Якщо не розглядати деталі, то основні принципи роботи цієї системи такі: файли діляться на блоки однакового розміру (крім останнього блоку); блоки зберігаються розподілено на вузлах кластеру; блок може бути продубльований на різних вузлах; при обробці даних, пакет з інструкціями передається вузлам, де вони самостійно обробляють локальні для себе дані. Таким чином досягається відмовостійкість системи (блоки даних дублюються на різних вузлах, тому при втраті деяких вузлів в кластері дані не втрачаються) та можливість паралельної обробки даних.

Apache Giraph – це фреймворк створений для обробки графів у сфері великих даних [32]. Він був побудований з використанням платформи Apache Hadoop та є відкритим (open-source) аналогом програмної системи для обробки графів Pregel, що була розроблена компанією Google. Обидві системи використовують BSP (Bulk Synchronous Parallel) модель розподіленої обробки даних, але Giraph, окрім можливостей, що надає Pregel, також включає в себе інструменти для обчислень на головному вузлі, сегментованих агрегаторів, використання ребро-центричних вхідних даних, обчислень поза ядер (наприклад дискові обчислення), тощо. Одним з прикладів використання цього фреймворку є Facebook, де його використовували для аналізу соціального графу.

Giraph++ – це фреймворк, який у своїй основі має Apache Giraph [33]. Основною відмінністю цієї гілки розвитку Giraph є використання гібридного підходу до обробки та розділення даних. Коли Giraph є вершино-центричним фреймворком, Giraph++ є гібридним, де одиницею якою оперує система є не вершина, а компонент графу.

Apache Spark – це фреймворк/платформа, що розроблена для обробки розподілених даних, наприклад на кластері Hadoop [34]. Створена як альтернатива MapReduce моделі, Spark має розширені можливості і забезпечує більшу продуктивність як при обробці даних у оперативній пам’яті так і на носії даних (будь то SSD чи HDD). Сам фреймворк надає можливості використовувати його на мовах Scala, Java, Python, R і здатен працювати з даними у кластерах та сховищах Elasticsearch, MongoDB, Apache Kafka, Delta Lake, Kubernetes, Apache Airflow,

Parquet, Microsoft SQL Server, Cassandra, Apache Orc, Apache Hadoop. Частиною фреймворку, що відповідає за роботу над графами є GraphX (Apache Spark GraphX). Це один з компонентів Spark який використовується для паралельної обробки графів. На високому рівні абстракції, цей компонент представляє системі розширення стандартного елементу даних (RDD – Resilient Distributed Dataset) в Spark – абстракцію графу (орієнтований мультиграф з додатковими значеннями у вершинах та ребрах). GraphX надає реалізацію різних алгоритмів а також особистий варіант Pregel API.

GiraphX – фреймворк, одна з гілок розвитку Apache Giraph [35]. Її відмінність у тому, що вона використовує модифіковану версію BSP, де замість черги повідомлень система може читати дані з вузла напряму і виконувати над ними операції. Таким чином ця «версія» Giraph більш придатна для алгоритмів, яким потрібна координація (фарбування графів, кластеризація, тощо).

GraphHP – фреймворк який використовує BSP (Bulk Synchronous Parallel) модель розподіленої обробки даних, але з додатковими модифікаціями для паралельної обробки [36]. GraphHP є вершино-центричним, ітеративним фреймворком, реалізує гібридну модель виконання суперкроків, де відокремлюються етапи опрацьовування даних в вузлах (partition) і в загальній системі. В етапі, де дані опрацьовуються локально (на вузлах), система мінімізує використання синхронізації і повідомлень до інших вузлів. Таким чином всередині вузла обчислення проходять швидше, а на рівні мережі (кластеру) використовується звичайні методи BSP для комунікації і синхронізації.

Cassovary – фреймворк створений в Twitter, для обробки великих графів, написана на мові програмування Scala і працює в середовищі JVM [37]. Цей фреймворк розроблений для ефективного опрацювання великої кількості даних, типовим його використанням є аналіз великих мереж та пошук корисних даних (data mining). Фреймворк не спроектований для використання у розподілених системах, він розрахований для використання на одній машині (комп’ютері). Прикладом його практичного використання є робота «WTF – Who to Follow» та «Similar to» алгоритмів в Twitter в минулому [38].

X-Stream – фреймворк розроблений для обробки графів в оперативній та постійні пам’яті, на одному комп’ютері чи системи з загальною пам’яттю [39]. Він являється ребро-центричним і використовує gather-apply-scatter модель обробки. Однією з особливостей системи є увага до процесів читання/запису даних, X-Stream намагається використовувати послідовний запис/читання, так як він виконується швидше ніж рандомний (random access).

Chaos – фреймворк, розроблений на основі X-Stream з покращенням у вигляді підтримки роботи на множині машин (кластері) [40]. Chaos сприймає розподілену пам’ять в мережі як звичайний ординарний диск і використовує техніки розподілення роботи для балансування навантаження на вузли.

GRACE – фреймворк для ітеративної обробки великих графів, використовує модифіковану BSP (Bulk Synchronous Parallel) модель розподіленої обробки даних де синхронна модель виконання поєднання з асинхронною [41]. На вищому рівні роботі мережі фреймворк виконує роботу синхронно, але на більш нижніх рівнях, наприклад на вузлі, він надає можливість оброблювати дані асинхронно. Таким чином задача може бути виконана швидше, а ресурси кластеру будуть використані більш ефективно.

GPS (Graph Processing System) – фреймворк розроблений для роботи з великими графами на розподіленій системі (кластер машин) [42]. GPS аналогічний системам Pregel та Apache Giraph, вершино-центричний, використовує BSP (Bulk Synchronous Parallel) модель розподіленої обробки даних та аналогічну до MapReduce модель для роботи.

Medusa – фреймворк для обробки графів особливістю якого являється те, що для своїх обчислень він застосовує GPU (блок обробки графіки) [43]. Він надає розробникам функціонал для написання послідовного C/C++ коду для розрахунків на графах, що будуть виконуватися з застосуванням можливостей GPU. Medusa використовує API для надання цих можливостей, і автоматично виконує надані для неї команди з використанням паралелізації та оптимізаційних методів для роботи на GPU. Додатково цей фреймворк має можливості для роботи на багатьох GPU в

одному комп'ютері одночасно (гібридна архітектура системи з багатьма блоками обробників і загальною пам'яттю).

TOTEM – фреймворк розроблений для роботи у гібридних системах [44]. Він надає рушій для створення програмних рішень/алгоритмів для обробки графів, що працює на гібридних системах з CPU та GPU. Використовуючи стратегії для розподілення даних між операційними вузлами, ціллю фреймворку є оптимальне розділення задач та даних з урахуванням особливостей системи, потужності/характеристик його операційних вузлів для швидкого та ефективного вирішення задач.

Представимо основні характеристики деяких графових фреймворків у таблиці, де будуть зібрані відомості по їх програмній моделі, архітектурі, моделі обробки та типу сховища даних (див. табл. 2.2).

Таблиця 2.2 – Фреймворки для обробки графів [Таблиця виконана самостійно*]

Назва	Програмна модель	Модель обробки	Архітектура системи	Координаційна модель	Сховище даних
Pregel	вершино-центрична	BSP	розподілена	синхронна	постійна пам'ять
Apache Giraph	вершино-центрична	BSP	розподілена	синхронна	постійна пам'ять
Giraph++	компонентно-центрична	BSP	розподілена	синхронна / асинхронна	постійна пам'ять
GiraphX	вершино-центрична	BSP	розподілена	асинхронна	постійна пам'ять
GPS	вершино-центрична	BSP	розподілена	синхронна	постійна пам'ять
X-Stream	ребро-центрична	Scatter-Gather	на одній машині	синхронна	постійна пам'ять
Medusa	вершино-центрична	EMV	гібридна	синхронна	оперативна пам'ять
GraphHP	вершино-центрична	BSP	розподілена	синхронна	постійна пам'ять
Chaos	ребро-центрична	GAS	розподілена	синхронна	постійна пам'ять
Cassovary	вершино-центрична	BSP	на одній машині	синхронна	постійна пам'ять

Кінець таблиці 2.2

Назва	Програмна модель	Модель обробки	Архітектура системи	Координаційна модель	Сховище даних
GRACE	вершино-центрична	Три-фазна	на одній машині	асинхронна	постійна пам’ять
TOTEM	вершино-центрична	BSP	гібридна	асинхронна	оперативна пам’ять
Apache Spark GraphX	вершино-центрична	BSP	розподілена	синхронна	постійна пам’ять

3 БАЗИ ДАНИХ ТА СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ ДЛЯ ГРАФІВ

3.1 СУБД для графів як аналог фреймворкам

Одними з сукупних завдань у сфері обробки графів є саме зберігання їх як даних. Фреймворки, що були розглянуті у більшості відповідають за їх обробку та роботу з ними, деякі з них розраховані/підтримують роботу з зовнішніми (відносно них) системами зберігання даних, одними з яких є бази даних. Бази даних (СУБД) для збереження графів є більш відомими програмними продуктами ніж фреймворки для роботи з графами, так як вони використовуються задля збереження даних. Не завжди програмні системи, де використовуються такі БД, потребують гнучкого та специфічного функціоналу, що пропонують розглянуті у попередніх розділах фреймворки. Самі СУБД до відповідних БД надають достатньо функціоналу для роботи над задачами збереження, пошуку, виконання простих/складних запитів, оновлення, тощо. СУБД не є фреймворками, але розробники таких рішень та треті сторони розробляють та надають фреймворки, що працюють з відповідними графовими БД/СУБД і їх функціоналу достатньо для роботи систем не потребуючих виконання комплексних завдань, для яких необхідно застосування свого індивідуального/нестандартного алгоритму обробки чи детального налаштування та оптимізації роботи системи. Такі фреймворки пов’язують програмні системи та БД/СУБД і надають можливість використовувати та зберігати дані у вигляді графів. Тому було вирішено також коротко розглянути популярні СУБД, що використовуються для збереження та роботи з графами (даних) проаналізувати їх характеристики та порівняти між собою. Завдяки цьому буде охоплена більша картина позиціонування програмних рішень у сфері фреймворків для обробки великих графів.

3.2 Огляд множини альтернатив

Для огляду у роботі було обрано 5 популярних графових баз даних на основі аналізу рейтингового списку, що створений редакторами та спільнотою веб ресурсу

DB-Engines [45]. Цими системами є Neo4j, Microsoft Azure Cosmos DB, Virtuoso, Amazon Neptune, GraphDB. Далі наведені короткі описи щодо обраних баз даних.

Neo4j — це система керування графовою базою даних, розроблена Neo4j, Inc [46] (див. рис. 3.1). Це ACID-сумісна транзакційна база даних із власним функціоналом зберігання і обробки графів. Neo4j доступна у «community edition» версії без відкритого вихідного коду (non-open-source), ліцензованого з модифікацію GNU General Public License, можливостями резервного онлайн копіювання та розширеннями для «високої доступності» (ліцензованими згідно з закритою комерційною ліцензією). Також є версії Neo4j із цими розширеннями на закритих комерційних умовах. Neo4j реалізований на Java та доступний із програмного забезпечення, написаного іншими мовами, використовуючи мову запитів Cypher через транзакційну кінцеву точку HTTP або через бінарний протокол «Bolt».

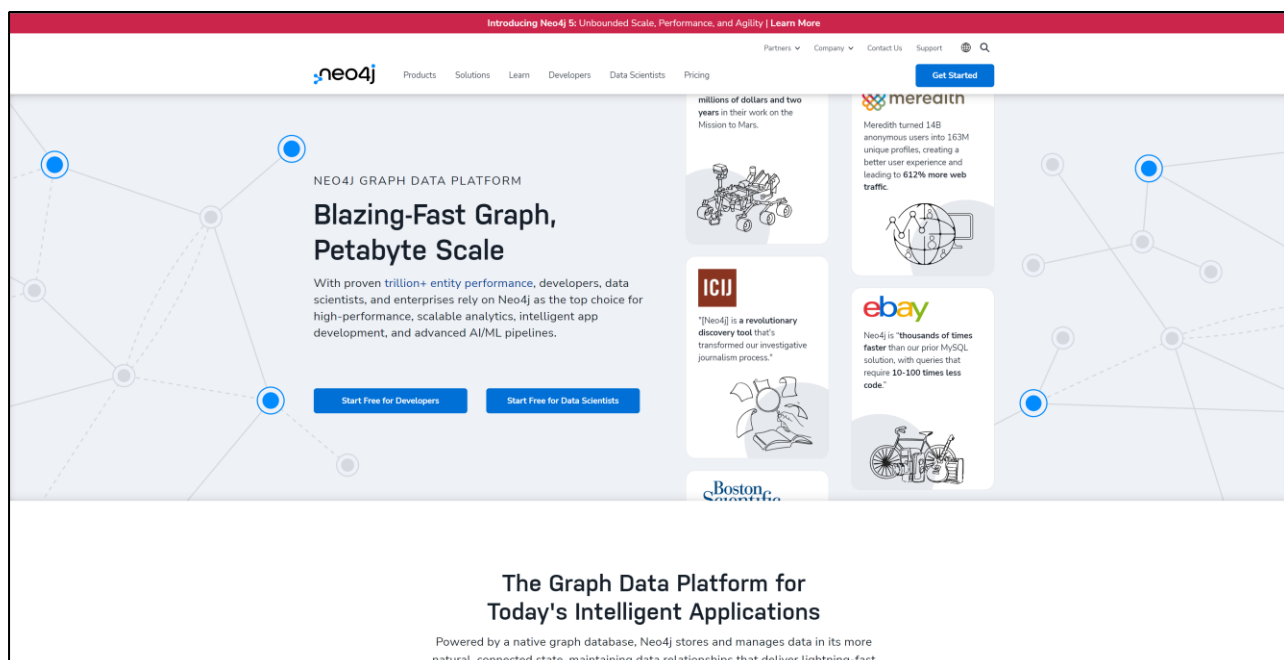

Рисунок 3.1 – Головна сторінка веб-сайту Neo4j [46]

Azure Cosmos DB — це глобально розподілена багатомодельна (multi-model) служба Microsoft для баз даних, що не залежить від схем, горизонтально масштабується і зазвичай класифікується як база даних NoSQL [47] (див. рис. 3.2).

Cosmos DB зберігає «елементи» в «контейнерах», причому ці дві концепції відображаються по-різному залежно від використовуваного API (наприклад, це будуть «документи» в «колекціях» при використанні сумісного з MongoDB API) . Контейнери згруповані в «бази даних», які аналогічні просторам імен над контейнерами. Контейнери не залежать від схеми, що означає, що жодна схема не застосовується при додаванні елементів.

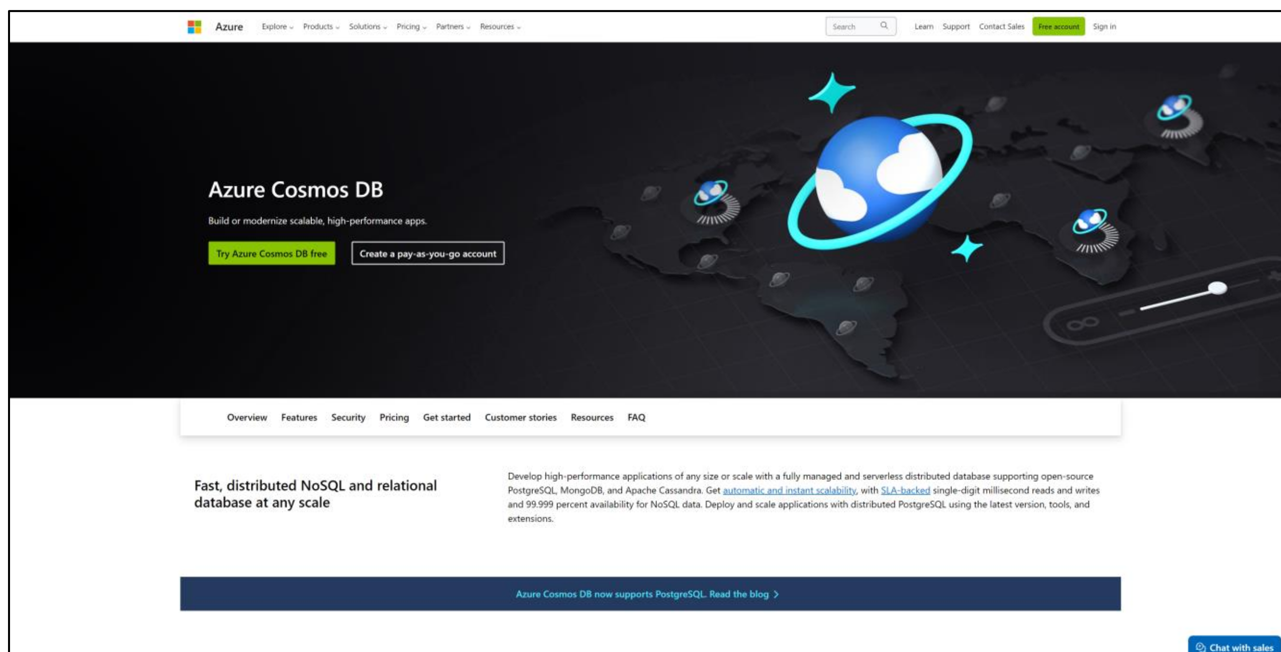

Рисунок 3.2 – Головна сторінка веб-сайту Azure Cosmos DB [47]

Virtuoso Universal Server — це гібрид проміжного програмного забезпечення та механізму баз даних, який поєднує в собі функціональність традиційної системи керування реляційною базою даних (RDBMS), об'єктно-реляційної бази даних (ORDBMS), віртуальної бази даних, RDF, XML, вільного тексту, сервера веб-додатків і файлового сервера в єдиній системі [48] (див. рис. 3.3). Замість того, щоб мати виділені сервери для кожної з вищезгаданих функціональних сфер, Virtuoso є «універсальним сервером»; він забезпечує єдиний багатопотоковий серверний процес, який реалізує кілька протоколів. Безкоштовна версія Virtuoso Universal Server з відкритим кодом також відома як OpenLink Virtuoso.

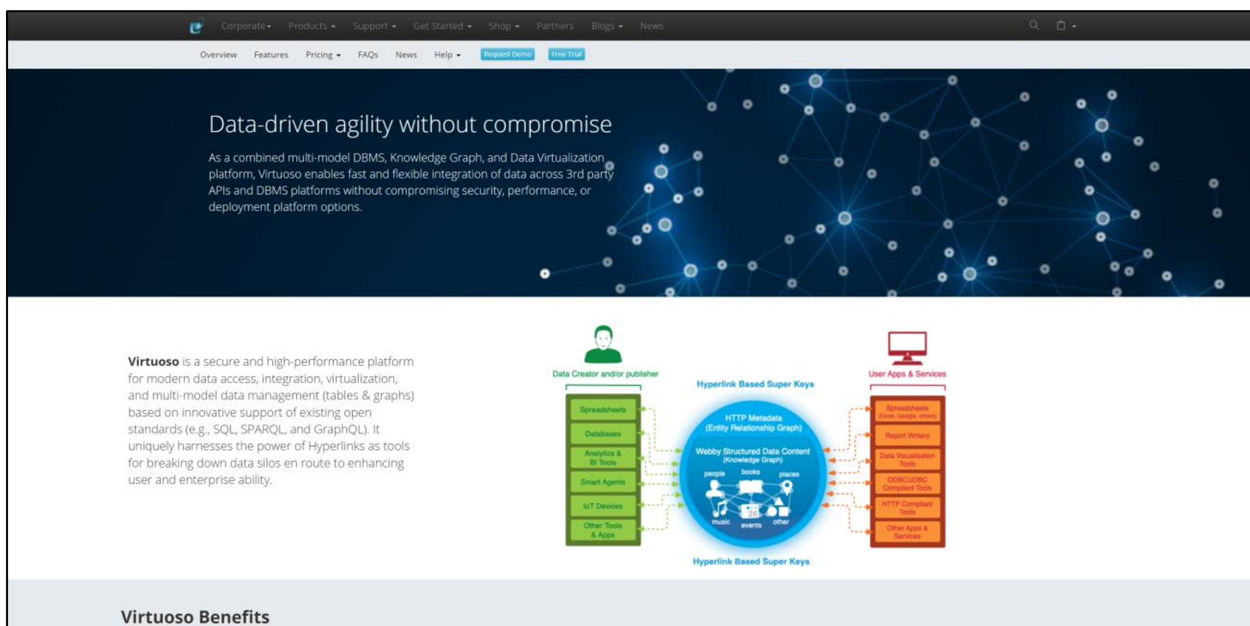

Рисунок 3.3 – Головна сторінка веб-сайту Virtuoso [48]

Amazon Neptune — графічна база даних, опублікована та керована Amazon.com. Вона використовується як веб-сервіс і є частиною Amazon Web Services (AWS) [49] (див. рис. 3.4). Amazon Neptune підтримує популярні графові моделі властивостей і RDF W3C, а також відповідні мови запитів Apache TinkerPop Gremlin, openCypher і SPARQL, включаючи інші продукти Amazon Web Services.

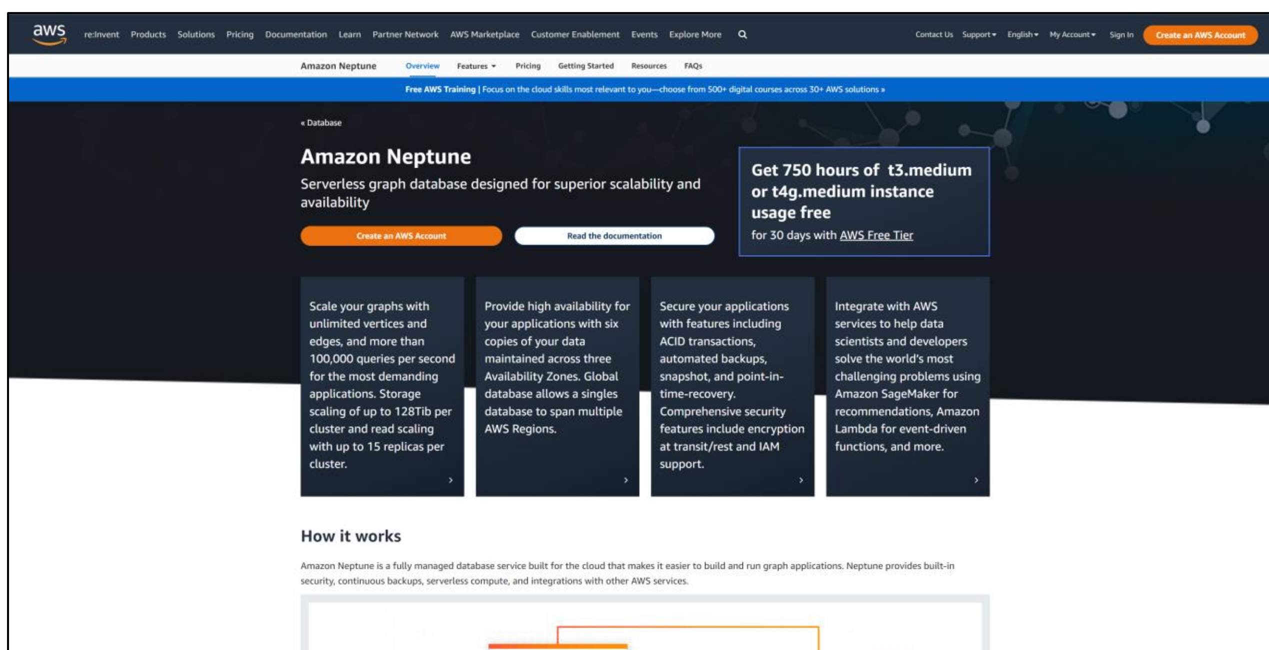

Рисунок 3.4 – Головна сторінка веб-сайту Amazon Neptune [49]

Ontotext GraphDB (раніше відомий як BigOWLIM) — це система управління графовою базою даних і інструмент виявлення знань сумісний із RDF і SPARQL і доступний як кластер високої доступності [50] (див. рис. 3.5). GraphDB використовується для зберігання та керування семантичними даними Knowledge Graph. Він створений на основі архітектури RDF4J, реалізованої через рівень зберігання та припущень (inference layer) RDF4J (SAIL). Архітектура складається з трьох основних компонентів: Workbench (веб орієнтований інструмент керування), Engine, Connectors.

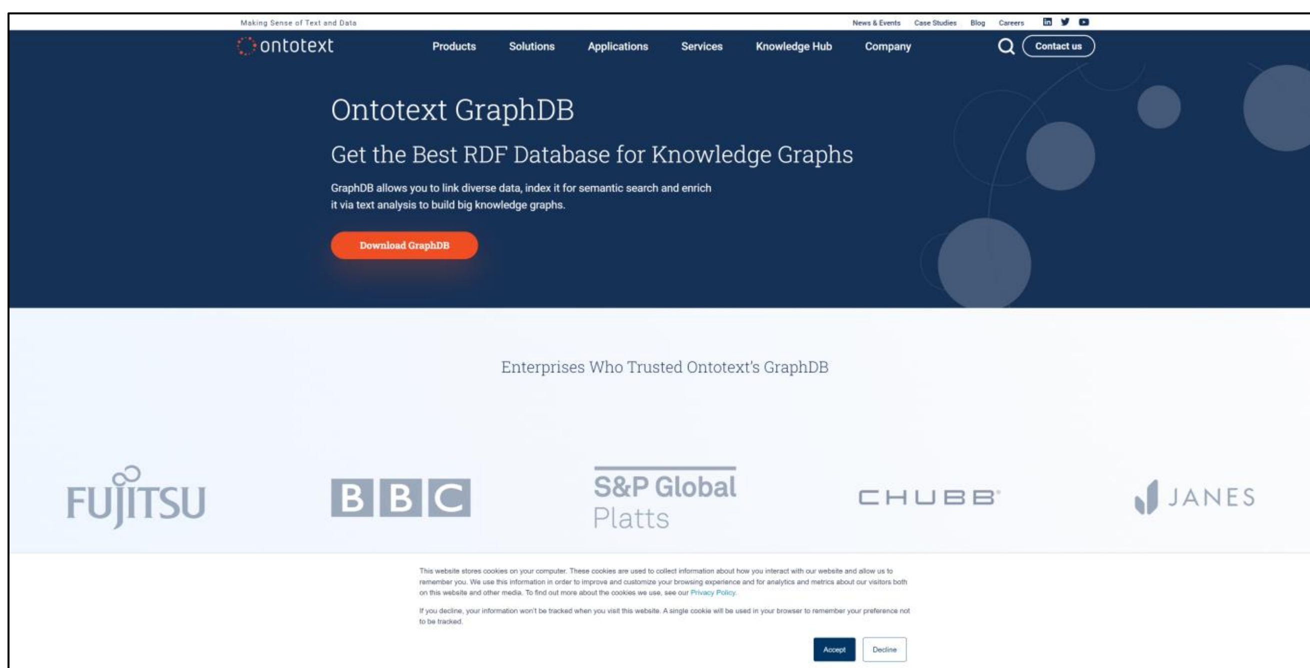

Рисунок 3.5 – Головна сторінка веб-сайту Ontotext GraphDB [50]

Таким чином ми обрали множину популярних та відомих систем управління графовими базами даних та баз даних, що підтримують можливість роботи з графовими структурами.

3.3 Аналіз характеристик обраних СУБД

Розглянувши загальний опис обраних баз даних можемо зробити висновок, що вони є багатогранними продуктами з різноманітними характеристиками, що дозволяють не тільки зберігати дані, але і виконувати різноманітні задачі, будь то

розрахунки чи обробка даних. Окрім цього вони надають можливості використання переваг хмарної архітектури, що не тільки спрощує роботу з їх обслуговування та планування інфраструктури системи, але і надає велику кількість можливостей інтеграції всередині хмари окрім використання фреймворків для роботи з ними. Через свою гнучкість у використанні ці елементи (бази даних для графів) мають більшу популярність та інформаційне освітлення ніж просто фреймворки для обробки графів. Використаємо дані про їх характеристики, щоб створити їх опис за єдиною структурою і використаємо це для їх порівняння.

3.3.1 Критерії оцінки характеристик баз даних

СУБД для графів досить різноманітні і у порівнянні з реляційними базами даних і не мають такої схожості між собою у типах властивостей, тому були обрані наступні, що можуть бути привласнені усім обраним екземплярам:

– рейтингова оцінка – оцінка, що надана веб ресурсом DB-Engines. Це база знань, що зберігає відомості про бази даних і має у собі розділ з рейтингом та оцінками популярності різноманітних баз даних. При наданні оцінки враховуються наступні фактори: кількість згадок системи на сайтах; загальний інтерес до системи; частота технічних обговорень системи; кількість пропозицій роботи, в яких згадується система; кількість профілів у професійних мережах, в яких згадується система; актуальність в соціальних мережах;

– вид ліцензії – open source чи commercial. Важливо під час вибору для використання у проектах – в залежності від цього необхідно дотримуватись певних ліцензійних угод, що можуть не відповідати критеріям проекту;

– доступність роботи через «хмари» – важливо при виборі БД для проектів, може не відповідати вимогам проекту чи законодавства, впливає на вартість розробки та її складність;

– операційні системи, що підтримуються – операційні системи на яких можливо запустити екземпляр БД;

– API та точки доступу, що підтримуються БД – технології за якими можливо взаємодіяти з БД;

– мови програмування, що підтримуються (офіційні конектори / бібліотеки);

– підтримка розподіленого зберігання – зберігання різних даних на множині різних екземплярах БД;

– підтримка реплікації – зберігання копії даних на множині різних екземплярів БД;

– підтримка MapReduce – підтримка MapReduce операцій;

– концепт транзакційності – підтримка ACID [51].

3.3.2 Розробка шкали оцінок за критеріями

Для порівняння обраних СУБД використаємо їх характеристики як критерії оцінки. Для переведення обраних критеріїв вибору будуть використовуватися нижче описані значення / шкали.

Рейтингова оцінка: значення будуть братися безпосередньо з рейтингу на веб-ресурсі db-engines.com.

Вид ліцензії: деякі бази даних мають різні види, що розповсюджуються за різними ліцензіями, тому буде братися найбільш відкрита з доступних. Для видів ліцензії будемо використовувати наступну шкалу:

– open source [52] – 3 бали;

– доступні Open Source версії – 2 бали;

– commercial – 1 бал.

Доступність роботи через «хмари»: бази даних можуть бути доступні лише у «хмарах» чи можуть бути використані на власних технічних засобах. Для даного критерію будемо використовувати наступну шкалу:

– тільки у «хмарах» (cloud based) – 0 балів;

– можливо розгорнути локально – 1 бал.

Операційні системи, що підтримуються: БД може підтримувати роботу на різноманітних операційних системах, тому для оцінки відзначимо найпоширеніші більшим числом (2) та менш поширені меншим (1) і вирахуємо суму, що вийшла під час переліку ОС, що підтримуються.

Список ОС по категоріям:

– поширені: Linux, Windows, Mac OS;

– непоширені: FreeBSD, AIX, Solaris і усі інші.

API та точки доступу, що підтримуються БД: підрахуємо кількість доступних технологій для роботи з БД (не будемо враховуючи популярність як коефіцієнт значущості для тих чи інших технологій).

Мови програмування, що підтримуються (офіційні конектори / бібліотеки): підрахуємо кількість технологій, що підтримуються при цьому під час рахування суми позначимо більш поширені більшою ціною. Оцінка для технологій:

– .Net, Java, JavaScript, PHP, Python, C++ – 2 бали;

– усі інші 1 бал.

Підтримка розподіленого зберігання: один бал при наявності, нуль балів при відсутності.

Підтримка реплікації: один бал при наявності, нуль балів при відсутності.

Підтримка MapReduce: один бал при наявності, нуль балів при відсутності.

Концепт транзакційності: для оцінки буде використовуватись наступна шкала:

– повне виконання ACID – 10 балів;

– атомарність на рівні серії запитів – 5 балів;

– на рівні запису – 1 бал.

3.3.3 Моделювання задачі прийняття рішення для порівняння та вибору найкращої БД

На основі обраних критеріїв вибору створимо таблицю з описом властивостей обраних баз даних (див. таблицю 3.1 та 3.2). Далі на основі цієї таблиці і шкали оцінок за критеріями перетворимо дані властивостей СУБД на їх оцінку у числовому еквіваленті для подальших розрахунків (див. табл. 3.3 та 3.4).

Таблиця 3.1 – Характеристики баз даних [Таблиця виконана самостійно*]

	Популярність	Ліцензія	Тільки у «Хмарі»	Операційна Система	API
Microsoft Azure Cosmos DB	37.95	Commercial	Так	hosted	DocumentDB API Graph API (Gremlin) MongoDB API RESTful HTTP API Table API
GraphDB	2.47	Commercial	Ні	Linux OS X Windows	GeoSPARQL GraphQL GraphQL Federation Java API JDBC RDF4J API RDFS RIO Sail API Sesame REST HTTP Protocol SPARQL 1.1
Amazon Neptune	2.89	Commercial	Так	hosted	RDF 1.1 / SPARQL 1.1 TinkerPop Gremlin 3.3
Neo4j	57.34	Partial open source	Ні	Linux OS X Solaris Windows	Bolt protocol Cypher query language Java API Neo4j-OGM RESTful HTTP API Spring Data Neo4j TinkerPop 3

Кінець таблиці 3.1

	Популярність	Ліцензія	Тільки у «Хмарі»	Операційна Система	API
Virtuoso	5.95	Partial open source	Ні	AIX FreeBSD HP-UX Linux OS X Solaris Windows	ADO.NET GeoSPARQL HTTP API JDBC Jena RDF API ODBC OLE DB RDF4J API RESTful HTTP API Sesame REST HTTP Protocol SOAP webservices SPARQL 1.1 WebDAV XPath XQuery XSLT

Таблиця 3.2 – Властивості баз даних, частина друга [Таблиця виконана самостійно*]

	Мові програмування	Розподілене зберігання	Підтримка реплікації	Підтримка MapReduce	Транзакційність
Microsoft Azure Cosmos DB	.Net Java JavaScript JavaScript (Node.js) MongoDB client drivers written for various programming languages Python	Так	Так	Так	ACID на рівні серії запитів

Кінець таблиці 3.2

	Мові програмування	Розподілене зберігання	Підтримка реплікації	Підтримка MapReduce	Транзакційність
GraphDB	.Net Clojure Java JavaScript (Node.js) PHP Python Ruby Scala	Ні	Так	Ні	ACID
Amazon Neptune	.Net Go Java JavaScript PHP Python Ruby Scala	Ні	Так	Ні	ACID
Neo4j	.Net Clojure Elixir Go Groovy Haskell Java JavaScript Perl PHP Python Ruby Scala	Так	Так	Ні	ACID
Virtuoso	.Net C C++ Java JavaScript Perl PHP Python Ruby Visual Basic	Так	Так	Так	ACID

Перевіряючи отримані дані (у таблицях 3.3, 3.4) за принципом Парето, було знайдено бази даних, що можливо виключити з обчислення, так як вони за усіма показниками гірше ніж певна інша база даних. Такими системами управління базами даних виявилися Amazon Neptune (при порівнянні з Virtuoso) та GraphDB (у порівнянні з Virtuoso). Таким чином для порівняння залишаються наступні бази даних: Microsoft Azure Cosmos DB, Neo4j, Virtuoso.

Таблиця 3.3 – Оцінка властивостей баз даних [Таблиця виконана самостійно*]

	Популярність	Ліцензія	Тільки у «Хмарі»	Операційна Система	API
Amazon Neptune	2.89	1	0	0	2
GraphDB	2.47	1	1	6	11
Microsoft Azure Cosmos DB	37.95	1	0	0	5
Neo4j	57.34	2	1	7	7
Virtuoso	5.95	2	1	10	16

Таблиця 3.4 – Оцінка властивостей баз даних, частина 2 [Таблиця виконана самостійно*]

	Мові програмування	Розподілене зберігання	Підтримка реплікації	Підтримка MapReduce	Транзакційність
Amazon Neptune	13	1	1	0	10
GraphDB	13	1	1	0	10
Microsoft Azure Cosmos DB	11	1	1	1	5
Neo4j	18	1	1	0	10
Virtuoso	16	1	1	1	10

3.3.4 Розрахунок задачі прийняття рішення

Для вирішення задачі прийняття рішення щодо найкращої СУБД за обраними характеристиками скористаємось методом на основі теорії корисності, а саме за допомогою лінійної адитивної згортки з ваговими коефіцієнтами. Цей метод був обраний, щоб врахувати у результаті всі критерії, що розглядаються, з урахуванням важливості самих критеріїв, а також допустити закриття одного недоліку в критерії перевагою в іншому критерії для СУБД, що розглядається. Досягається це завдяки згорткової оптимізаційної моделі:

$$\max \sum_{j=1}^{n} \alpha_j b_j a_{ij}$$

Для цього спочатку розрахуємо вагові коефіцієнти. Визначимо їх через ранжування (див. табл. 3.5). Розставимо критерії за їх важливістю:

- рейтингова оцінка (популярність);
- API, мова програмування;
- операційна система, транзакційність;
- розподілене зберігання, підтримка реплікації;
- розташування тільки у хмарі, підтримка MapReduce;
- ліцензія.

Таблиця 3.5 – Визначення вагових коефіцієнтів [Таблиця виконана самостійно*]

Критерій	Ранжування	Є рівноцінні критерії	Ранжування з рівноцінними критеріями	Ваговий коефіцієнт
Рейтингова оцінка	10	Ні	10	10 / 55 = 0.19
API	9	Так	8.5	8.5 / 55 = 0.16
Мова програмування	8	Так	8.5	8.5 / 55 = 0.16

Кінець таблиці 3.5

Критерій	Ранжування	Є рівноцінні критерії	Ранжування з рівноцінними критеріями	Ваговий коефіцієнт
Операційна система	7	Так	6.5	6.5 / 55 = 0.12
Транзакційність	6	Так	6.5	6.5 / 55 = 0.12
Розподілене зберігання	5	Так	4.5	4.4 / 55 = 0.08
Підтримка реплікації	4	Так	4.5	4.4 / 55 = 0.08
Розташування тільки у хмарі	3	Так	2.5	2.5 / 55 = 0.04
Підтримка MapReduce	2	Так	2.5	2.5 / 55 = 0.04
Ліцензія	1	Ні	1	1 / 55 = 0.01

Далі перейдемо до нормування критеріїв без еталонів для СУБД, що залишилися для розгляду (див. табл. 3.6, 3.7). Так як усі критерії відповідають одному принципу оптимізації (більше – краще), то нам не потрібно приводити їх до одного принципу. Для самого нормування використаємо метод нормування критеріїв без еталонів (з урахуванням min і max):

$$f = \frac{f_{\text{вимір.}} - f_{min}}{f_{max} - f_{min}}$$

Таблиця 3.6 – Нормування критеріїв без еталону [Таблиця виконана самостійно*]

	Популярність	Ліцензія	Тільки у «Хмарі»	Операційна Система	API
Microsoft Azure Cosmos DB	0.65	0	0	0	0
Neo4j	1	1	1	0.7	0.18
Virtuoso	0	1	1	1	1

Таблиця 3.7 – Нормування критеріїв без еталону, друга частина [Таблиця виконана самостійно*]

	Мові програмування	Розподілене зберігання	Підтримка реплікації	Підтримка MapReduce	Транзакційність
Microsoft Azure Cosmos DB	0	1	1	1	0
Neo4j	1	1	1	0	1
Virtuoso	0.71	1	1	1	1

У результаті отримуємо наступні формули для згортки за критеріями:

$$w(x) = 0.19f_1 + 0.01f_2 + 0.04f_3 + 0.12f_4 + 0.16f_5 + 0.16f_6 + 0.08f_7 + 0.08f_8 + 0.04f_9 + 0.12f_{10}$$

$$w(a_1) = 0.19 * 0.65 + 0.01 * 0 + 0.04 * 0 + 0.12 * 0 + 0.16 * 0 + 0.16 * 0 + 0.08 * 1 + 0.08 * 1 + 0.04 * 1 + 0.12 * 0 = 0.3235$$

$$w(a_2) = 0.19 * 1 + 0.01 * 1 + 0.04 * 1 + 0.12 * 0.7 + 0.16 * 0.18 + 0.16 * 1 + 0.08 * 1 + 0.08 * 1 + 0.04 * 0 + 0.12 * 1 = 0.7928$$

$$w(a_3) = 0.19 * 0 + 0.01 * 1 + 0.04 * 1 + 0.12 * 1 + 0.16 * 1 + 0.16 * 0.71 + 0.08 * 1 + 0.08 * 1 + 0.04 * 1 + 0.12 * 1 = 0.7636$$

Таким чином у результаті кращим варіантом за розрахунками є Neo4j з результатом $Z^* = 0.7928$ (див. табл. 3.8).

Таблиця 3.8 – Результати згортки критеріїв [Таблиця виконана самостійно*]

	Z^*
Amazon Neptune	Не є оптимальним за Парето, не розраховувався
GraphDB	Не є оптимальним за Парето, не розраховувався
Microsoft Azure Cosmos DB	0.3235
Neo4j	0.7928
Virtuoso	0.7636

За результатами порівняння характеристик СУБД графових баз даних ми здатні виділити 3 найкращих з заздалегідь обраних, а саме: Neo4j, Virtuoso, Microsoft Azure Cosmos DB. Також треба зауважити, що розрахунки проводилися за обмеженим набором характеристик (що ми обрали серед усіх можливих) та які зазначені самим розробником (тому можуть не відповідати роботі у реальному середовищі), а також не враховані комплексні та суб'єктивні характеристики і властивості, такі як легкість застосування, підтримка від розробників і спільноти користувачів, тощо.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Основні зібрані дані під час дослідження фреймворків та СУБД для роботи з графами

Під час проведення дослідження було розібрано різноманітні аспекти роботи фреймворків та приведені приклади існуючих фреймворків та СУБД, що здатні обробляти великі об'єми графових даних, а також самі графи як структури даних і їх приклади використання.

Зібравши дані про графи як абстрактну структуру та як структуру даних у програмних системах були виділені їх основні властивості, що пропонують велику кількість варіантів використання. Графи є структурою, що виділяють у першу чергу відносини між даними – їх зв'язки, близькість/схожість, відносність за різними ознаками, тощо. Це надає можливість поглянути на інформацію під іншим кутом і використовувати її для вирішення відносно незвичайних задач. Графи природньо підходять для відображення різноманітних зрізів/сфер з реального життя, де об'єкти взаємодіють друг з другом, формують зв'язки та встановлюють відповідності/відношення між собою. Таким чином у багатьох випадках для збору інформації чи створенні моделі (даних) не потрібно їх багато обробляти/конвертувати та підводити під певні стандарти чи встановлені правила, як це робиться для реляційних структур. Окрім цього, завдяки своїм властивостям, вони дозволяють використовувати різні специфічні алгоритми більш ефективно (такі як пошук схожих об'єктів, виділення популярних, знаходження найкоротшого шляху, виділення спільнот, моделювання нейронної мережі, тощо). В роботі також наведена класифікація загальних алгоритмів, що використовуються в обробці графів.

Розглянуті основні проблеми та виклики перед фреймворками, СУБД та іншими системами для опрацювання графів, що виникають завдяки їх властивостям. Основними з них виявилися:

– залежність ходу обробки графу від самої топології графу, що значно ускладнює прогнозування роботи процесів обчислень та можливостей їх оптимізації;

– змінність та гнучкість графу впливають на хід роботи алгоритму, що в свою чергу може знизити ефективність наявних оптимізацій на різних рівнях роботи системи (оптимізації на рівні програм, процесору, компілятора/інтерпретатора, тощо);

– висока залежність від швидкості читання, завантаження та передачі даних, що спричинені особливостями алгоритмів, де існує високе співвідношення операцій над даними до самих обчислень з їх використанням.

Під час розбору фреймворків для обробки великих графів були виділені основні узагальнені етапи їх роботи: читання даних, підготовка, розділення даних, обчислення, запис даних. Також були встановлені основні проблеми в ефективній роботі фреймворків через апаратні обмеження, головною з яких є характеристики пам'яті, а саме швидкість і розмір/об'єм. Оглянуті прийняті рішення які використовуються для забезпечення можливостей обробки великих даних. До них входять застосування різних системних архітектур (розподілена система/кластер, одиночна на основі однієї машини, гібридна архітектура), використання різних стратегій/моделей обробки задач (синхронні/асинхронні методи виконання процесів, гібридні рішення), підходи до розподілення даних та встановлення акценту на елементах графу при його опрацюванні (вершино-центричний, ребро-центричний, компонентний підходи). Наведені приклади існуючих фреймворків та їх опис на основі вище розібраних характеристик і властивостей.

Також в рамках дослідження були переглянуті супутні програмні рішення для обробки графів, а саме СУБД графових баз даних. Огляд показав, що вони надають базові можливості для роботи з графами, які є достатніми для простих програмних систем, що не потребують високої гнучкості чи можливостей виконувати складні або специфічні задачі. Для огляду було взято 5 популярних СУБД для яких зроблений короткий опис. В ході роботи були виділені найголовніші характеристики СУБД, було проведене їх порівняння і вирішена задача пошуку найкращої з них (на основі раніше вибраних та описаних характеристик).

4.2 Аналіз отриманих даних

На основі отриманих даних під час дослідження ми можемо зробити висновки щодо використання графів у системах, фреймворків розроблених для обробки великих графів та СУБД для них. Графи виявилися багатогранною структурою, що надають різноманітні можливості для виконання різних задач. Вони використовуються у багатьох важливих та великих системах, але попри це є відносно «непопулярними» через специфіку їх використання та непотрібність такого функціоналу в більшості випадках. Через це відомості про фреймворки для роботи з графами також не є розповсюдженими (у порівнянні з рішеннями для реляційних чи об’єктових структур). Самі властивості графів та алгоритмів додають проблем та викликів для фреймворків разом з тими, що накладаються апаратними обмеженнями. Для їх вирішення або часткового покриття у фреймворках використовуються різноманітні рішення: використання різних типів архітектур систем, моделей обробки, стратегій з розподілення даних та задач. Таким чином з’явилося різноманіття фреймворків, що мають різні характеристики, слабкі та сильні сторони, тому для вибору певного фреймворку, пригожого до встановлених задач, необхідно це враховувати. Складність використання, гнучкість та велика кількість можливостей не завжди необхідна у проєктах де використовуються графи, тому СУБД графових баз даних є раціональною альтернативою до графових фреймворків. Вони є готовим повноцінним рішенням для зберігання графів та виконання відносно простих задач над ними. Так як це окрема повноцінна система, що надає доступ до свого функціоналу фреймворками чи бібліотеками на різних мовах програмування чи взагалі можливістю робити запити до зовнішніх API (системи) вони є простим та універсальним вибором для рішень простих задач над графами.

4.3 Рекомендації з вибору фреймворків та СУБД для використання

На основі отриманої інформації ми також можемо сформулювати загальні рекомендації щодо вибору фреймворку чи СУБД для використання у своїх проєктах, вказати на моменти які потребують уваги при виборі та особливості які потрібно враховувати.

Першим моментом для розгляду є задачі які плануються виконуватись у програмній системі. Це є одним з найголовніших чинників при виборі інструменту для роботи з графами. Якщо необхідно виконувати власні розроблені алгоритми над графами чи постає необхідність виконувати складні задачі і мати гнучкість у налаштуваннях роботи та потенціал для майбутніх розширень системи, то серед розглянутих варіантів використання фреймворки для обробки графів є найкращим. Вони надають більшу гнучкість у розробці власних систем, містять реалізований функціонал по загальним алгоритмам обробки та дозволяють реалізовувати власні алгоритми обробки графів використовуючи вже готову методику обчислень (властиву обраному фреймворку). При виборі фреймворку, окрім сумісності з вашою програмною системою, потрібно враховувати середовище в якому програма буде працювати. Це можуть бути розподілені системи (кластери комп'ютерів), одиночна на основі однієї машини чи гібридна. В залежності від специфіки проєкту кожна з них може бути оптимальним варіантом. Якщо система для якої ви обираєте фреймворк являється розподіленою чи має справу з великими даними, що вимірюються терабайтами, то фреймворки для розподілених систем є найкращим вибором. Одними з прикладів застосування являються Facebook та Google, що використовують фреймворки такого класу для вирішення своїх задач. Але при роботі з даними, що вимірюються гігабайтами одиночна та гібридна архітектури в фреймворках являються більш раціональними та ефективними. Прикладом до цього є Twitter, що використовував Cassovary (фреймворк розрахований для роботи на одній машині) для роботи своїх алгоритмів підбору релевантних аккаунтів/постів. При цьому це була робота з великими даними у рамках графу, але самі по собі фізично вони займали відносно мало місця (граф з 10 мільйонів вершин та 1 мільярда ребер може займати лише 6 ГБ пам'яті). Окрім архітектури системи

необхідно звернути увагу на сховище даних, деякі фреймворки можуть працювати з даними у БД чи на диску (постійна пам’ять), іншим необхідно переводити усі дані в оперативну пам’ять для подальшої обробки.

Якщо постає необхідність виконувати прості задачі з графами, наприклад просто зберігати та маніпулювати даними/інформацією в графі, то використання СУБД для графових баз даних є найдоцільнішим. Це готові рішення/системи які відносно прості в використанні і інтеграції, можуть надавати достатній рівень функціоналу та можливостей для вашого проєкту. При виборі СУБД необхідно звернути увагу на різні аспекти їх використання. Потрібно перевірити підтримку операційних систем на яких вони працюють, доступність чи винятковість роботи у «хмарі» (це важливо, так як використання може не відповідати вимогам проєкту чи законодавства і впливає на вартість розробки та підтримки), роботу відповідного фреймворку/бібліотеки з певними мовами програмування та інші характеристики (підтримка розподіленого зберігання, реплікації, підтримка MapReduce та ACID, вид ліцензії open source чи commercial). Незважаючи на великий список точок уваги, їх використання є набагато простішим ніж розробка власного рішення з фреймворком, так як це готовий продукт, що працює з графами. Вам залишається реалізувати бізнес логіку системи і інтегрувати використання графової СУБД.

ВИСНОВКИ

В ході виконання роботи були досліджені фреймворки для обробки графів великого розміру, були встановлені їх характеристики та властивості, можливі середи їх використання, методи роботи, програмні та архітектурні рішення.

Протягом виконання атестаційної роботи було розглянуто:

– представлення графів як абстрактної структури, їх використання і розповсюдженість в сферах програмних рішень;

– основні проблеми та виклики, що постають перед фреймворками для обробки графів та методи і архітектурні рішення якими ці недоліки вирішуються чи частково компенсуються;

– високорівневе представлення процесів в роботі фреймворків, узагальнені етапи в їх обчисленнях;

– класифікація та перелік прикладів алгоритмів, що використовуються у роботі з графовими даними;

– системні архітектурні рішення, різноманіття та особливості стратегій/моделей обробки даних та виконання задач які використовуються фреймворками для забезпечення можливостей обробки великих графів;

– приклади існуючих фреймворків для обробки графів, їх опис та аналіз згідно переглянутих властивостей та характеристик;

– приклади існуючих СУБД для збереження графових даних, їх оцінка та аналіз згідно вказаних характеристик;

– оцінка отриманих результатів дослідження, вироблення висновків щодо програмних рішень у сфері обробки графів та рекомендацій для вибору систем згідно необхідних критеріїв.

Отримані результати можуть бути використані для розробки чи при проєктуванні фреймворків та систем, що повинні працювати з графами, а також для визначення використання тих чи інших архітектурних рішень, методик обробки даних, інструментів у відповідності до необхідних потреб.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Relational Database Overview. / Oracle Documentation. URL: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html (дата звернення 20.01.2023)

2. Richard J. Trudeau. Introduction to Graph Theory: Dover Publications, 2nd edition, 1994. 224 с.

3. Dan Sullivan Sullivan. NoSQL for Mere Mortals: Addison-Wesley Professional, 1st edition, 2015. 542 с.

4. The OPTE Project. Internet Network Graph. URL: https://www.opte.org/ (дата звернення 30.04.2023)

5. Wikimedia. Fraction of WWW relationships with Wikipedia node. URL: https://en.wikipedia.org/wiki/World_Wide_Web#/media/File:WorldWideWebAroundWikipedia.png (дата звернення 30.04.2023)

6. Meta Reports Fourth Quarter and Full Year 2022 Results. URL: https://investor.fb.com/investor-events/event-details/2023/Q4-2022-Earnings/default.aspx (дата звернення 30.04.2023)

7. Twitter Statistics For Marketers In 2023. / Demand Sage. URL: https://www.demandsage.com/twitter-statistics/ (дата звернення 30.04.2023)

8. Instagram Statistics for Marketers In 2023. / Demand Sage. URL: https://www.demandsage.com/instagram-statistics/ (дата звернення 30.04.2023)

9. Twitter Social Graph by Marc Smith. URL: https://www.flickr.com/photos/marc_smith/4618279087/ (дата звернення 30.04.2023)

10. Radouan Ait Mouha. Internet of Things (IoT) // Journal of Data Analysis and Information Processing 9, 2021. С. 77-101.

11. I. Afanasieva, N. Golian, O. Hnatenko, Y Daniiel, K. Onyshchenko. Data exchange model in the internet of things concept // Telecommunications and Radio Engineering, 2019. С. 869-878.

12. Internet of Things by jeferrb. / Pixabay. URL: https://pixabay.com/vectors/network-iot-internet-of-things-782707/?download (дата звернення 30.04.2023)

13. Bela Bollobas. Modern Graph Theory: Springer New York, 2013. 394 c.

14. Lumsdaine, D. Gregor, B. Hendrickson, J. Berry. Challenges in parallel graph processing // Parallel Processing Letters, 2007. C. 5-20.

15. Dominguez-Sal. A discussion on the design of graph database benchmarks // Performance Evaluation, Measurement and Characterization of Complex Systems - Second TPC Technology Conference, 2010. 40 c.

16. Maharshi J. P. Comparative Analysis of Search Algorithms // Journal of Computer Applications, 2018. 4 c.

17. Graph Communities Visualization by Paul Brodersen / Stack Overflow. URL: https://stackoverflow.com/questions/43541376/how-to-draw-communities-with-networkx (дата звернення 30.04.2023)

18. Graph PageRank Visualization / Scikit-network. URL: https://scikit-network.readthedocs.io/en/latest/tutorials/ranking/pagerank.html (дата звернення 30.04.2023)

19. Asus. Pro WS WRX80E. URL: https://www.asus.com/us/motherboards-components/motherboards/workstation/pro-ws-wrx80e-sage-se-wifi/techspec/ (дата звернення 30.04.2023)

20. Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. URL: http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf (дата звернення 30.04.2023)

21. Khovrat A., Kobziev V., Nazarov A., Yakovlev S. Parallelization of the VAR Algorithm Family to Increase the Efficiency of Forecasting Market Indicators During Social Disaster // Information Technology and Implementation, 2022. C. 222-233.

22. Apache Hadoop. MapReduce Tutorial: Apache Software Fondation, 2022. 42 c.

23. Rowstron A., Narayanan D., Dopnnelly A., O’Shea G., Douglas A. Nobody ever got fired for using Hadoop on a cluster // 1st International Workshop on Hot Topics in Cloud Data Processing, 2012. 5 c.

24. Cassovary: A Big Graph-Processing Library / Twitter Engineering. URL: https://blog.twitter.com/engineering/en_us/a/2012/cassovary-a-big-graph-processing-library (дата звернення 30.04.2023)

25. Solutions for the Data Center / Nvidia. URL: https://www.nvidia.com/en-us/data-center/ (дата звернення 30.04.2023)

26. Anthony P. GPU Computing: The Future of Computing: Proceedings of the West Virginia Academy of Science, 2018. 90 с.

27. The BSP Programming Model / MIT.edu URL: https://groups.csail.mit.edu/cag/bayanihan/papers/javapdc99/html/node2.html (дата звернення 30.04.2023)

28. Gonzales J.E, Low Y., Gu H., Bickson D., Guestrin C. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs // USENIX Symposium on Operating Systems Design and Implementation, 2012. С. 17-30.

29. Malewicz, M. H. J. C. Austern, A. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale. // ACM SIGMOD International Conference on Management of Data, 2010. С. 135-146.

30. Valiant G. A bridging model for parallel computation. // Association for Computing Machinery 33, 1990. С 103-111.

31. Apache Hadoop. / Apache Software Foundation. URL: https://hadoop.apache.org/ (дата звернення 30.04.2023)

32. Apache Giraph. / Apache Software Foundation. URL: https://giraph.apache.org/ (дата звернення 30.04.2023)

33. Tian Y., Balmin A., Corsten S. A., Tatikonda S., McPherson J. From "Think Like a Vertex" to "Think Like a Graph" // International Conference on Very Large Data Bases, 2014. 30 с.

34. Apache Spark Documentation. / Apache Software Foundation. URL: https://spark.apache.org/docs/latest/index.html (дата звернення 30.04.2023)

35. Taski S., Demirbas M., Giraphx: Parallel yet serializable large-scale graph processing. // International Conference on Parallel Processing, 2013. С. 458-469

36. Chen Q., Bai S., Li Z., Gou Z., Suo B., Pan W. GraphHP: A Hybrid Platform for Iterative Graph Processing: Cornell University Arxiv, 2017. 12 с.

37. Cassovary framework / GitHub. URL: https://github.com/twitter/cassovary (дата звернення 30.04.2023)

38. Gupta P., Goel A., Lin J., Sharma A., Wang D., Zadeh R. WTF: The Who to Follow Service at Twitter // International World Wide Web Conference Committee, 2013. 11 с.

39. Roy A., Mihailovic I., Zwaenepoel W. X-Stream: edge-centric graph processing using streaming partitions. // ACM Symposium on Operating Systems Principles, 2013. С. 472-488.

40. Chaos framework / GitHub. URL: https://github.com/bindscha/chaos (дата звернення 30.04.2023)

41. Kamhoua B. F., Zhang L., Ma K., Cheng J., Li B., Han B. GRACE: A General Graph Convolution Framework for Attributed Graph Clustering // ACM Transactions on Knowledge Discovery from Data 17, 2023. С. 1-31.

42. GPS: A Graph Processing System / Standford.edu. URL: http://infolab.stanford.edu/gps/ (дата звернення 30.04.2023)

43. Zhong J., He B., Medusa: Simplified Graph Processing on GPUs // EEE Transactions on Parallel and Distributed Systems 25, 2014. С. 1543-1552.

44. Gharaibeh A., Reza T., Santos-Neto E., Costa L. B., Sallinen S., Ripeanu M. Efficient Large-Scale Graph Processing on Hybrid CPU and GPU Systems: Cornell University Arxiv, 2014. 36 с.

45. DB-Engines. URL: https://db-engines.com/en/ (дата звернення 30.04.2023)

46. NEO4J Graph Data Platform. URL: https://neo4j.com/ (дата звернення 30.04.2023)

47. Azure Cosmos DB. URL: https://azure.microsoft.com/en-us/products/cosmos-db (дата звернення 30.04.2023)

48. Virtuoso Universal Server. URL: https://virtuoso.openlinksw.com/ (дата звернення 30.04.2023)

49. Amazon Neptune. URL: https://aws.amazon.com/neptune/ (дата звернення 30.04.2023)

50. Ontotext GraphDB. URL: https://www.ontotext.com/products/graphdb/ (дата звернення 30.04.2023)

51. Database Basics: ACID Transactions. URL: https://towardsdatascience.com/database-basics-acid-transactions-bf4d38bd8e26 (дата звернення 30.04.2023)

52. Andrew M. St. Laurent. Understanding Open Source and Free Software Licensing: O'Reilly Media Inc., 2004. 208 с.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

11. I. Afanasieva, N. Golian, O. Hnatenko, Y Daniiel, K. Onyshchenko. Data exchange model in the internet of things concept // Telecommunications and Radio Engineering, 2019. С. 869-878.

21. Khovrat A., Kobziev V., Nazarov A., Yakovlev S. Parallelization of the VAR Algorithm Family to Increase the Efficiency of Forecasting Market Indicators During Social Disaster // Information Technology and Implementation, 2022. С. 222-233.