Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів паралельної обробки великих об'ємів даних в iOS застосунку
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-3

Полурєзов Д.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. кафедри ПІ Кравець Н.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри ПІ _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Полурєзову Дмитру Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів паралельної обробки великих об’ємів даних в iOS застосунку»
Затверджена наказом по університету від 29.03.2024 № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 14.06.2024
3. Вихідні дані до роботи інформація щодо принципів паралельної обробки, особливості організації паралельної обробки даних на мобільних пристроях під управлінням iOS.
4. Перелік питань, що потрібно опрацювати в роботі
аналіз предметної галузі, постановка задачі дослідження, огляд та аналіз наукових першоджерел за тематикою дослідження, розробка моделей методів паралельної обробки даних в середовищі iOS для проведення експериментальних досліджень, проведення експериментів та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	22.01 – 11.02.24	*виконано*
2	Аналіз першоджерел в предметній області дослідження	12.02 – 24.02.24	*виконано*
3	Математичне моделювання предметної області	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Програмна реалізація моделі програмного забезпечення	25.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 23.04.24	*виконано*
8	Написання та оформлення статті та тез доповіді	17.04 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.04 – 30.05.24	*виконано*
10	Підготовка презентації та доповіді	31.05 – 2.06.24	*виконано*
11	Нормоконтроль	3.06 – 08.06.24	*виконано*
12	Рецензування	08.06 – 14.06.24	*виконано*
13	Занесення диплома в електронний архів	15.06.2024	*виконано*
14	Попередній захист	15.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	18.06.2024	*виконано*

Дата видачі завдання 22 січня 2024 р.

Студент (ка) _____ Полурєзов Д.С.
(підпис)

Керівник роботи _____ доц. кафедри ПІ Кравець Н.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 84 с., 44 рис., 27 джерел.

ПАРАЛЕЛЬНА ОБРОБКА, КОНВЕЄРИЗАЦІЯ, BIG DATA, SWIFT, iOS

Об’єктом дослідження є принципи організації паралельної обробки Big Data в середовищі iOS.

Мета роботи – визначення шляхів підвищення продуктивності iOS-додатків за рахунок порівняння методів паралельної обробки Big Data, що базуються на використанні GCD, NSLock та OperationQueue.

Результати роботи – проведені дослідження свідчать про можливість використання методів розпаралелювання для підвищення ефективності використання мобільних додатків під управлінням iOS для обробки великих об’ємів даних.

PARALLEL PROCESSING, CONVEYORIZATION, BIG DATA, SWIFT, iOS

The object of the discovery is the principles of organizing parallel processing of Big Data in the iOS environment.

The aim of the work is to identify ways to improve the performance of iOS applications by comparing methods of parallel processing of Big Data based on the use of GCD, NSLock, and OperationQueue.

The results of the study indicate the feasibility of using parallelization methods to enhance the efficiency of iOS-based mobile applications in processing of Big Data.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Полурєзов Дмитро Сергійович, студент гр. ІПЗм-22-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів паралельної обробки великих об’ємів даних в iOS застосунку», що буде представлена в екзаменаційну комісію 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(на) 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

ПЕРЕЛІК СКОРОЧЕНЬ

BI – Business Intelligence

CMP – Chip Multiprocessing

CPU – Central Processing Unit

GCD – GrandCentralDispatch

GPU – Graphics Processing Unit

MPI – Message Passing Interface

PMD – Personal Mobile Device

QoS – Quality of Service

SoC – System-on-a-Chip

ВСТУП

На теперішній час спостерігається зростаючий інтерес до проблем обробки великих об’ємів даних за допомогою мобільних пристроїв. Це стало актуальним зокрема через зростання обсягів інформації, яку користувачі споживають та генерують за допомогою своїх мобільних пристроїв. Мобільні застосунки стали важливим інструментом для доступу до великої кількості даних, включаючи текст, зображення, відео, сенсорні дані та інші типи інформації.

Вимоги користувачів до сучасних мобільних застосунків містять потребу в отриманні доступу до важливої інформації у будь-який момент і в будь-якому місці. Це означає, що застосунки повинні бути здатні обробляти великі об’єми даних та забезпечувати швидку і ефективну реакцію на запити користувачів. Більш того, в багатьох випадках користувачі очікують, що дані будуть оновлюватися в режимі реального часу, надаючи їм актуальну інформацію.

Актуальність теми дослідження обумовлена поширенням персональних мобільних пристроїв під управлінням операційної системи iOS та бажанням користувачів отримати за допомогою мобільного додатку прискорення обробки інформаційних запитів. Враховуючи особливості розробки та поширення програмних продуктів для операційної системи iOS, одним з можливих варіантів є використання методів паралельної обробки, яка є однією з основних стратегій для покращення продуктивності та забезпечення її швидкої реакції на запити користувачів.

Дослідження методів паралельної обробки великих об’ємів даних в iOS застосунку проводились у відповідності до наукових напрямків досліджень кафедри програмної інженерії ХНУРЕ [1-2], серед яких можна виділити:

– інформаційні технології дистанційного навчання та електронної комерції;

– інтелектуальний аналіз даних;

– розробка систем мобільного навчання.

Метою дослідження є визначення методів організації розпаралелювання виконання запитів за допомогою мобільних пристроїв під управлінням операційної системи iOS та порівняння швидкості їх виконання та завантаженості процесора.

Для досягнення цієї мети необхідно було побудувати математичну модель сховища неструктурованих даних, визначити параметри, за якими будуть визначатись результати виконання запитів до цих сховищ, спроектувати та виконати програмну реалізацію мобільного додатку, провести дослідження поведінки програмної системи та визначити критерії, за якими порівнювались програмні додатки та ефективність виконання запитів, зробити висновки щодо отриманих результатів.

Об’єктом дослідження виступають запити до великих обсягів неструктурованої інформації, які можуть виконуватись або послідовно, або з використанням засобів розпаралелювання шляхом формування окремих черг запитів до сховища.

Предметом дослідження є засоби організації паралельної обробки за допомогою мобільних пристроїв під управлінням iOS.

Серед наукових методів, які були використані під час підготовки кваліфікаційної роботи, для досягнення мети дослідження було проведено:

– теоретичний аналіз: використовувався під час вивчення літературних та наукових першоджерел для проведення аналізу існуючих теорій та методів паралельної обробки даних;

– емпіричні дослідження: проведення експериментів для оцінки продуктивності різних підходів до паралельної обробки даних;

– математичне моделювання: розробка математичних моделей подання BigData для оцінки ефективності паралельних алгоритмів;

– експериментальний метод: розробка прототипів iOS застосунків з реалізацією різних методів паралельної обробки;

– порівняльний аналіз: порівняння різних методів паралельної обробки даних за визначеними критеріями, такими як швидкість, час виконання запитів та завантаженість ядер процесора;

Використання цих методів дозволить ґрунтовно дослідити різні аспекти паралельної обробки великих об'ємів даних в iOS застосунках, знайти оптимальні підходи та розробити певні рішення для розв’язання реальних задач.

У якості практичних результатів проведених досліджень можна зазначити отриману за результатами експериментів інформацію щодо нерівномірності завантаженості енергоефективних та високопродуктивних ядер процесорів сімейства Apple Bionic, та залежність їх завантаженості від кількості черг до цих ядер та версії операційної системи iOS. Матеріали, що описують математичні основи проведених досліджень, були представлені на наукових конференціях та форумах, опубліковані у наукових виданнях [25, 28].

Отримані результати досліджень, що проводились в рамках написання кваліфікаційної роботи магістра, допомагають визначити, як методи паралельної обробки запитів можуть бути використані для покращення продуктивності мобільних застосунків, орієнтованих на обробку значних обсягів інформації, та створює підґрунтя для проведення подальших досліджень в цій галузі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

Необхідність у прискоренні обробки інформації та виконанні обчислень через розширене використання паралельних обчислень існує вже протягом кількох десятиліть. Прогрес та еволюція комп'ютерних архітектур зробили паралельні обчислення більш доступними та економічно ефективнішими, ніж це було раніше. Обсяги інформації, які щоденно утворюються та зберігаються, стрімко збільшуються в різних сферах людського життя, включаючи науку та інженерію, промисловість, медицину та розваги. Різноманітні типи вимог значним чином стимулюють розвиток та впровадження паралельних обчислень в інженерію програмного забезпечення.

У світі нових вимог до додатків, що з'являються на поточному етапі розвитку інженерії програмного забезпечення, на перший план виходять вимоги щодо підвищення ефективності обробки даних. Серед шляхів такого підвищення майбутнє сучасних обчислень слід шукати в паралелізмі. У переважній більшості прикладні програми, сучасні ігри та науково-технічні програми містять вимоги, які не обмежуються продуктивністю лише центрального процесору (CPU). Виникла нова загальна обчислювальна модель, в якій центральний процесор працює разом із графічним процесором (GPU). У цих паралельних конфігураціях, що зазвичай містять декілька багатоядерних процесорів (наприклад, CPU і GPU), закладений потенціал для задоволення зростаючих вимог до прискорення обчислень. Саме тому це дослідження присвячено аналізу та порівнянню методів паралельної обробки, які можуть бути ефективно використані для прискорення обробки великих обсягів даних за допомогою персональних обчислювальних пристроїв під управлінням операційної системи iOS.

1.1.1 Паралельні обчислення

Технології розпаралелювання широко застосовуються для додатків, де обсяг даних та/або обчислень є надзвичайно великим. Для досягнення результатів у прийнятні строки необхідно, щоб кілька комп'ютерів одночасно виконували

обчислення різних частин однієї задачі. Протягом останніх десятиліть багато мов програмування, а також специфічних мережевих і комп'ютерних технологій використовувалися для вирішення наукових або статистичних проблем у державному та приватному секторах науки і промисловості. Спочатку паралелізм реалізовувався на рівні інструкцій через векторні блоки, а згодом – за допомогою мультипроцесорів зі спільною пам'яттю, що певною мірою ввело паралелізм у робочі станції. Це дозволило більш ширшому колу користувачів приймати участь у розробці паралельних програм та збільшило щільність процесорів у кластерах.

Під паралельними обчисленнями розуміють одночасне виконання комп’ютерної програми за допомогою декількох паралельних процесорів, що взаємодіють між собою [3]. Комп’ютерна програма, що виконується, поділяється на декілька процесорних блоків за допомогою методів паралельного програмування, часто – автономно, операційною системою та базовою архітектурою апаратної системи. Існує все більше застосувань, які потребують паралелізму обчислень для вирішення більших, складніших і все більш вимогливих обчислювальних завдань, що потребують швидкого, економічно ефективного та продуктивного розв’язання.

Модель паралельного програмування строго визначає її паралельну поведінку і функціональність для виконання основних операцій, таких як розподіл нових завдань, передача повідомлень і доступ до пам’яті, як ці структури впливають на обчислення, як вони можуть бути складені і коли вони можуть бути використані в обчисленнях.

Багатозадачність (multitasking) – це властивість операційної системи або середовища програмування забезпечувати можливість паралельної або псевдопаралельної обробки декількох процесів [4].

Послідовна багатозадачність передбачає формування черги, у відповідності до якої визначається порядок виконання завдань. Кожна задача формує власний потік виконання.

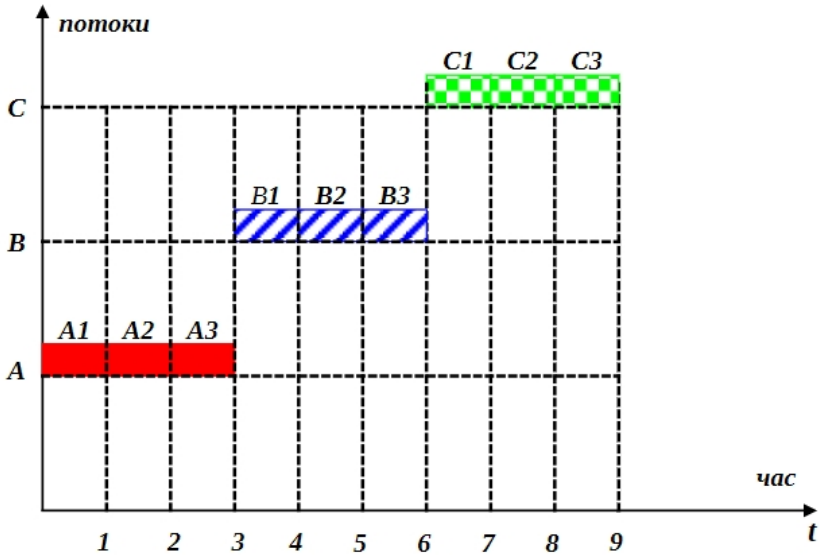

Рисунок 1.1 – Послідовне виконання (за даними [4])

У багатозадачних безпріоритетних операційних системах зазвичай застосовується пріоритетна (витісняюча) багатозадачність, при якій операційна система контролює передачу управління між процесами. Кожному процесу надається певний квант машинного часу, після завершення якого процес повинен звільнити процесор (див. рис. 1.2). Цей режим роботи часто називають паралельним потактним виконанням, коли всім процесам виділяється однаковий квант часу. Проте, за наявності лише одного фізичного процесора, жодна стратегія паралелізму не забезпечить прискорення виконання задач.

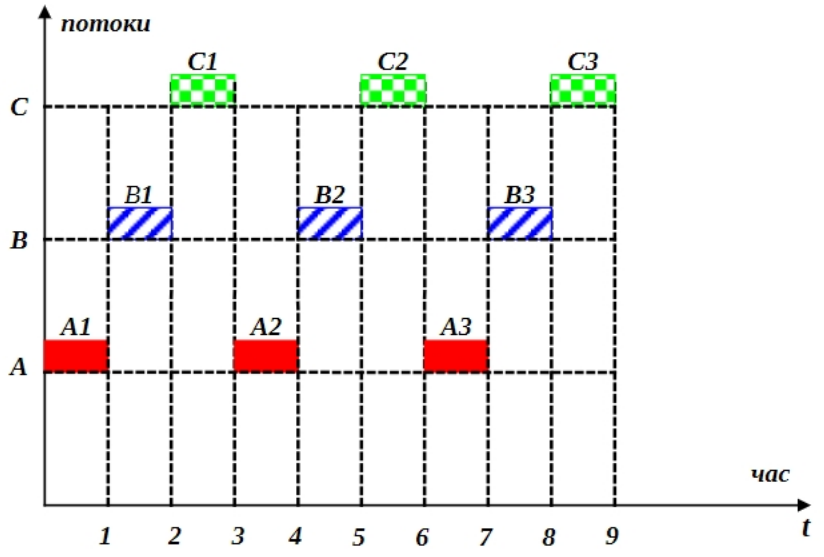

Рисунок 1.2 – Паралельне потактне виконання (за даними [4])

Коли процесор має декілька фізичних ядер, паралельна обробка називається багатопроцесорною обробкою на кристалі (Chip Multiprocessing, CMP). Багатоядерні процесори забезпечують справжню апаратну багатопотоковість. Кожне ядро виконує апаратні потоки незалежно від інших, тобто паралелізм досягається тим, що кожний потік обробляється окремим ядром (див. рис. 1.3). Обмін даними між потоками здійснюється через спільну пам’ять.

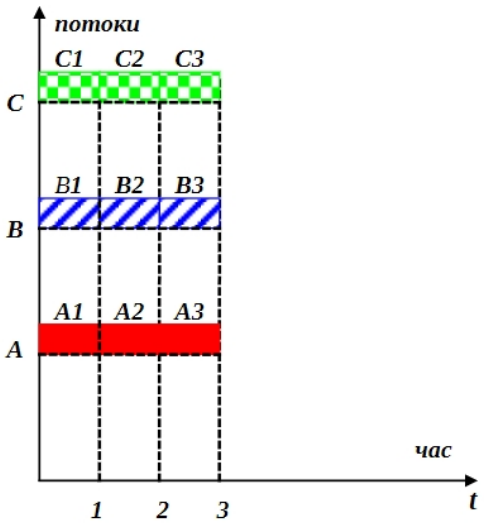

Рисунок 1.3 – Паралельне одночасне виконання (за даними [4])

Технології розпаралелювання широко використовуються в додатках, де обсяг даних або обчислень є дуже великим. Для отримання результатів за прийнятний час необхідно, щоб кілька комп'ютерів одночасно обчислювали різні частини однієї та тієї ж задачі. Багато мов програмування та дуже специфічні мережеві та комп'ютерні технології використовувалися для вирішення наукових або статистичних проблем протягом останніх десятиліть. Спочатку векторні блоки (які використовували паралелізм на рівні інструкцій), а пізніше – мультипроцесори зі спільною пам'яттю до певної міри впровадили паралелізм на робочих станціях, що дозволило широкій аудиторії створювати паралельні програми або збільшити щільність процесорів у кластерах, що призвело до появи стандартів OpenMP для багатопроцесорних систем із загальною пам'яттю та MPI для кластерів. Разом з тим майже одночасно виникли й технічні проблеми:

– висока вартість розробки та виробництва процесорів, складність зменшення їх розмірів;

– зростання вартості та складності, пов’язані зі зниженням рівня енергоспоживання при збільшенні кількості транзисторів на кристалах, призвело до підвищеного попиту на енергоефективність програмних засобів для мобільних пристроїв;

– підвищена вартість і складність обслуговування інтерфейсів передачі даних для підтримки продуктивність мобільних пристроїв на рівні з обчислювальними можливостями настільних процесорних систем.

Для зменшення негативного впливу цієї проблеми найпростішим рішенням на рівні апаратного дизайну є збільшення паралелізму на кількох рівнях архітектури, а також – підвищення модульності живлення процесорів. Це призводить до того, що багато елементів вмикаються та вимикаються в залежності від їх використання для збереження енергетичних ресурсів та у відповідності до теплових вимог.

Таке збільшення паралелізму на рівні процесора (за даними [5]) ускладнює їх використання:

– паралелізм сам по собі не є необхідністю для багатьох додатків, але він утворює додаткові складнощі для того, щоб повноцінно використовувати нові системи;

– в багатопроцесорних системах кеш-пам'ять, яка призначена для пом'якшення проблеми продуктивності інтерфейсу пам'яті, стала проблемою: чим більше процесорів – тим складнішою та об’ємною стає кеш-пам'ять, а механізм забезпечення когерентності пам’яті в кінцевому підсумку споживає ще більшу частину енергії та розігріває поверхню чіпу.

Сучасні персональні мобільні пристрої (PMD) – це енергоефективні комп’ютери з багатоядерними центральними процесорами (CPU) та потужними графічними прискорювачами (GPU), інтегрованим периферійним обладнанням, камерами, сенсорами і широкими можливостями бездротового зв’язку.

Середовища паралельних обчислень PMD мають багато спільних принципів з тими, що застосовуються у персональних ЕОМ та серверних системах.

Графічні процесори (GPU) були дуже специфічними процесорами, які мали набагато більше свободи для розвитку, ніж процесори загального призначення, такі як CPU, які повинні керувати операційною системою з усіма ускладненнями їх функціональних можливостей. Поява стандартів програмування для GPU, таких як OpenGL, дозволяють абстрагуватися від архітектури та отримати єдину модель графічного програмування для всіх GPU. Використання паралелізації GPU в основному полягає у програмованому паралелізмі даних, що дозволяє програмісту створювати власні візуальні ефекти, мати велику пропускну здатність пам'яті та використовувати деякі дуже поширені функціональні можливості, що можуть бути реалізовані за допомогою надшвидких фіксованих функціональних блоків. В результаті GPU стає відносно дешевою, дуже паралельною та швидкою архітектурою, яка після адаптації за допомогою відповідних мов програмування (CUDA та OpenCL) перетворила їх на паралельні процесори більш загального призначення (GPGPU). Тим не менш, графічні процесори все ще не є універсальними: вони добре підходять для окремих видів завдань, але дуже погано підходять для всіх видів використання послідовних кодів або виконання завдань паралелізму на рівні інструкцій (наприклад, для операційних систем) [6].

Платформи PMD підтримують достатньо багато типів паралелізму на апаратному та програмному рівнях. Для досягнення додаткової обчислювальної потужності, яку надає мобільна гетерогенна система, необхідно мати програмний доступ до графічного процесора пристрою. Апаратні програмні інтерфейси GPU, які знайшли своє застосування на поширених платформах PMD, таких як Android та iOS, в основному підтримують моделі програмування, що орієнтовані на використання графічних мов програмування.

Графічні процесори – це переважно набір векторних процесорів, що мають спільний простір пам'яті і кожен з яких має окремий блок керування. Ці векторні процесори мають різну кількість арифметико-логічних блоків (ALU) для цілочисельних операцій та операцій з плаваючою комою. Наприклад, процесор M2,

який використовується в PMD iPad Pro, має чотири високопродуктивних ядра «Avalanche» і чотири енергоефективні ядра «Blizzard», що забезпечує гібридну конфігурацію [7]. Високопродуктивні ядра мають надзвичайно великий кеш інструкцій L1 об'ємом 192 КБ і кеш даних L1 об'ємом 128 КБ, а також 16 МБ кеша L2; енергоефективні ядра мають кеш-пам'ять інструкцій L1 розміром 128 КБ, кеш-пам'ять даних L1 об'ємом 64 КБ і спільний кеш другого рівня об'ємом 4 МБ. Він також має 8 МБ кеш-пам'яті системного рівня, яка використовується графічним процесором. Apple M2 має десятиядерний графічний процесор, кожне ядро графічного процесора розділено на 32 функціональні блоки, кожен з яких містить вісім блоків ALU. Загалом графічний процесор M2 містить до 320 функціональних блоків або 2560 арифметико-логічних блоків [7]. Саме тому найбільш доречним при дослідженні методів паралельної обробки даних на платформі iOS є використання засобів розробки ПЗ, рекомендованих саме Apple.

AppleMetal [8] – це низькорівневий графічний та обчислювальний API для графічних процесорів пристроїв на платформах Apple macOS, iOS та tvOS. Він забезпечує використання високорівневої моделі програмування зі спільною пам’яттю. Так саме, як і OpenGL, він надає шаблони для програмування користувацьких програм на рівні ядра GPU та ефективний API для організації управління. Будучи інтегрованим до складу операційних систем Apple, Metal має можливість забезпечити детальний контроль над апаратним забезпеченням і має на меті забезпечити більш універсальні обчислення, ніж примітивні маніпуляції з графікою. Найбільш перспективним варіантом застосування методів паралелізації на платформі PMD під управлінням iOS є їх використання для обробки великих та надвеликих обсягів неструктурованої інформації, до яких доречно розглянути застосування методів обробки, що розроблялись для баз даних.

1.1.2 Технології обробки великих даних (BigData)

Поняття BigData з’явилось на початку XXI-го сторіччя як альтернатива традиційних систем управління базами даних. BigData на теперішній час використовується для позначення структурованих та неструктурованих даних, які

є значними за обсягом, і різного типу, та масштабуються горизонтально [9]. Разом з тим слід зазначити, що великі дані – це також сукупність технологій, орієнтованих на виконання наступних операцій:

– обробка більших, у порівнянні зі «стандартними» сценаріями, об’ємів даних;

– вміння опрацьовувати данні, що швидко надходять у дуже великих обсягах;

– вміння працювати зі структурованими та мало структурованими даними паралельно і у різних аспектах їх використання.

Термін BigData, який вперше у 2008 році використав К. Лінч [9], описує масиви інформації різної природи з обсягом приросту, що перевищує 150 Гб на добу. На сьогоднішній день відсутні чіткі критерії для визначення обсягів даних, які можна вважати «великими», проте зазначений поріг залишається основним засобом ідентифікації таких масивів. У загальному розумінні, термін «великі дані» пов’язується з великими об’ємами інформації, які через свій розмір стають недоступними для обробки користувачами та потребують використання цифрових методів і інструментів. Ці дані вирізняються швидкістю формування, нагромадження і обробки.

У практичному сенсі архітектуру даних можна розглядати як модель їх збору, зберігання та обробки. Вибір архітектурної моделі залежить від основного призначення інформаційної системи та контексту її застосування, а також доступних на даний момент технологій. Оскільки щодня на ринку програмного забезпечення з'являються нові рішення, а бізнес стикається з новими викликами, архітектури даних також розвиваються.

Поширення технологій BigData значною мірою стимулюють економічне зростання багатьох галузей, включаючи Інтернет-індустрію та медицину, а також традиційні галузі промисловості. Наразі типовим підходом до обробки BigData вважається використання фреймворку Hadoop та похідних від нього. Технічно архітектуру Hadoop, яку в основному підтримують Yahoo! та Facebook, доповнює та оптимізує MapReduce [10]. З моменту своєї появи у 2006 році архітектура Hadoop перетворилася на динамічну екосистему, що містить понад шістдесят компонентів.

Крім того, екосистема значно сприяла розвитку досліджень розподілених обчислень, хмарних обчислень та пов’язаних з ними робіт.

За останнє десятиріччя спостерігається величезний прогрес та інновації у системах обробки великих обсягів даних та пов'язаних з ними обчислень на основі даних. Серед багатьох інших вони включають обчислювальні системи на основі MapReduce, технології потокового передавання даних та системи баз даних NoSQL. Основним викликом стало створення систем, які, з одного боку, могли б обробляти великі обсяги пакетних даних, а з іншого – пропонувати необхідну масштабованість, продуктивність та низькі затримки, необхідні для інтеграції та обробки в реальному часі величезних безперервних потоків даних.

1.1.3 Архітектури BigData

Типовий робочий процес, що застосовується для обробки великих обсягів неструктурованих даних, містить численні запити за змістом шуканої інформації та її часових міток, просторові/часові агрегації їх розташування та узагальнення. Існуючі рішення здебільш зосереджуються на часі виконання, масштабованості та пропускній здатності, які мають велике значення для впровадження та роботи в режимі, наближеному до режиму реального часу [11]. У загальному розумінні під архітектурою BigData розуміють модель, яка включає методи, правила та способи опису поточного стану даних, що необхідні для формування вимог до даних, їх інтеграції та контролю використання відповідно до загальної стратегії управління. З цією метою розглянемо основні типи цих архітектурних підходів.

Першими серед архітектурних моделей даних стали системи класу Business Intelligence (BI), які вперше з'явилися у 80-х роках. XX ст. На сьогодні ці системи не можуть охопити всі поточні бізнес-сценарії роботи з великими обсягами даних, оскільки BI-системи більше орієнтовані на аналіз структурованих бізнес-даних з високою щільністю, але не орієнтовані на підтримку обробки неструктурованих та напівструктурованих даних типу зображень, тексту та аудіо тощо. Крім того, традиційна архітектура призначена для пакетної обробки даних і не підтримує потокову передачу та обробку подій у реальному часі [11].

Порівняно з традиційною архітектурою, Streaming-модель орієнтована на потокову обробку, а пакетна частина видалена: дані обробляються протягом усього процесу. Дані безпосередньо передаються споживачам в якості повідомлень, але відсутня будь-яка історична ретроспектива.

Однак у чистій потоковій архітектурі відсутня пакетна обробка, тому відтворення даних та історична статистика не підтримуються належним чином. Це стане в нагоді лише для малого числа реальних бізнес-сценаріїв, наприклад, раннє попередження, різні аспекти моніторингу та вимог до терміну дії даних.

Системи великих даних часто зіштовхуються з проблемою, як інтегрувати обробку «нових» даних, які постійно надходять в систему, з історичними (пакетними) даними. Нові дані, що надходять в режимі реального часу, зазвичай обробляються за допомогою потокових методів обробки, в той час як історичні дані періодично переробляються за допомогою пакетної обробки. Архітектура Лямбда – це схема системи великих даних, яка поєднує в собі потокову обробку даних у реальному часі та пакетну обробку історичних даних [12].

Лямбда-архітектура пропонує загальний підхід до розробки систем великих даних з метою подолання складнощів та обмежень, що виникають при спробі масштабування традиційних систем даних, що базуються на інкрементно оновлюваних реляційних базах даних. У системах інкрементних баз даних стан бази даних (тобто її вміст) поетапно оновлюється під час обробки нових даних. На відміну від таких систем, лямбда-архітектура використовує функціональний підхід, який ґрунтується на незмінних даних, тобто нові дані додаються поверх історичних пакетних даних, які вже містяться в системі.

На відміну від традиційних розподілених систем баз даних, де розподіл таблиць між декількома машинами повинен бути визначений розробником, ключовим принципом архітектури Лямбда є те, що сама система усвідомлює свою розподілену природу, щоб автоматично керувати розподілом, реплікацією та пов'язаними з цим аспектами. Ще одним ключовим аспектом Лямбда-архітектури є її підтримка незмінних даних, на відміну від даних, які поступово оновлюються в реляційних системах баз даних. Використання незмінних даних має важливе

значення для забезпечення стійкості до людських помилок. На високому рівні абстракції архітектура Лямбда складається з трьох шарів: пакетного, обслуговуючого та швидкісного.

На рисунку (див. рис.1.4) зображено основні архітектурні аспекти лямбда-архітектури. Дані, що надходять з джерел даних (датчиків, вебклієнтів тощо), паралельно подаються як на пакетний рівень, так і на швидкісний потоковий рівень, які обчислюють відповідні пакетні представлення та представлення в реальному часі відповідно. Лямбда-архітектуру можна розглядати як компроміс між двома протилежними цілями: швидкістю та точністю. Під час обчислення в реальному часі відбувається з мінімальними затримками, у той час як обчислення пакетних даних, як правило, вимагає значних затримок. З іншого боку, оскільки швидкісний рівень не ураховує всі наявні дані, подання в реальному часі зазвичай є лише наближеними, у той час як пакетні подання надають точні результати, враховуючи всі дані, доступні в сховищі основних даних на певний момент часу [13].

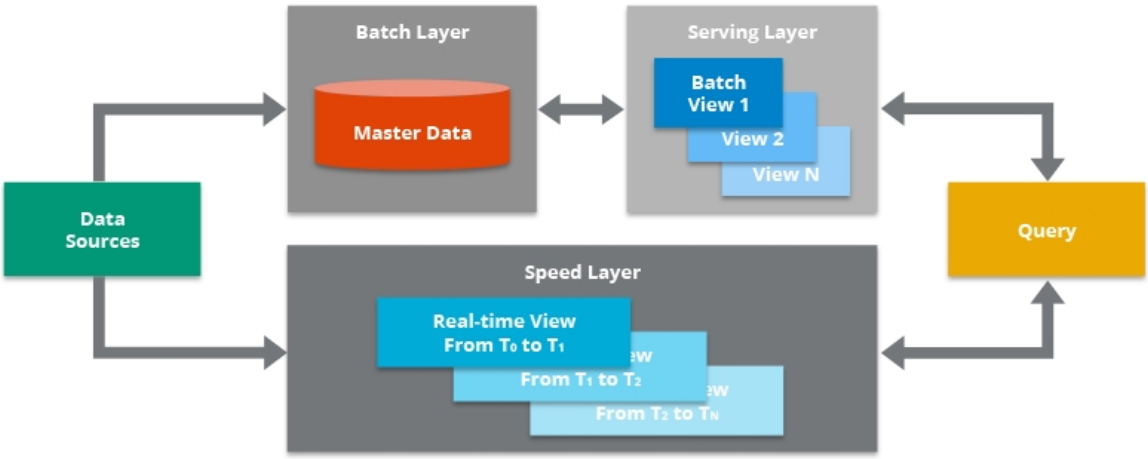

Рисунок 1.4 – Лямбда-архітектура (за даними [13])

Лямбда-архітектура отримала широке визнання як життєздатний підхід до поєднання пакетної та потокової обробки, оскільки вона підтримує потокову обробку в реальному часі та повторну обробку пакетів на незмінних даних. Однак у лямбда-архітектури є деякі потенційні недоліки, пов'язані з її застосуванням. Хоча основною метою лямбда-архітектури є зменшення складності порівняно з традиційними розподіленими системами баз даних, ця мета часто не може бути

повністю реалізована. У той же час, пакетний рівень зазвичай приховує складність від розробників, як правило, покладаючись на деякі високорівневі фреймворки MapReduce (наприклад, Hadoop), але швидкісний рівень все ще може демонструвати значні складнощі для розробників рішень для великих даних. Крім того, необхідність розробляти і підтримувати два окремі компоненти обробки даних, потоковий і пакетний рівні, додає загальної складності [14]. Ще однією потенційною проблемою лямбда-архітектури є те, що постійний перерахунок пакетних представлень з нуля може стати непомірно дорогим з точки зору використання ресурсів і затримок.

Каппа-архітектура (див.рис. 1.5) є спрощенням лямбда-архітектури шляхом рівного ставлення до даних в реальному часі та пакетних даних як до потоків. Замість пакетної обробки, як це робиться в лямбда-архітектурі, в Каппа-архітектурі використовується потокова обробка. Каппа-архітектура передбачає, що історичні пакетні дані також можна розглядати як обмежений потік, що часто і відбувається. Однак, для того щоб компонент обробки потоку міг ефективно відтворювати історичні дані у вигляді потоку, потрібно забезпечити ефективний механізм відтворення. Тільки в такому випадку пакетні дані можуть бути переобчислені тим самим механізмом потокової аналітики, який також відповідає за обробку даних у реальному часі. Крім можливості відтворення історичних даних, для забезпечення детермінованості результатів у системі має бути строго збережений порядок всіх подій даних [15].

Замість використання пакетного та швидкісного шарів, архітектура Каппа базується на одному потоковому шарі, який може обробляти обсяги даних як у реальному часі, так і у вигляді пакетних представлень. Використання Каппа-архітектури призводить до зменшення загальної складності системи, як це показано на рисунку.

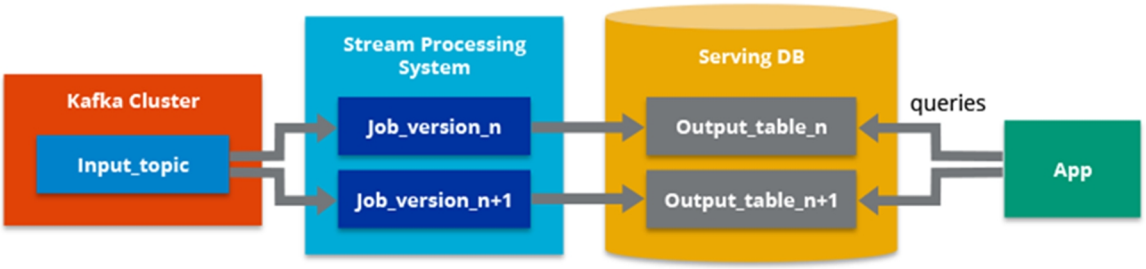

Рисунок 1.5 – Каппа-архітектура (за даними [15])

Слід зауважити, що Каппа-архітектура не є повноцінною заміною для Лямбда-архітектури, оскільки вона не підходить для всіх можливих сценаріїв використання.

Всі розглянуті вище архітектури у своїй більшості орієнтовані на масову обробку даних без їх аналізу. Архітектура Unifield використовує елементи машинного навчання. В її основі лежить Лямбда-архітектура, але налаштована на потокову обробку. Для цього окремо винесений шар Machine Learning, який у центрах обробки даних та Data Lake через канал даних додає нову навчальну частину моделі, яка використовується на рівні потокової передачі. Цей потоковий рівень використовує ML модель та постійно навчає її. Фактично Unifield надає цілий набір архітектурних рішень, що поєднують аналіз даних та машинне навчання. Однак архітектуру Unifield дуже складно реалізувати через проблеми розробки та розгортання ML-систем. Проте така архітектура є дуже ефективною для проектів, де необхідно проаналізувати великий обсяг даних за допомогою алгоритмів машинного навчання.

Архітектура мобільного додатку, орієнтовного на паралельну обробку, суттєвим чином залежить від архітектури BigData та принципів організації доступу до даних. Тому подальші дослідження будуть спрямовані на визначенні певного архітектурного підходу при побудові програмних додатків, які будуть забезпечувати доступ до BigData за рахунок організації паралельного спрямування запитів до неструктурованої інформації за допомогою персональних мобільних пристроїв під управлінням iOS.

1.2 Постановка задачі дослідження

Метою роботи є дослідження шляхів підвищення продуктивності iOS-додатків за рахунок використання методів паралельної обробки. Для досягнення мети дослідження в роботі необхідно:

– дослідити підходи до організації паралельної обробки даних на платформі iOS, виконати порівняльний аналіз щодо їх застосування для обробки великих обсягів неструктурованої інформації;

– визначити інструментальні засоби та фреймворки, що дозволяють розробити програмні продукти, призначені для паралельної обробки даних на платформі iOS;

– визначити предметну область, в рамках якої будуть проводитись дослідження щодо паралельної обробки великих обсягів неструктурованої інформації;

– побудувати модель програмної системи, яка дозволяє організувати паралельну обробку великих обсягів неструктурованої інформації в певній предметній області;

– виконати програмну реалізацію моделі програмної системи для дослідження різних підходів до організації паралельної обробки великих обсягів неструктурованої інформації в певній предметній області;

– провести експерименти щодо визначення ефективності кожного із запропонованих підходів до організації паралельної обробки великих обсягів неструктурованої інформації, зробити висновки щодо результатів дослідження.

На підставі визначених задач загальна постановка задачі дослідження буде мати наступний вигляд:

– провести аналіз методів паралельної обробки даних

– проаналізувати підходи паралелізації обчислювальних процесів, що можуть бути використані для обробки великих обсягів неструктурованої інформації;

– визначитись з предметною галуззю та побудувати модель програмного додатку для персональних мобільних пристроїв під управлінням iOS;

– виконати програмну реалізацію системи та провести дослідження щодо порівняння ефективності запропонованих методів паралелізації при обробці великих обсягів даних;

– зробити висновки щодо отриманих результатів проведених досліджень.

Дослідження будуть проводитись за допомогою програмного додатку, спроектованого в рамках роботи над проектом Nibble: Your Bite of Knowledge (https://apps.apple.com/us/app/nibble-your-bite-of-knowledge/id6444046612). Це освітній продукт із короткими інтерактивними уроками з різних тем: від математики до мистецтва [16], які використовують метод надання рекомендацій для вибору та формування змісту цих уроків. Рекомендації надаються на основі моделі колаборативної фільтрації, за допомогою якої система використовує матрицю взаємодій, де кожен рядок представляє користувача, а кожен стовпчик — елемент. Відповідно, значення в матриці вказують на взаємодії між користувачами та елементами (див. рис.1.3).

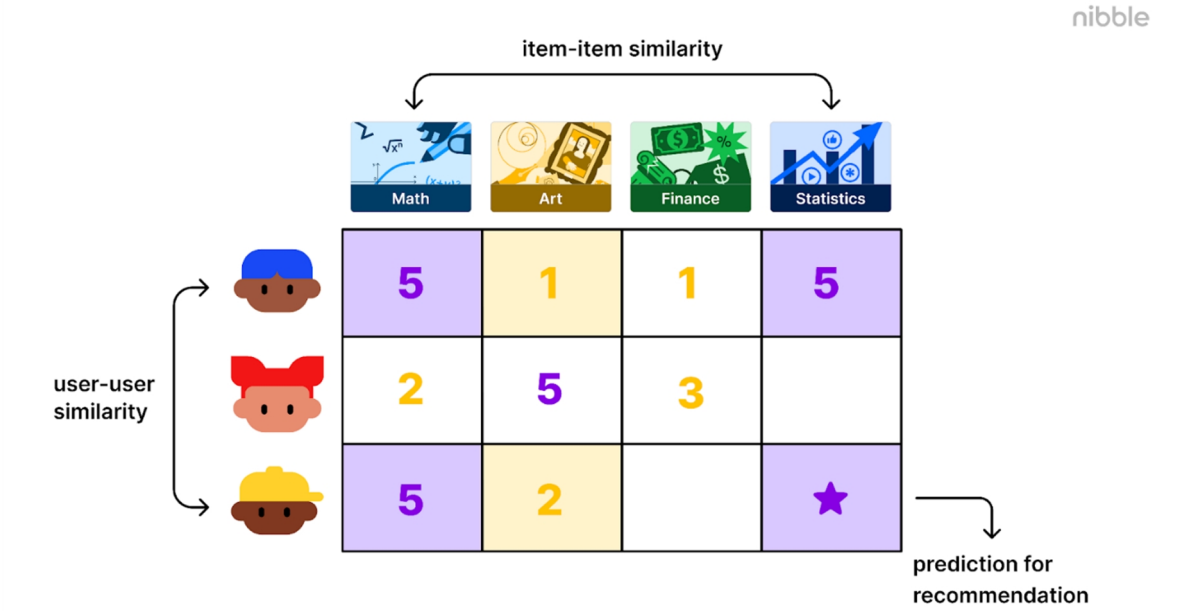

Рисунок 2.1 – Матриця взаємодій (за даними [16])

Однією з ключових особливостей програмної системи Nibble є динамічна секція «For you», як за допомогою надання рекомендацій у предметній області дозволяє зробити аналітику уроків більш варіативною, підвищити відповідність

конкретним потребам користувачів та підняти індивідуалізацію за рахунок запровадження власного підходу до вивчення матеріалу. Саме формування рекомендацій за рахунок виконання запитів до зовнішніх сховищ інформації було взято за основу для проведення досліджень в рамках підготовки кваліфікаційної роботи магістра, присвяченої порівнянню ефективності методів розпаралелювання на платформі iOS. Проведення досліджень передбачає використання методів розпаралелювання при формуванні та обробці запитів до корпоративного хмарного сховища, побудованого за технологією Google Cloud Bigtable, з метою майбутнього впровадження отриманих результатів досліджень у проект Nibble для покращення його продуктивності.

2 ОПИС ПРОВЕДЕНИХ ТЕОРЕТИЧНИХ ТА ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

2.1 Визначення предметної галузі для проведення досліджень

Системи рекомендацій виконують важливу роль у споживчому та професійному досвіді користувачів, інтегруючись ефективно у функціонал багатьох продуктів і роблячи їх лідерами у відповідних галузях. Наприклад, Spotify, один з найпопулярніших стрімінгових сервісів аудіо, налічує понад 500 мільйонів користувачів. Близько третини з них відкривають для себе нових виконавців завдяки функції «Зроблено для вас». Цей успіх свідчить про ефективне поєднання обширної бази контенту і вдосконалених систем рекомендацій. Основною перевагою використання систем рекомендацій слід вважати їх здатність аналізувати величезний обсяг інформації та створювати персоналізований досвід для користувачів. Вони враховують різноманітні аспекти, включаючи особисті вподобання, соціальні взаємодії користувачів і динамічні зміни у контенті.

Як вже було вказано вище, дослідження будуть проводитись в рамках роботи над проектом Nibble: Your Bite of Knowledge [17]. Додаток, який отримав міжнародне визнання, використовує метод надання рекомендацій для вибору та формування змісту індивідуальних уроків. Розглянемо методи, які використовуються для формування системи рекомендацій.

Існують два основні типи систем рекомендацій: на основі вмісту та з використанням колаборативної фільтрації.

Системи рекомендацій на основі вмісту (Content-basedfiltering) базуються на спільних атрибутах елементів контенту, які сподобалися користувачеві раніше. Вони позначають кожен елемент або користувача певними характеристиками, а потім вивчають та роблять відповідні припущення щодо їхньої схожості.

Основна ідея полягає в тому, щоб рекомендувати користувачеві елементи на основі спільних атрибутів або ознак тих, які він уже вподобав. Наприклад, для рекомендації фільмів можуть використовуватися такі атрибути, як жанр, режисер,

актори, рік випуску та інші. Всі вони допомагають системі зрозуміти, які елементи можуть бути подібні за контентом.

Такий тип рекомендаційних систем особливо ефективний у галузях, де важливий вміст або якісні характеристики елементів. Він дозволяє забезпечити користувачів релевантним та персоналізованим контентом, спираючись на їхні смаки та уподобання.

Однак системи на основі вмісту також містять ряд недоліків. Наприклад, вони обмежені в тому, що враховують тільки контент та не беруть до уваги аналіз взаємодії користувачів. Також вони можуть мати труднощі з рекомендацією нового контенту, оскільки оперують інформацією на основі наявних атрибутів. Для роботи із цими запитами слід звернути увагу на системи на основі колаборативної фільтрації.

Системи на основі колаборативної (або спільної) фільтрації (Collaborativefiltering) – це тип рекомендаційних систем, який базується на аналізі фактичної взаємодії користувачів з певними елементами вмісту [18]. Система може рекомендувати контент на основі ітерацій з такими елементами як оцінки, відгуки чи покупки.

Колаборативний фільтр Item-Item аналізує схожість між елементами контенту, щоб рекомендувати користувачеві елементи на основі його попередніх вподобань. Якщо користувач взаємодіяв позитивно з певним елементом, то система рекомендує інші схожі елементи, які також можуть йому сподобатися.

Колаборативний фільтр User-Item аналізує взаємодії між користувачами та конкретними елементами контенту. Якщо два користувачі мають схожі вподобання контенту, система рекомендує одному користувачеві елементи, які інший користувач оцінив позитивно, і навпаки.

Узагальнюючи можна сказати, що фільтр Item-Item аналізує схожість контентних елементів, тоді як фільтр User-Item аналізує ітерації, спираючись на схожі користувацькі уподобання.

Найбільшу продуктивність демонструють рекомендаційні системи, що побудовані саме на основі колаборативної фільтрації. Такі системи дозволяють

якісно та ефективно персоналізувати вміст та значно підвищити залученість користувачів в застосунку, що й обумовило їх використання в проєкті Nibble.

2.2 Математична модель організації обробки даних

Загалом будь-які дані, що можуть розглядатись як джерело BigData, мають щонайменше два елементи: самі дані та їх характеристики. Формально поділимо ці об’єкти на категорії:

– сутності $\boldsymbol{e}$;

– характеристики $\boldsymbol{f}$;

– асоціації між сутностями $\boldsymbol{e}$ та характеристиками $\boldsymbol{f}$.

Наприклад:

– назва $\boldsymbol{e}$ згадується у документі $\boldsymbol{f}$;

– термін $\boldsymbol{f}$ з’явився у документі $\boldsymbol{e}$.

Таким чином також можна визначити:

– множину сутностей $\boldsymbol{E}$;

– множину характеристик $\boldsymbol{F}$;

– для кожних $\boldsymbol{e}$ та $\boldsymbol{f}$ визначено номер асоціацій між $\boldsymbol{e}$ та $\boldsymbol{f}$ як $\boldsymbol{n}_{e,f}$.

Визначимо загальну кількість сутностей через $|\boldsymbol{E}|$; тоді кількість характеристик можемо визначити як потужність множини $\boldsymbol{F}$: $|\boldsymbol{F}|$. Відповідно до обраних припущень можна отримати:

– для кожної характеристики $\boldsymbol{f}$ – множину $\boldsymbol{e}(\boldsymbol{f}) = \{\boldsymbol{e} \in \boldsymbol{E}: \boldsymbol{n}_{\boldsymbol{e},\boldsymbol{f}} > \boldsymbol{0}\}$ для усіх асоційованих з $\boldsymbol{f}$ сутностей;

– для кожної сутності $\boldsymbol{e}$ – множину $\boldsymbol{f}(\boldsymbol{e}) = \{\boldsymbol{f} \in \boldsymbol{E} : \boldsymbol{n}_{\boldsymbol{e},\boldsymbol{f}} > \boldsymbol{0}\}$ для усіх асоційованих з $\boldsymbol{e}$ характеристик.

Наведена модель орієнтовна на врахування якісних характеристики, що відносяться до можливої структури даних, а кількісні показники можуть мати певні відмінності [19].

В тому випадку, коли присутні декілька сутностей, пов'язаних з однією характеристикою об’єкту, необхідно визначити кількість бінарних запитів, які необхідно виконати, щоб визначити необхідний об'єкт. Якщо відомо, що деякий

об'єкт належить множині, яка складається з N елементів, то існує можливість поділити цей набір на дві рівні частини (принцип алгоритму бінарного пошуку в масиві). Таким чином, після формування та отримання відповіді на q бінарних запитів отримаємо множину з $N \cdot 2^{-q}$ елементів, яка і буде містити необхідний об'єкт, а кількість запитів, на які треба відповісти для того, щоб отримати шуканий об'єкт, буде становити $q = log_2(N)$.

Такий саме спосіб можна використати і для сутностей. Існує E сутностей, що містять визначену кількість інформації: $\log_2(E)$. Якщо відомо, що будь-яка сутність асоційована з певною характеристикою (існує $e(f)$ сутностей), то кількість інформації буде визначена виразом: $\log_2(|e(f)|)$.Тоді той факт, що сутність пов'язана з характеристикою f, дає змогу зменшити кількість бінарних запитів до:

$$\log_2(|E|) - \log_2(|e(f)|) = \log_2\left(\frac{|E|}{|e(f)|}\right).$$

Кількість асоціацій також можна визначити за допомогою бінарних запитів, які необхідно сформувати для того, щоб і надалі асоціація з необхідною сутністю була відома. Кожний бінарний запит для $n_{e,f}$ зменшує кількість цих об'єктів вдвічі, а формування q запитів зменшує цю кількість до $n_{e,f} \cdot 2^{-q}$. Асоціація буде існувати до того часу, поки кількість об'єктів буде більшою за 1. Тоді найбільша кількість запитів q , для якої ще існує асоціація, можна визначити як $N \cdot 2^{-q} = 1$, що, у свою чергу, можна визначити як $q = \log_2(n_{e,f})$. Формування будь-якого додаткового запиту буде визначатись як $1 + \log_2(n_{e,f})$.

На підставі викладеного загальна важливість характеристики f для сутності e будемо визначати як:

$$\log_2\left(\frac{|E|}{|e(f)|}\right)$$

з фактором важливості $1 + \log_2(n_{e,f})$. Враховуючи це, результуючу кількість

інформації можна представити виразом наступного виду:

$$I(e,f) = (1 + \log_2(n_{e,f})) * \log_2\left(\frac{|E|}{|e(f)|}\right) \qquad (1)$$

Формула (1) є одним з варіантів визначення зворотної частоти документа **tf-idf**, в якій для кожної сутності ***e*** існує важливість ***I (e, f)*** у відповідності до різних характеристик ***f*** .

Виконаємо нормалізацію значення важливості:

$$V(e,f) = \frac{(1+\log_2(n_{e,f})) * \log_2\left(\frac{|E|}{|e(f)|}\right)}{\sqrt{\sum\left((1+\log_2(n_{e,f})) * \log_2\left(\frac{|E|}{|e(f)|}\right)\right)^2}}. \qquad (2)$$

Кожна із сутностей ***e*** має вагу ***V (e, f)***, тому мірою наближення об'єкту E_iдо об'єкту E_nслід вважати відстань між відповідними векторами $V(e, f_i)$ та$V(e, f_n)$.

Для кожної ваги ***V (e, f)***, що репрезентує визначену кількість бітів, відстань між сутностями e_1 та e_2 буде обчислюватись у відповідності до:

$$d(e_1, e_2) = \sum_f \left|V(e_1, f) - V(e, f_2)\right|. \qquad (3)$$

Ця відстань залежить від кількості характеристик: якщо, наприклад, крім самих документів ми зберігаємо ще їх копії, то відстань збільшується вдвічі. Для подолання цієї залежності необхідно виконати нормалізацію відстані $d(e_1,e_2)$ в інтервалі[0,1] шляхом її ділення на максимально можливе значення цієї відстані [2].

В умовах невизначеності, коли значення ***A*** та ***B*** невідомі, а є лише верхні межі цих величин $\overline{a}$ та $\overline{b}$, то найбільша можлива різниця буде становити $\mathbf{m}\left(\overline{a}, \overline{b}\right)$, а саме:

– якщо $\overline{a} \le \overline{b}$, то $|\overline{a} - \overline{b}| = \overline{b} - \overline{a} \le \overline{b}$та $|\overline{a} - \overline{b}| \le \max(\overline{a}, \overline{b})$;

– якщо $\overline{b} \le \overline{a}$, то $|\overline{a} - \overline{b}| = \overline{a} - \overline{b} \le \overline{a}$та $|\overline{a} - \overline{b}| \le \max(\overline{a}, \overline{b})$,

тобто в обох випадках виконується $|\overline{a} - \overline{b}| \leq \max(\overline{a}, \overline{b})$.

Межа $\mathbf{m}(\overline{a}, \overline{b})$ досягається у двох випадках:

– якщо $\overline{a} \leq \overline{b}$, то при $a = 0$, $b = \overline{b}$;

– якщо $\overline{b} \leq \overline{a}$, то при $a = \overline{a}$, $b = 0$.

Наведена модель обробки великих обсягів даних буде використана для побудови та подальшого дослідження методів паралельної обробки запитів до великих обсягів неструктурованої інформації з використанням персональних мобільних пристроїв під управлінням операційної системи iOS. Перейдемо до розгляду математичної моделі для визначення асоціацій між елементами БД.

2.3 Моделі асоціацій між сутностями та характеристиками

Основним елементом будь-якої БД слід вважати ключ. Носієм даних в моделі NoSQL БД є кортеж:

$$K = \{f, e\},$$

де f – ключ, який приймає унікальне значення для кожної пари; e – значення, яке йому відповідає.

Сигнатура моделі має наступний вигляд:

$$O = \langle \pi, \sigma \rangle,$$

де π – операція проекції за атрибутами (ключ або значення), σ – селекції атрибутів (вибір значення за ключем, ключів за значеннями, наслідування ключів). Ці операції відносяться до категорії читання [11].

Одним з варіантів реалізації розподіленого зберігання великих обсягів даних є система BigTable від компанії Google, яка має наступні властивості:

– неповна реляційна модель даних;

– підтримка динамічного контролю над розміщенням даних.

Основу моделі збереження даних в BigTable складають рядки, стовпці та

тимчасові мітки:

$$\boldsymbol{BigTable} = \{<\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{t}>\}.$$

Якщо в декількох стовпцях зберігаються дані, що відносяться до одного типу, то такі стовпці, згідно з моделлю Bigtable, утворюють сімейство:

$$\boldsymbol{col\ F} = \{ \boldsymbol{c_i}, \boldsymbol{c_j} \mid \text{dom}\ (\boldsymbol{c_i}) \subseteq \boldsymbol{T} \text{dom}\ (\boldsymbol{c_j}) \subseteq \boldsymbol{T}\}.$$

Використовувати сімейство стовпців досить зручно хоча б з того міркування, що це дозволяє стиснути однорідні дані, тим самим зменшивши їх обсяг. Саме сімейства стовпців складає одиницю доступу до даних.

Рядки BigTable (їх максимальна довжина може досягати 64 кілобайти) теж важливі. Операція звернення до рядка є атомарною (це означає, що поки одна програма звертається до рядка, жодна інша не має права змінювати дані в сімействах стовпців цього рядка).

Вміст інформації, що супроводжує історію хвобори пацієнта, постійно змінюється. Щоб врахувати ці зміни, кожна з копій даних, що зберігаються в стовпці, отримують тимчасову мітка (timestamp). У BigTable в якості тимчасової мітки використовується 64-розрядне число, яким можна кодувати час і дату таким чином, як це потрібно клієнтським програмам. Шляхом використання тимчасові мітки, додатки можуть виконувати в BigTable пошук, наприклад, тільки найновіших копій даних [20].

Отже, для будь-якої предметної області в сервісі Google можна створити власну карту даних Bigtable, що містить задану кількість рядків і унікальний для цієї предметної області набір сімейств стовпців. Повтори даних у стовпцях упорядковуються за значеннями тимчасових міток.

Головною перевагою цього підходу є те, що таку базу неважко поділити на незалежні елементи та розподілити по множині серверів. Відсортовані за алфавітом рядки діляться на діапазони, що мають назву tablet. Оскільки рядки в

кожному таблеті відсортовані за ключовим іменем, то клієнтським додаткам достатньо просто знайти потрібний таблет, а в ньому – необхідний рядок [21].

Для синхронізації таблетів призначений сервіс Chubby. Для кожного таблет-сервера Chubby створює спеціальний chubby-файл, за рахунок чого файл Bigtable може визначити, які із серверів є працездатними. Ще один chubby-файл містить посилання на розташування кореневого таблета (Root-tablet) з даними про розташування усіх інших. Цей файл повідомляє майстру, який з серверів якими таблетами керує.

Використання сервісу Chubby в Bigtable дозволяє забезпечити підтримку несуперечності даних у розподіленому середовищі з безліччю реплік. Bigtable стала першою спробою досягти балансу між продуктивністю системи, її масштабованістю та непротиріччям даних, що тут зберігаються [21]. Результатом стала підтримка так званої слабкої несуперечності, яка, в принципі, задовольняє вимоги більшості працюючих з Bigtable сервісів.

Розглянуті математичні моделі було використано при проєктуванні програмної системи, призначеної для проведення досліджень в рамках підготовки кваліфікаційної роботи магістра, присвяченої дослідженню методів паралельної обробки великих обсягів неструктурованої інформації на PMD під управлінням iOS.

2.4 Особливості організації обробки потоків в iOS

Архітектура iOS є багатошаровою. На найвищому рівні iOS діє як посередник між базовим апаратним забезпеченням та додатками, які не взаємодіють безпосередньо з базовим апаратним забезпеченням.

Додатки взаємодіють з апаратним забезпеченням за допомогою набору чітко визначених системних інтерфейсів (див.рис.2.2). Нижні шари надають основні послуги, на яких ґрунтуються всі додатки, а верхній рівень забезпечує високорівневу графіку та сервіси, пов'язані з інтерфейсом [6].

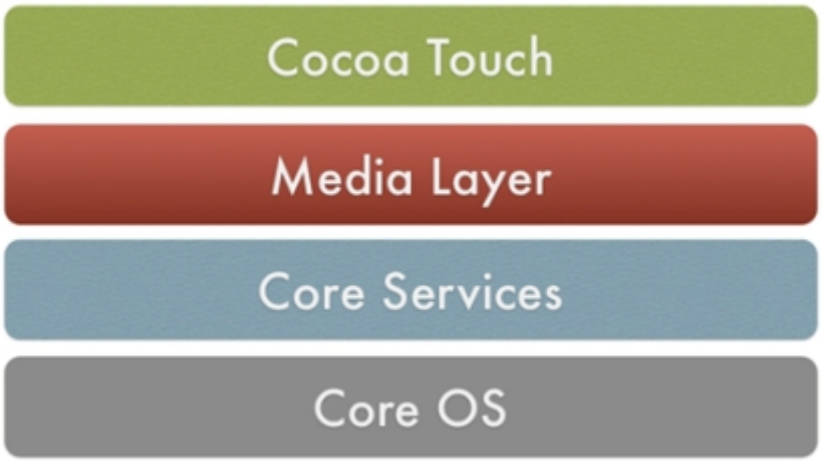

Рисунок 2.2 – Рівні iOS (рисунок виконано самостійно)

Рівень ядра операційної системи (Core OS) відповідає за взаємодію з апаратним забезпеченням пристрою, таким як гіроскоп, Bluetooth та зовнішні аксесуари. Цей шар також забезпечує керування живленням, безпеку та багато інших низькорівневих програмних функцій.

Служби ядра (Core Service) відповідають за організацію мережевих операцій, автоматичний підрахунок посилань, операції з рядками та інші функції форматування даних. Вони також забезпечують доступ до файлів і функціональні можливості багатопоточності для програмістів.

Медіашар (Media Layer) відповідає за відтворення медіа, а також – за запис і редагування аудіовізуальних матеріалів. Також він забезпечує рендеринг 2D та 3D графіки і анімацію.

Cocoa Touch є шаром, у якому відбувається більшість взаємодії з користувачем. Він виконує обробку жестів, дотиків, загального візуального аспекту застосунку та його поведінку.

Apple забезпечує доступ до більшості своїх системних інтерфейсів за допомогою фреймворків. Фреймворк – це каталог, який містить динамічну загальнодоступну бібліотеку (.files), пов'язані ресурси і допоміжні програми, необхідні для підтримки цієї бібліотеки. Кожен шар має власний набір фреймворків, якими користується розробник при розробці додатків.

У розробці додатків для iOS термін «потік» відноситься до одиниці

виконання в межах процесу. Процес може мати декілька потоків, кожен з яких може виконувати код паралельно із іншими потоками у тому ж самому процесі.

Оскільки додатки для iOS мають обмежені ресурси, то створення значної кількості потоків може призвести до проблем з продуктивністю або навіть до збоїв. Тому важливо використовувати багатопотоковість в iOS, враховуючи компроміс між паралельністю та використанням ресурсів.

Apple використовує різні типи процесорів для своїх мобільних телефонів та для планшетів: сучасні моделі Apple iPhone будуються на процесорах Bionic, а Apple iPad Pro, починаючи з 2022 року, використовують родину процесорів M.

Сучасні моделі багатоядерних процесорів Apple мають у своєму складі як високопродуктивні ядра (P-ядра), так і високоефективні (E-ядра). Ці різні типи ядер дозволяють створювати програми, які мають як високу продуктивність, так і забезпечують тривалий час автономної роботи.

Процесори родини Apple Bionic мають конфігурацію 6-ядерного процесора з 2 високопродуктивними ядрами та 4 енергоефективними ядрами, які можуть забезпечити продуктивність на будь-якому рівні потужності [22]. Ядра одного типу групуються в кластери. У межах кожного кластера всі ядра працюють на одній частоті, і вони використовують спільну локальну пам'ять у своєму кеші рівня 2 (L2). Це також спрощує переміщення потоків між ядрами в межах одного кластера. У всіх версіях процесорів є один кластер з чотирьох E-ядер і один кластер з двох P-ядер, тому процесор може бути позначений як 2P+4E (рис.2.3).

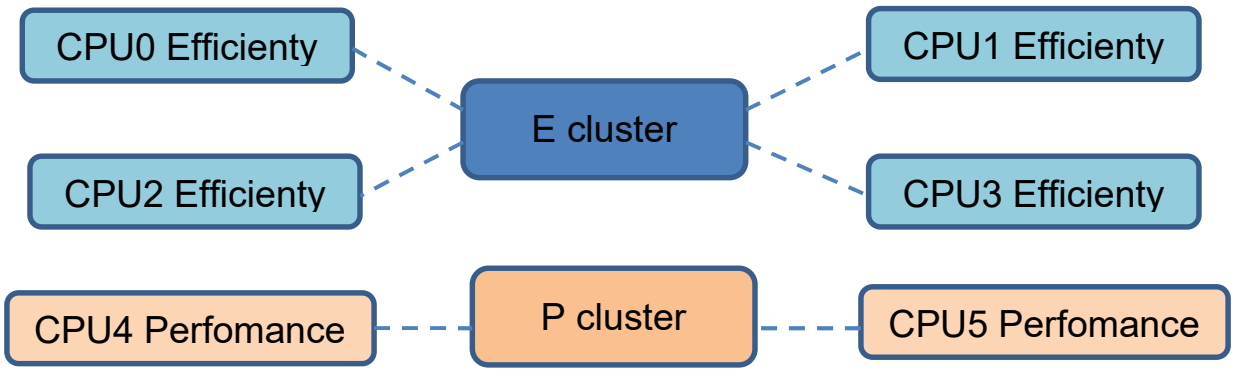

Рисунок 2.3 – Архітектура багатоядерного процесора сімейства Apple Bionic (рисунок виконано самостійно)

Чіпи серії M є радикальним відходом від використання набору мікросхем (чіпсету) для виконання певних функцій, які тепер зосереджені в єдиному кристалі (SoC). З цією інтеграцією пов’язані два подальших нововведення: наявність двох типи процесорних ядер, а пам'ять для них, графічного процесора (GPU) та інших компонентів є спільною для всіх. Кожен чіп серії M містить два різні типи ядер: продуктивні (P) та енергоефективні (E). Базова версія процесорів сімейств M1-M3 має 4 P-ядра та 4 E-ядра на відміну від серії процесорів M3 Pro. M3 Pro є єдиним процесором, який має повні 6 E-ядер в одному кластері разом із 6 P-ядрами, що фактично визначає конфігурацію 6P+6E (див.рис.2.4). Це збільшує його здатність обробляти як фонові завдання, так і завдання з високим рівнем якості обслуговування (QoS). Apple визначає головну роль використання двох додаткових E-ядер в запобіганні перевантаження P-ядер.

Операційна система iOS самостійно приймає рішення щодо використання ядер процесора на основі пріоритетів, призначених потокам, більш відомих як якість обслуговування (QoS). Потоки з низьким QoS майже завжди виконуються на E-ядрах, тоді як потоки з високим QoS переважно розподіляються на P-ядра, але за потреби можуть виконуватися і на E-ядрах.

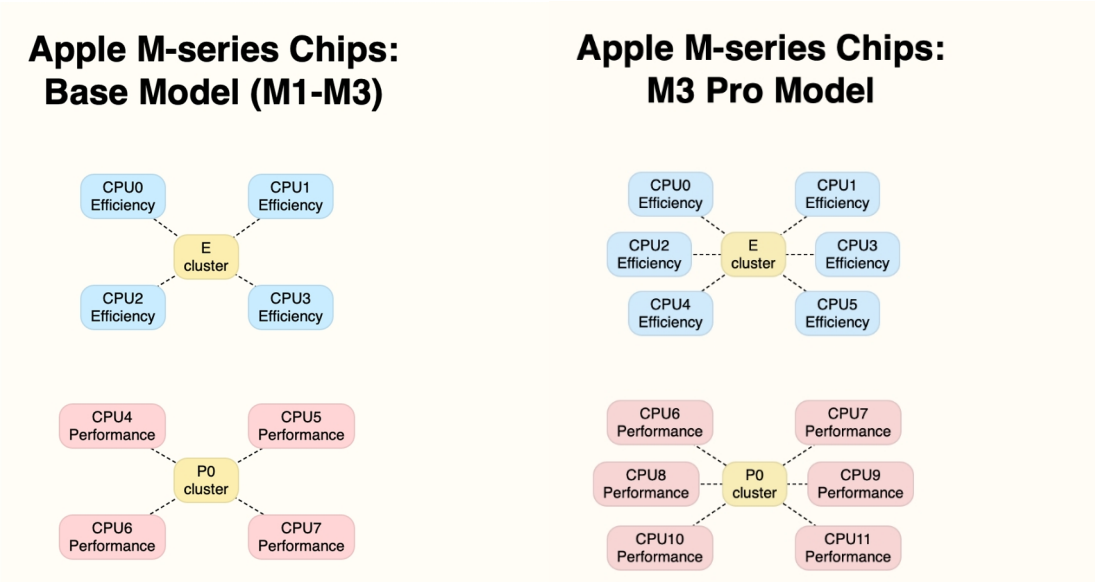

Рисунок 2.4 – Архітектура багатоядерних процесорів сімейства Apple M (за даними [22])

iOS безпосередньо взаємодіє з процесором на рівні ядра операційної системи за рахунок використання функції Grand Central Dispatch (GCD), яка приймає блок даних і викликає його виконання декілька разів на доступних ядрах системи. Ця функція використовує алгоритм крадіжки, що забезпечує завантаження кожного ядра роботою, і (за інформацією сайту Developer, https://developer.apple.com/) є ефективним способом організації паралельної обробки великих обсягів даних. При використанні процесорів Apple цей алгоритм дозволяє ефективно розподілити роботу між P-ядрами та E-ядрами, забезпечуючи динамічне коригування розподілу завдань. Для забезпечення максимальної вигоди від використання цього алгоритму кількість потоків, що обслуговуються в одному мобільному програмному додатку, рекомендується робити принаймні втричі більшою за загальну кількість ядер у системі [23]. Перевірку гіпотези щодо визначення вказаної у рекомендаціях кількості потоків та їх подальша оптимізація буде виконано у практичній частині дослідження з використанням розробленого мобільного додатку.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Розробка моделі ПЗ

Архітектура програмного забезпечення є основним елементом проектування програмного забезпечення. Мобільний додаток було спроєктовано з використанням архітектурного підходу The Composable Architecture [24], адаптованого для операційної системи iOS (див.рис.3.1).

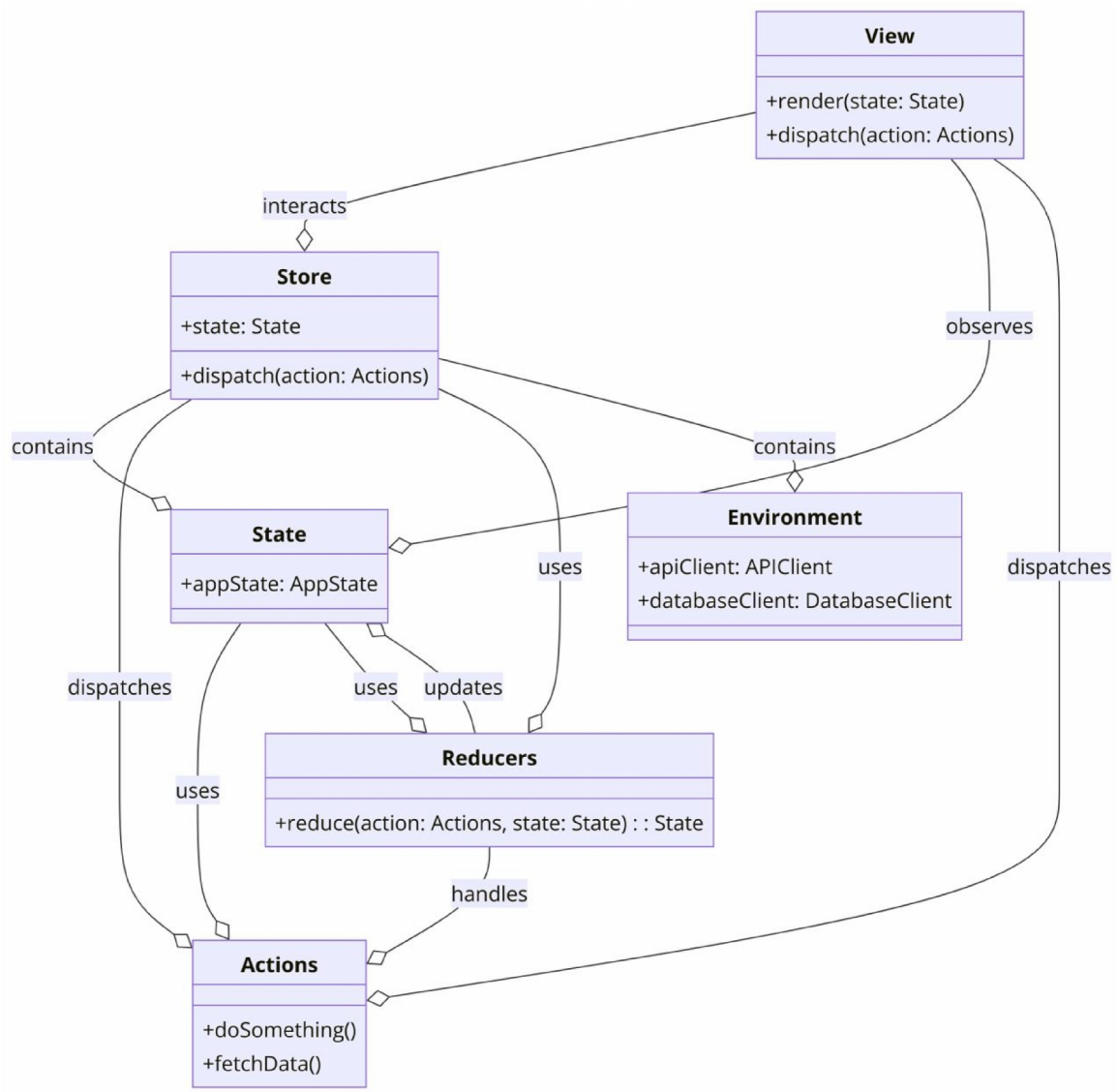

Рисунок 3.1 – Архітектура додатку у відповідності до The Composable Architecture (рисунок виконано самостійно)

The Composable Architecture (TCA) вважається сучасним підходом для

створення складних додатків з використанням SwiftUI. Відмінність TCA від інших підходів до розробки ПЗ полягає в тому, що при використанні технології TCA рекомендується ділити великі функції на дрібніші компоненти, використовуючи ту ж саму логіку розробки. Для кожного з цих компонентів здійснюється модульне тестування, рецензування коду та налагодження на реальному обладнанні. Винесення коду компонента в окремий модуль прискорює процес компіляції проекту.

Процес компонування полягає у з'єднанні незалежних компонентів у більш складну функцію відповідно до заздалегідь визначеної ієрархії та логіки.

Компонентами TCA є:

– View: відповідає за представлення екрана мобільного додатка; в рамках розробки для операційної системи iOS він представлений стандартним класом фреймворку Cocoa Touch UIViewController або його дочірніми класами;

– Store: містить стан програми та управляє діями, які можуть змінити цей стан;

– State: об’єкти, якими управляє Store; є класом або структурою з певними полями, але не містить бізнес-логіку;

– Reducers: функції, які обробляють дії та оновлюють стан;

– Environment: містить залежності, необхідні для виконання бізнес-логіки (API клієнти, бази даних тощо).

Для програмної реалізації функціоналу додатку було використано середовище розробки Xcode та мова програмування Swift. Мова Swift найкращим чином поєднується з платформою iOS, що забезпечує ефективну та швидку розробку мобільних додатків, а Xcode є офіційним інтегрованим середовищем розробки (IDE) для платформ Apple. Схему програмного додатку представлено на рисунку (див.рис. 3.2).

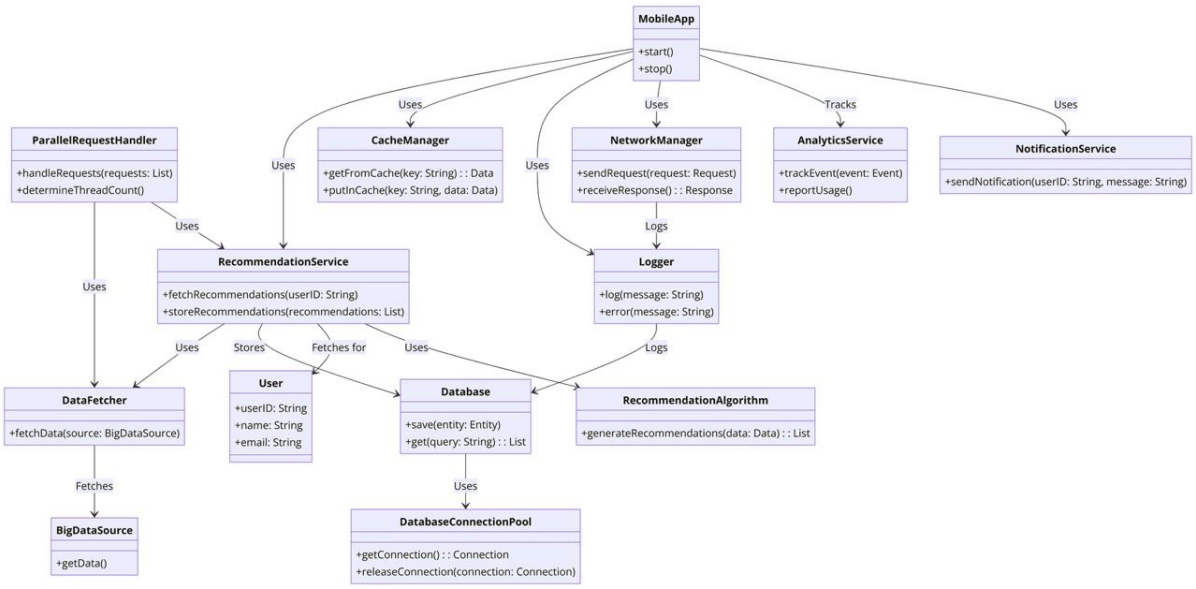

Рисунок 3.2 – Архітектура програмного додатку (рисунок виконано самостійно)

Мобільний додаток через Logger взаємодіє з окремими елементами системи, що забезпечує контроль за усіма операціями, які зосереджено як на стороні бізнес-логіки додатку, так і опрацювання запитів до БД. Основну увагу слід звернути саме на організацію взаємодії БД з джерелом BigData.

Доступ до сховища даних забезпечується через сервіс рекомендацій: запит одночасно надходить як до локальної БД, так і через ParallelRequestHandler – до джерела BigData, в якості якого у проєкті було обрано Google BigTable. ParallelRequestHandler забезпечує організацію розпаралелювання запитів за рахунок використання механізму GCD та OperationQueue.

В якості локальної БД будемо використовувати Firebase Database. Схему та стуктуру зв’язків між полями БД зображено на рисунку (див.рис.3.3).

Слід звернути увагу на той факт, що локальна БД формується виключно шляхом додавання записів на підставі обробки запитів, спрямованих до корпоративного сховища Google Bigtable, визначеного архітектурою додатка Nibble.

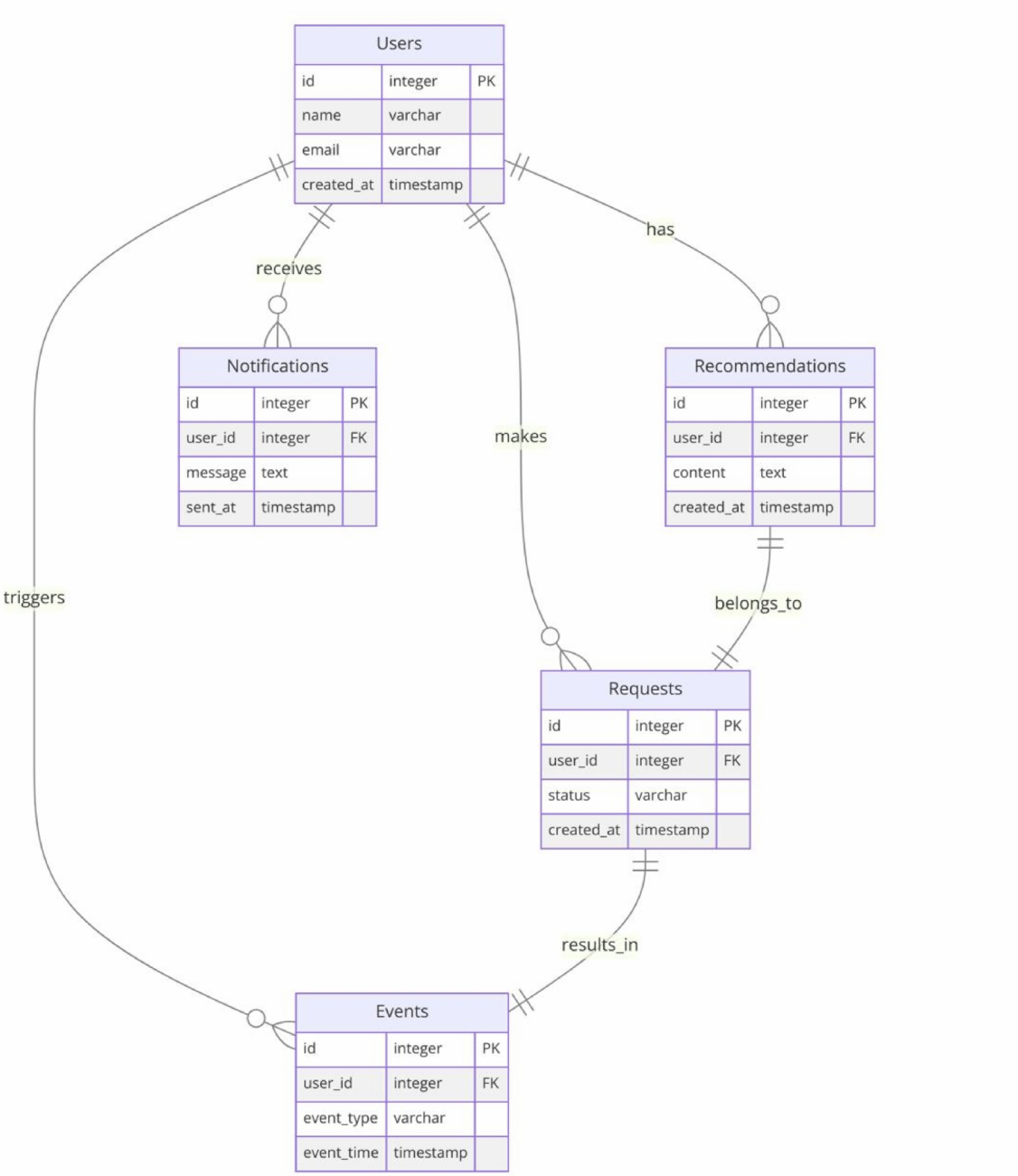

Рисунок 3.3 – Структура БД (рисунок виконано самостійно)

Архітектура БД містить в собі ключові поля, що пов’язані як із користувачами, які зареєстровані у системі, так і рекомендаціями, які можуть надавати інші користувачі, інформація про яких зберігається у зовнішньому сховищі. Формування та оновлення локального сховища відбувається шляхом додавання результатів обробки віддалених запитів, що виконувались паралельно з основним потоком виконання додатку. Таким чином формування та врахування результатів аналізу рекомендацій для визначення змісту та наповнення уроків в

додатку безпосередньо залежить від ефективності використання методів організації паралельної обробки та кількості потоків.

3.2 Реалізація обробки потоків в iOS

Середовище розробки Swift пропонує різні методи підтримки багатопотоковості, серед яких GrandCentralDispatch, OperationQueue, NSLock, а також низькорівневі технології, такі як pthread і NSThread.

GrandCentralDispatch (GCD) – це високорівневий механізм управління багатопотоковістю, який дозволяє легко та ефективно виконувати завдання паралельно. Він використовує черги для організації та виконання завдань та забезпечує автоматичне розподілення завдань між різними потоками.

OperationQueue також є високорівневим інтерфейсом над GCD, який дозволяє створювати та виконувати операції. Це забезпечує додаткові можливості, такі як встановлення залежностей між операціями та управління конкурентністю.

NSLock є механізмом блокування для управління доступом до ресурсів з різних потоків. Він дозволяє тільки одному потоку доступ до даних, блокуючі у конкретний момент часу всіх інших для того, щоб уникнути конфліктів доступу до даних.

Pthread (POSIX Threads) – узагальнений інтерфейс для роботи з потоками, що забезпечує значну кількість можливостей для контролю над створенням та управлінням потоками, але вимагає й більшої відповідальності від розробника, оскільки орієнтований на пряму взаємодію з апаратним обладнанням і може бути більш складним у практичному використанні на персональних мобільних пристроях під управлінням iOS.

GCD, NSLock та OperationQueue є основними механізмами управління багатопотоковістю в програмних додатках для iOS. Вони забезпечують досить простий і безпечний спосіб організації багатопотоковості, ніж використання pthreads. Тому в роботі приділено увагу реалізації розпаралельвання шляхом використання цих методів.

Незважаючи на той факт, що iOS є багатозадачною ОС, вона не підтримує

декілька одночасних процесів в межах однієї програми. Разом з тим розробникам надається у використання клас Process, призначений для створення нових дочірніх процесів, які не залежать від батьківського процесу, але містять всю інформацію, яку мав батьківський процес на момент створення дочірнього процесу. Тому головною задачею, пов'язаною з організацією паралельної обробки в iOS, є організація ефективного управління чергами.

GCD використовує два типи черг: серійні та паралельні.

Серійні (Serial) черги забезпечують виконання задачі одну за одною. Коли перше завдання завершується, починається наступне. Це є корисним, коли необхідно забезпечити, щоб завдання виконувались строго у визначеному порядку. Також такий підхід допомагає уникнути проблем зі станом гонки, оскільки у будь-якої момент часу в серійній черзі виконується лише одне завдання.

Паралельні (Concurrent) черги можуть забезпечити виконання декількох завдань одночасно. Завдання починають виконуватися у тому порядку, в якому вони були додані в чергу, але закінчитися вони можуть у будь-якому порядку. Таким чином, наступне завдання може розпочатися навіть до того, як попереднє завдання буде завершено.

В iOS існує один користувальницький процес, в якому можна визначити до 64 окремих потоків [18]. Для керування цими потоками Apple рекомендує використання черг розсилки: існує можливість додавати завдання до черги шляхом відправлення повідомлень до операційної системи та очікувати, доки вони не будуть виконані у певний момент часу. Відповідно до принципу реалізації механізму GCD, всі операції розміщуються в черзі, яка забезпечує їх виконання у певному порядку. Існують три типи черг: основна черга (Main Queue), яка виконується в основному потоці, глобальна черга (Global Queue), яка виконується у фоновому режимі, та користувальницькі черги (Custom Queue), які дозволяють створювати власні черги – послідовні або паралельні.

Для організації черг запитів до БД та сховища неструктурованої інформації будемо використовувати наступний фрагмент коду, який забезпечує доступ до черги запитів в асинхронному режимі:

```
      // Enqueueworkasynchronouslyontothe ( serial ) mainqueue
      DispatchQueue . main . async {
      // worktobedone
      }
      // Enqueueworkasynchronouslyontooneofthefourbackgroundqueues (
withqos : . defaultinferred )
      DispatchQueue . global (). async {
      // worktobedone
      }
```

Цей фрагмент коду забезпечує використання паралельного коду за допомогою інфраструктури GCD, який полягає в асинхронному відправленні завдання в одну з черг відправлення запиту, таким чином звільняючи основний потік для виконання іншої роботи, поки базова система керує плануванням та виконанням поточного завдання. Після завершення завдання функції, які забезпечують асинхронну роботу в моделі GCD, приймають параметр, який має назву обробника завершення. Асинхронна функція, у свою чергу, може також викликати свій обробник завершення з власним результатом:

```
      funcfoo ( ch : @escaping (Foo) ≠>Void ) {
      // ( Possibly ) dosome " regular" syncwork
      DispatchQueue . global (). async {
      letres : Foo = slowFunction ()
      ch (res)
      }
      // Theasync () functioncallisasynchronous , sowemay ( andoftenwill )
reachthispoint
      // beforeslowFunctionreturns ( orevenbeforeitiseveninvoked )
      return
      }
```

Модель паралелізму, реалізована за допомогою GCD, також підтримує поняття груп диспетчеризації, які дозволяють керувати групами завдань, що знайшло своє використання під час програмної реалізації мобільного додатку.

При створенні черг необхідно брати до уваги той факт, що існує можливість обробляти одночасно будь-яку кількість завдань типу зчитування даних, оскільки вони не змінюють зміст. Однак коли виникає необхідність змінити дані, то необхідно заблокувати всю чергу, щоб усе, що вже було надіслано, було завершено, а нові запити – не виконувались до завершення оновлення даних. З цією метою використовується бар’єр диспетчеризації (dispatchbarrier):

```
      privateletthreadSafeCountQueue = DispatchQueue(label: "...",
attributes: .concurrent)
      privatevar_count = 0
      publicvarcount: Int {
            get{
```

```
      returnthreadSafeCountQueue.sync {
            return _count
       }
            }
            set{
      threadSafeCountQueue.async(flags: .barrier) { [unownedself]
inself._count = newValue
      }
            }
      }
```

Черга операцій (OperationQueue) забезпечує виконання послідовності операцій відповідно до значень залежностей, які має операція. Після додавання операції до черги вона виконується до тих пір, поки не буде завершена або скасована. Після додавання операції до OperationQueue, її неможливо додати до будь-якої іншої OperationQueue, однак існує можливість об’єднувати їх у підкласи, щоб у разі необхідності їх можна було виконувати декілька разів.

Комбінація черги операцій з бар’єром диспетчеризації дозволяє будувати достатньо складні схеми логічного спрацювання, що може знайти своє використання при дослідженні програмної реалізації додатку, орієнтованого на організацію паралельної обробки запитів до великих обсягів неструктурованої інформації у середовищі iOS.

3.3 Планування експериментів

Як вже було вказано у розділі 1, програмна система Nibble: Your Bite of Knowledge – це освітній застосунок з короткими уроками за різними темами, який забезпечує їх формування з урахуванням рекомендацій [16]. Дослідження, що проводились в рамках підготовки кваліфікаційної роботи магістра, орієнтовані на впровадження методів розпаралелювання для покращення продуктивності цього додатку з метою його подальшого вдосконалення.

Експерименти щодо дослідження ефективності методів паралельної обробки великих обсягів неструктурованої інформації з використанням PMD під управління iOS будуть проводитись з використанням наступного апаратного обладнання:

– Apple iPhone 14 128GB (процесор Apple A15 Bionic, iOS 17.5.1);
– Apple iPad Pro 11" (4 Gen) Wi-Fi 128GB (процесор Apple M2, iOS 17.5.1).

Розробка та програмна реалізація моделі програмної системи, яка орієнтована на організацію паралельної обробки BigData, було виконано з використанням Apple MacBook Pro 16" M1 Pro 32/512GB, операційна система macOS Monterey, середовище розробки Xcode 15.3 (15E204a), мова програмування Swift 5. Для вимірювання завантаження процесора під час виконання окремих операцій використовувалось підключення PMD до комп'ютера з використанням програми Activity Monitor.

Для порівняння продуктивності організації обробки даних дослідження проводитись з використанням робочого простору, організованого з використанням Google BigTable, у відповідності до наступних сценаріїв запитів:

– пошук 10000, 100000 та 200000 випадково визначених чисел в діапазоні від 0 до 1000 серед записів у БД;

– вибірка з БД розміром у 10000, 100000 та 200000 записів;

– додавання 100, 500, 1000 записів до БД;

– читання 1000, 10000, 100000 записів з БД.

Кожен із сценаріїв виконувався 10 разів.

Результати проведення експериментів будуть порівнюватись між собою за наступними параметрами [26]:

– кількість потоків;
– навантаження процесору;
– час виконання сценарію.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Проведення експериментів

У відповідності до планування експерименту початкову кількість потоків у програмному додатку, який реалізовував технологію паралельної обробки черг, було визначено у відповідності до загальної кількості ядер: для PDM Apple iPhone на базі родини процесорів Bionic вона складає 6 ядер, тому кількість потоків було визначено за рекомендацією Apple у 18 потоків. Перший запуск додатку на Apple iPhone 14 128GB (операційна система iOS 17.4.1) призвів до необґрунтовано довгого виконання сценарію: дочекатись результату не вдалось. Детальний аналіз повторних запусків з використанням Activity Monitor дозволив визначити достатньо дивну ситуацію: на початку виконання програми задачі отримали всі ядра процесора, але через приблизно 8 сек. високопродуктивні ядра припиняють свою роботу, залишаючи завантаженими тільки 4 енергоефективних ядра E (див.рис.4.1), а інші запити залишались у черзі до цих ядер до завершення програми.

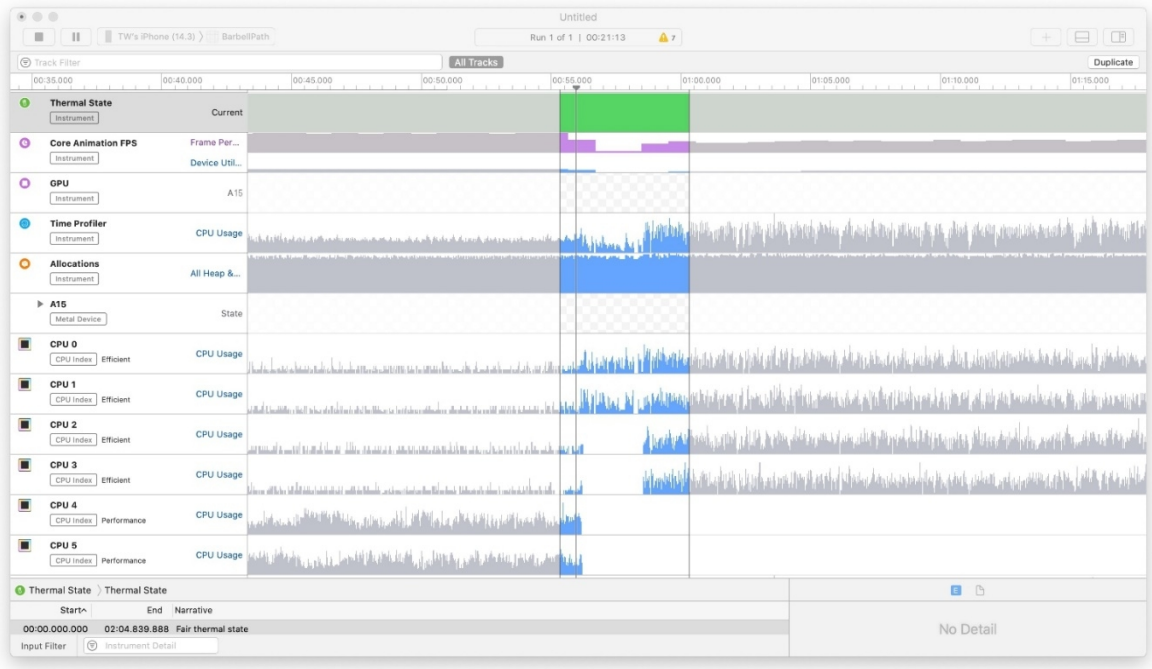

Рисунок 4.1 – Використання ядер процесора A15 Bionic
(рисунок виконано самостійно)

Проаналізувавши отримані результати та вивчивши технічну документацію Apple я дійшов висновку, що Apple при побудові операційної системи iOS для використання на PMD віддала перевагу використанню саме енергоефективних ядер E в якості основних з міркувань збільшення терміну служби батареї, а також (швидше за все) для запобігання перегріву SoC. Зміна параметрів QoS для користувальницьких потоків при дослідженні поведінки додатку на Apple iPhone 14 ефективності не мала, тому було прийнято рішення про зменшення кількості потоків у відповідності до загальної кількості ядер.

Тому за результатами досліджень за поведінкою ядер E та P можна зробити припущення: при формуванні значної кількості одночасних запитів до віддалених ресурсів збільшення кількості потоків для їх обробки, яка перевищує загальну кількість ядер E та P процесора Apple призводить до уповільнення процесу їх обробки. Проблему можна вирішити за рахунок оптимізації механізму формування черг до ядер, однак зараз серед фремворків, що рекомендовані для використання Apple, відсутній інструментарій для вибіркового завантаження окремих ядер – формування черг забезпечується використанням GCD, OperationQueue та їх підтримка з боку iOS. Разом з тим мобільний додаток, запущений на Apple iPad Pro 11" з процесором Apple M2, дозволив за рахунок зміни параметрів QoS сформувати окремі черги як до ядер E, так і до ядер P, однак обмежене поширення використання Apple iPad Pro у порівнянні з Apple iPhone всі подальші дослідження проводились на PMD Apple iPhone.

4.1.1 Пошук випадкових чисел

Дослідження проводились шляхом порівняння результатів, отриманих від виконання двох однакових за функціональністю додатків, один з яких було реалізовано з використанням стандартної послідовної черги виконання запитів, а другий – з використанням GCD та OperationQueue. Результати виконання процедури пошуку 10000, 100000 та 1000000 випадково визначених чисел в діапазоні від 0 до 1000 серед записів у БД наведено на рисунку (див.рис.4.2), на якому відображено середнє використання процесора відповідними програмами під

час виконання пошуку випадкового числа в залежності від кількості записів у БД. З рисунку видно, що додаток, який реалізує паралельну обробку запитів, досягнув більш високого відсоткового навантаження на обчислювальний блок, ніж додаток, який реалізує послідовне виконання запитів. Різниця становила відповідно близько 10,5% для 10000 чисел, близько 5% для 100000 чисел і близько 2,5% для 200000 чисел.

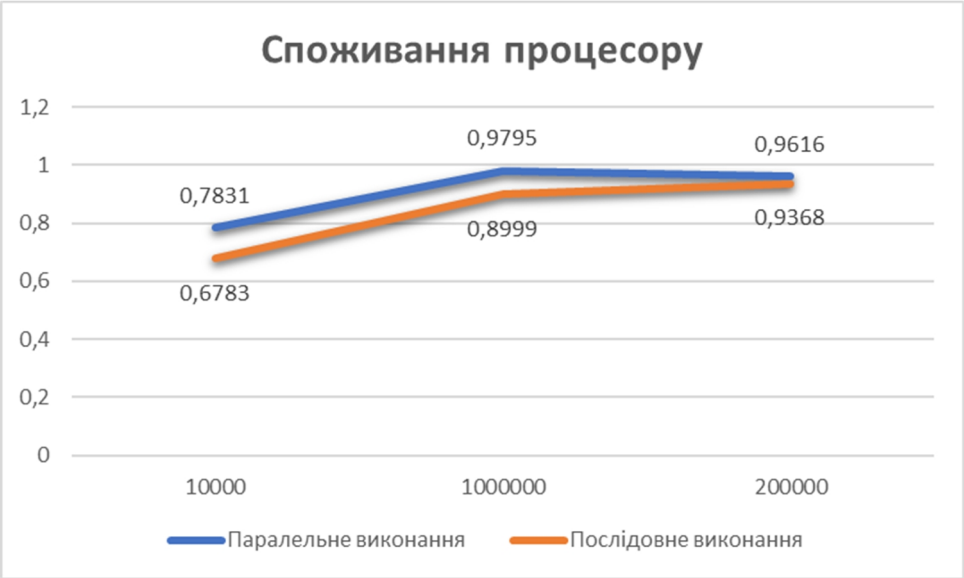

Рисунок 4.2 – Навантаження процесора при виконанні програм (рисунок виконано самостійно)

Наведені результати свідчать про наближення результатів обчислень один до одного та втрату переваг використання розпаралелювання саме за рахунок використання різних за продуктивністю ядер: використання чотирьох енергоефективних E-ядер за тривалий термін часу виконання програми втрачає свої переваги у порівнянні з використанням для виконання тієї ж самої роботи одноядерного високопродуктивного процесора.

4.1.2 Вибірка даних з БД

На рисунку (див.рис.4.3) відображено середній час виконання вибірки запису з БД. Додаток, створений з використанням методів паралельного виконання запитів, виконав процедуру вибірки швидше: найменша різниця в часі була

отримана при розмірі вибірки у 1000 записів і склала близько 0,07 сек. що становить близько 436% на користь паралельного виконання запитів, тоді як найбільша різниця в часі була отримана при розмірі вибірки у 200000 записів і склала близько 0,13 сек. Отримані результати свідчать, що додаток, який реалізує послідовну обробку черги, був приблизно на 320% повільнішим у порівнянні з паралельним додатком. З рисунку 4.3 видно, що різниця між ефективністю виконання запитів додатками у відсотках збільшується зі збільшенням розміру вибірки.

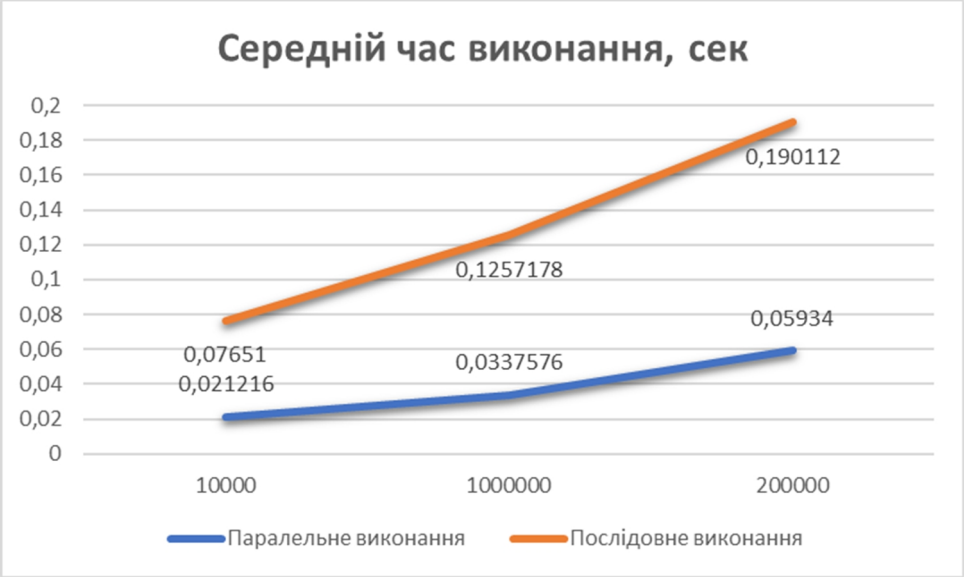

Рисунок 4.3 – Середній час виконання вибірки зі сховища даних (рисунок виконано самостійно)

Цей факт свідчить, що при формуванні вибірки продуктивність процесора не має суттєвого впливу на результати виконання програми: перевагу отримують засоби паралельного виконання запитів.

4.1.3 Запис до БД

Наступний рисунок (див.рис.4.4) відображає середній час запису записів до БД в залежності від кількості записів. Результати проведених експериментів свідчать, що додаток, реалізований з використанням методів паралельної обробки, швидше виконував процедуру записи до БД: найменша різниця в часі була отримана у випадку внесення 100 записів і склала приблизно 0,12 с, що становить

приблизно 530% на користь паралельного додатку, тоді як найбільша різниця була отримана у випадку внесення 1000 записів і склала приблизно 0,53 сек. (див.рис.4.4).

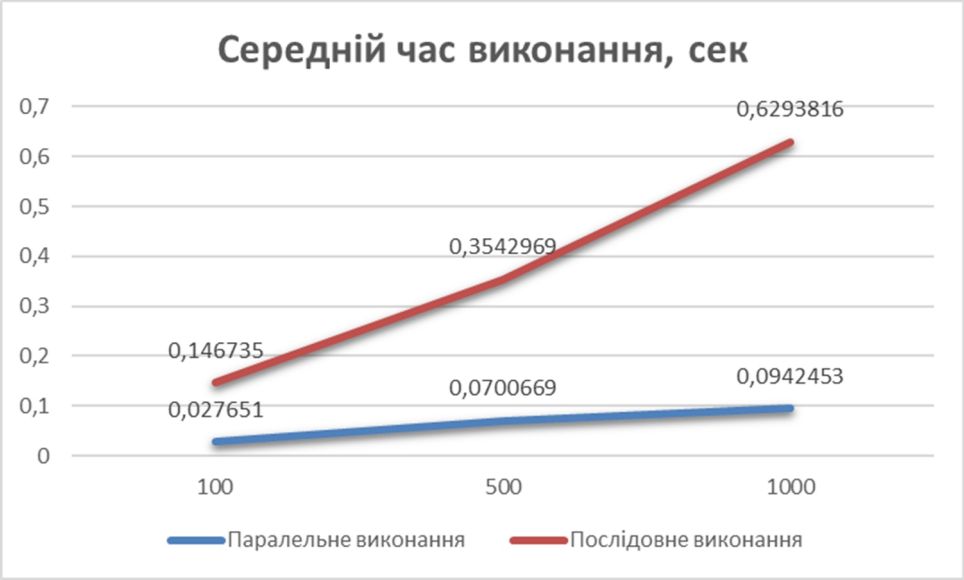

Рисунок 4.4 – Середній час виконання запису у сховище даних (рисунок виконано самостійно)

Це свідчить, що додаток, який реалізує послідовну обробку черги з використанням лише одного високопродуктивного ядра, був повільнішим приблизно на 667%. З результатів проведених досліджень видно, що різниця в продуктивності між додатками збільшується зі зростанням кількості доданих записів.

На наступному рисунку (див.рис.4.5) показано середнє споживання процесора під час організації запису до БД залежно від загальної кількості записів. Результати досліджень свідчать, що додаток, який використовує 4 потоки для опрацювання запитів на енергоефективних ядрах з використанням методів розпаралелювання, очікувано менше завантажував процесор для всіх наборів даних. Однак завантаження процесора Apple iPhone (як для однопотокового, так і для багатопотокового виконання) все одно було достатньо значним.

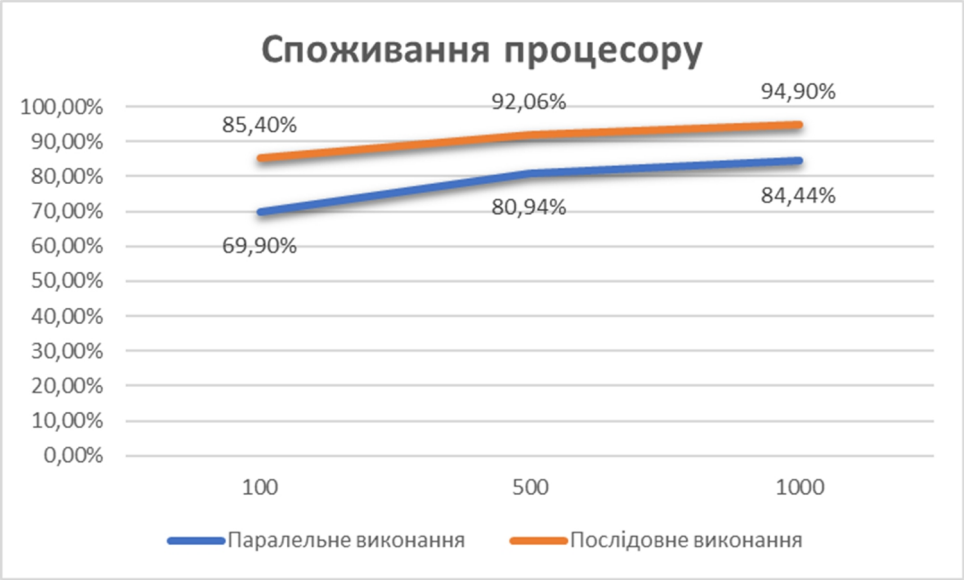

Рисунок 4.5 – Середнє споживання процесору під час запису до сховища (рисунок виконано самостійно)

Найбільша різниця завантаженості спостерігалась при 100 записах, що становить приблизно 15 відсотків, тоді як найменша різниця була отримана при обробці 1000 записів, що становить приблизно 10,5 відсотків. З результатів проведених досліджень видно, що різниця в продуктивності між програмами зменшується зі збільшенням кількості збережених записів.

4.1.4 Читання записів з БД

Результати подальших досліджень наведені на наступному рисунку (див.рис.4.6). На ньому показано середній час виконання процедури зчитування записів з БД в залежності від їх кількості. Результати проведених досліджень свідчать, що додаток, розроблений з використанням технологій паралельного обслуговування черг, виконав цей дослідницький сценарій швидше для всіх наборів даних. Найбільша різниця спостерігалась при зчитуванні 1000 записів і становила приблизно 0,42 сек., що майже у 20 разів перевищує час зчитування, який продемонстрував однопотоковий додаток.

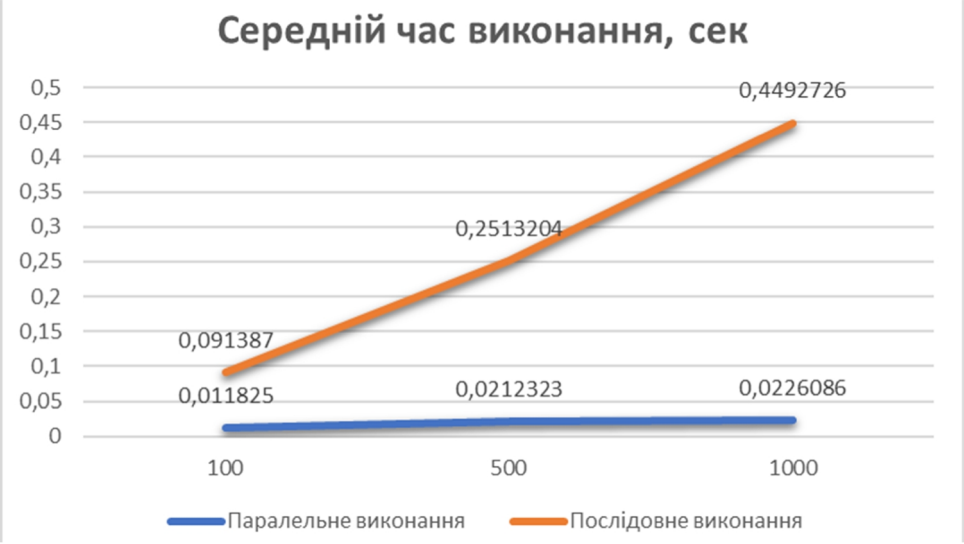

Рисунок 4.6 – Середній час виконання зчитування зі сховища (рисунок виконано самостійно)

Найменша різниця в часі була отримана у випадку читання 100 записів і становила близько 0,08 с, що означає, що послідовний додаток був приблизно на 772% повільнішим у порівнянні з додатком, створеним з використанням розпаралелювання черг за допомогою GCD та OperationQueue. З рисунку видно, що різниця у часі виконання збільшується зі збільшенням кількості зчитаних записів.

На рисунку 4.7 наведено залежність середньої завантаженості процесора при зчитуванні записів з БД в залежності від їх кількості.

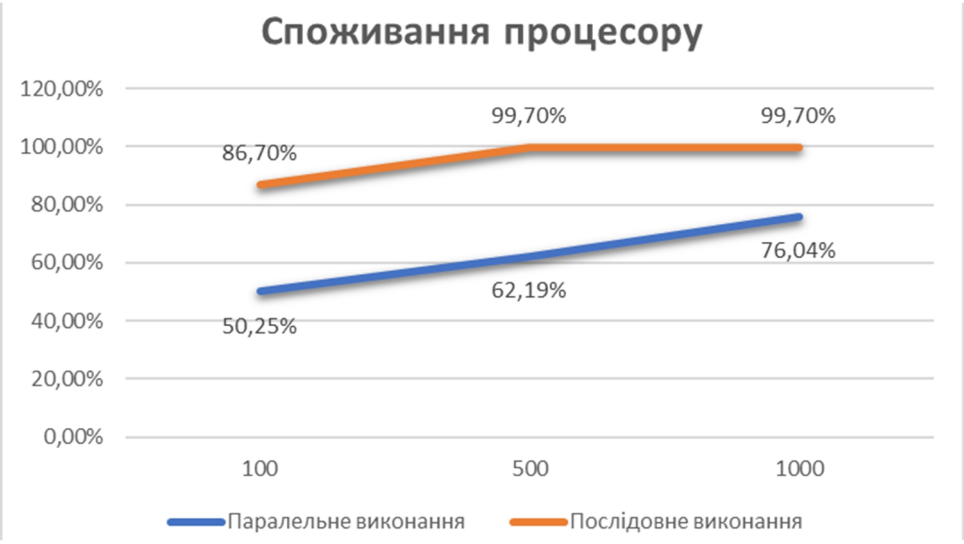

Рисунок 4.7 – Середнє споживання процесору під час читання з БД (рисунок виконано самостійно)

Результати проведених експериментів свідчать, що програма, яка реалізує виконання процедур зчитування за рахунок розпаралелювання черг, менше завантажувала процесор для всіх наборів даних. Найменша різниця між додатками була отримана для 1000 записів і становила приблизно 27%, тоді як найбільша різниця була отримана при зчитуванні 500 записів і становила приблизно 37,5%. Отримані результати свідчать, що різниця в продуктивності між додатками зменшується зі збільшенням кількості зчитаних записів, але це відбувається при майже 100% завантаженості процесора при виконанні послідовної обробки черги.

4.2 Аналіз отриманих результатів

За результатами проведених досліджень можна зробити наступні висновки:

а) результати проведених експериментів свідчать про відносну ефективність використання методів розпаралелювання при розробці програмних додатків для PMD Apple у зв’язку з неможливістю організації відокремлених черг до енергоефективних ядер E та високопродуктивних ядер P;

б) ефективність застосування GCD та OperationQueue при формуванні та обслуговуванні черг значним чином залежить від версії операційної системи та родини процесора, який було використано в певному PMD;

в) сучасні фреймворки та інструментальні засоби, що використовуються на теперішній час при розробці програмних додатків для PMD Apple iPhone під управлінням iOS, не забезпечують можливості керувати типом ядра процесора, на якому бажано запустити процес, однак достатньо ефективні при організації обробки черг і надають можливість керувати ними за рахунок зміни параметрів QoS для кожного з потоків;

г) проведені експерименти свідчать, що довготривале виконання запитів різного типу до сховища неструктурованої інформації, організованого за допомогою Google BigTable, при використанні розпаралелювання на 4, 6 та 8 незалежних потоків запитів не змінює загальну продуктивність системи, оскільки

не спостерігалось простоїв енергоефективних ядер типу E; спроби виведення з простою високопродуктивних ядре типу P успіхів не мало;

д) збільшення загальної довжини черг запитів до енергоефективних ядер типу E з плином часу втрачає переваги у продуктивності за рахунок використання розпаралелювання у порівнянні з використанням лише одного потоку запитів на одному високопродуктивному ядрі процесора родини Bionic.

Отримані під час проведення досліджень результати свідчать про необхідність подальшого вивчення можливих шляхів підвищення продуктивності програмних додатків для PMD Apple під управлінням iOS за рахунок використання інших засобів організації розпаралелювання, у тому числі – при організації запитів до сховищ великих обсягів неструктурованої інформації.

ВИСНОВКИ

Під час підготовки кваліфікаційної роботи, яка ставила на меті дослідження можливих шляхів організації паралельної обробки запитів до джерел неструктурованої інформації, було проведено детальний аналіз проблемної області дослідження, вивчено особливості організації паралельної обробки на PMD Apple та особливості використання великих обсягів інформації на прикладі мобільного додатку Nibble. На підставі проведеного аналізу наукової літератури та технічної документації, що описує принципи та особливості організації паралельної обробки даних на процесорах Apple, було визначено та описано проблеми, пов’язані з обробкою великих обсягів неструктурованої інформації, на підставі яких було виконано постановку задачі дослідження та визначено обмеження, що накладаються на предметну область.

Використовуючи науковий підхід до розв’язання проблеми паралельної обробки неструктурованої інформації було побудовано математичну модель подання BigData, визначено ключові параметри доступу до даних, що були у подальшому використані для проектування архітектури та програмної реалізації моделі програмного забезпечення для проведення експериментальних досліджень щодо ефективності використання розпаралелювання для PMD Apple при організації обробки запитів до сховища BigData, організованого з використанням Google Bigtable.

Дослідження, які проводились з використанням різних типів PMD Apple під управлінням iOS, виявили суттєву залежність ефективності організації паралельної обробки не лише у залежності від сімейства процесорів Apple, а і у залежності від версії операційної системи – використання застарілої версії операційної системи призвела до «клінча» програмного додатку у спробах керування чергою до ядер CPU Apple.

Також за результатами досліджень було з’ясовано, що кількість черг до кожного з ядер CPU, яка була визначена за рекомендаціями Apple у кількості 3 потоків для кожного з ядер CPU, не призводить до оптимізації виконання черг запитів до віддалених сховищ, а може бути використана виключно при організації

локальних арифметичних обчислень на PMD Apple. Тому при організації багатопоточної обробки черг запитів до зовнішніх сховищ кількість потоків не повинна перевищувати загальної кількості ядер CPU з урахуванням того факту, що на початковому етапі виконання програмного додатку завантажуються всі ядра – як високопродуктивні P, так і енергоефективні E, а вже під час обробки запитів черги послідовно перемикаються на використання виключно енергоефективних ядер E.

Отримані у підсумку проведених експериментів висновки дозволяють рекомендувати використання методів розпаралелювання, що базуються на GCD та OperationQueue, при формуванні та обробці запитів до сховищ великих об’ємів даних за допомогою персональних мобільних пристроїв під управлінням iOS, але вимагають проведення подальших досліджень в цій галузі щодо організації ефективного управління чергами. Засоби розпаралелювання, що були досліджені під час підготовки кваліфікаційної роботи магістра, можуть знайти своє впровадження при розробці сучасних мобільних програмних додатків.

Результати досліджень, що проводились під час підготовки та написання кваліфікаційної роботи, було представлено на 28-й Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті» [23] та Міжнародній науково-практичній конференції «Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку» [25].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kirill Smelyakov, Olha Klochko, Zoia Dudar. Building Quantile Regression Models for Predicting Traffic Flow // Proceedings of the 7th International Conference on Computational Linguistics and Intelligent Systems (COLINS), Volume I: Main Conference, 2023. In CEUR Workshop Proceedings, Vol-3387, 2023, pp. 117-132. URL: https://ceur-ws.org/Vol-3387/ (дата звернення: 16.02.2024).

2. Natalija Sharonova, Glib Tereshchenko. Application of Big Data Methods in E-Learning Systems // 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2021), Kharkiv, Ukraine, April 22-23, 2021. – CEUR Workshop Proceedings 2870, Volume I, PP. 1302-1311. URL: http://ceur-ws.org/Vol-2870/ (дата звернення: 16.02.2024).

3. John L. Hennessy, David A. Patterson. Computer Architecture: A Quantitative Approach. 6th Edition – November 23, 2017.– eBook ISBN: 9780128119068. URL: http://acs.pub.ro/~cpop/SMPA/Computer%20Architecture,%20Sixth%20Edition_%20A%20Quantitative%20Approach%20(%20PDFDrive%20).pdf (дата звернення: 30.01.2024).

4. Семеренко, В. П. Технології паралельних обчислень: навчальний посібник / Семеренко В. П. – Вінниця : ВНТУ, 2018. – 104 с.

5. Bouras, M., Idrissi, A. (2023). A Survey of Parallel Computing: Challenges, Methods and Directions. In: Idrissi, A. (eds) Modern Artificial Intelligence and Data Science. Studies in Computational Intelligence, vol 1102. Springer, Cham. URL: https://doi.org/10.1007/978-3-031-33309-5_6 (дата звернення: 10.02.2024).

6. Julian Schutte, Dennis Titze. liOS: Liftingi OS Apps for Funand Profit. URL: https://arxiv.org/pdf/2003.12901(дата звернення: 16.02.2024).

7. Apple Silicon M2 Roadmap (назва з екрану) https://techjourneyman.com/blog/apple-silicon-m2-roadmap/ (дата перегляду 20.03.2024)

8. Apple Inc. Accelerate graphics and much more with Metal. URL: https://developer.apple.com/metal/ (дата звернення: 03.03.2024).

9. Marz, N., Warren, J. BigData: Principles and bestpractices of scalable realtime data systems. Manning Publications, 2015 – 328 p. URL: https://github.com/Jiamim/ecneics-atad/blob/master/Big%20Data%20-%20Principles%20and%20best%20practices%20of%20scalable%20realtime%20data%20systems%202015.pdf (дата звернення 20.03.2024)

10. Zheng, T., Chen, G., Wang, X. et al. Real-time intelligent big data processing: technology, platform, and applications. Sci. China Inf. Sci. 62, 82101 (2019). URL: https://doi.org/10.1007/s11432-018-9834-8 (дата звернення: 26.02.2024).

11. Bigdata architectures (назва з екрану). URL: https://learn.microsoft.com/en-us/azure/architecture/databases/guide/big-data-architectures (дата звернення 20.03.2024)

12. Zhou Feng, W. Hsu, Mong Li Lee. Efficient pattern discovery for semistructured data // 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-05). – 2005. – P. 301–309.

13. What Is Lambda Architecture? (назва з екрану). URL: https://hazelcast.com/glossary/lambda-architecture/ (дата звернення 20.03.2024)

14. A. Khrystova, N. Kravets. Using lambda architecture for big data analysis / URL: https://scholar.google.ru/citations?view_op=view_citation&hl=ru&user=wnxpkmsAAAAJ&sortby=pubdate&citation_for_view=wnxpkmsAAAAJ:YOwf2qJgpHMC (дата звернення 20.05.2024)

15. What Is the Kappa Architecture? (назва з екрану). URL: https://hazelcast.com/glossary/kappa-architecture/ (дата звернення 20.03.2024)

16. Дмитро Полурєзов. Від концепції до реалізації: як побудувати ефективну рекомендаційну систему на основі машинного навчання за два дні (назва з екрану). URL: https://dou.ua/forums/topic/47504/ (дата звернення: 12.05.2024).

17. Український застосунок Nibble став застосунком дня у Великій Британії та Ірландії (назва з екрану). URL: https://cases.media/news/ukrayinskii-zastosunok-nibble-stav-zastosunkom-dnya-u-velikii-britaniyi-ta-irlandiyi (дата звернення: 12.05.2024).

18. Количева П.А., Волощук О.Б. Дослідження методів побудови рекомендаційних систем з використанням графових нейронних мереж / Матеріали 26-й Міжнародного молодіжного форуму «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 7. – Харків: ХНУРЕ. 2022.– С.7-8.

19. Shakhovska N., Medykovsky M., Stakhiv P. Application of algorithms of classification for uncertainty reduction/ Przeglad Elektrotechniczny. – 2013, Vol 89, 4. – P. 284–286.

20. Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C; Wallach, Deborah A; Burrows, Michael 'Mike'; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E (2006), "Bigtable: A Distributed Storage System for Structured Data", Research (PDF), Google.

21. Cloud Bigtable: Scale your latency-sensitive applications with the NoSQL pioneer (назва з екрану). URL: https://cloud.google.com/bigtable/ (дата звернення: 26.02.2024).

22. Apple A17 Pro performance review: How fast is Apple's new 3nm chip (назва з екрану). URL: https://www.hardwarezone.com.sg/feature-apple-a17-pro-performance-review-iphone15pro-singapore-price-specs (дата звернення: 26.05.2024).

23. Optimize for Apple Silicon with performance and efficiency cores (назва з екрану). URL: https://developer.apple.com/documentation/apple-silicon/tuning-your-code-s-performance-for-apple-silicon (дата звернення: 20.05.2024).

24. Chung-Sheng Li, Hubertus Franke, Colin Parris, Bulent Abali, Mukil Kesavan, Victor Chang. Composable architecture for rack scale big data computing / URL: https://eprints.soton.ac.uk/398349/1/FGCS_Disaggregated_Datacenter_accepted.pdf (https://doi.org/10.1016/j.future.2016.07.014)

25. Полурєзов Д. С. Про один підхід до обробки великих обсягів неструктурованих даних / 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6., – Харків: ХНУРЕ. 2024. – С. 405-407. DOI: https://doi.org/10.30837/IYF.IIS.2024.405

26. Jalby, William & Wong, David &Kuck, David &Acquaviva, Jean-Thomas &Beyler, Jean-Christophe. (2012). Measuring Computer Performance. DOI: 10.1007/978-1-4471-2437-5_3 (дата звернення: 16.05.2024).

27. Полурєзов Д.С. Обробка BIG DATA на мобільних пристроях / Міжнародна науково-практична конференція — Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку / Збірник тез доповідей (м. Харків, 14 березня 2024 р.). – Харків. – 2024. – С.265-266. URL: https://openarchive.nure.ua/handle/document/9882