НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут телекомунікаційних систем
Кафедра Інформаційно-телекомунікаційних мереж

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ Л.С.Глоба
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 172 Телекомунікації та радіотехніка

на тему: «Метод балансування задач між мобільними вузлами розподіленої системи обчислень**»**

Виконав:
студент VI курсу, групи ТІ-71мп
Павленко Владислав Миколайович _____

Керівник:
доцент кафедри ІТМ, к.т.н., с.н.с.
Алєксєєв М. О _____

Рецензент:

_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут телекомунікаційних систем
Кафедра Інформаційно-телекомунікаційних мереж

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 172 Телекомунікації та радіотехніка («Інформаційно-комунікаційні технології»)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Л.С.Глоба
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Павленку Владиславу Миколайовичу

1. Тема дисертації «Метод балансування задач між мобільними вузлами розподіленої системи обчислень», науковий керівник дисертації Алєксєєв Микола Олександрович, доцент кафедри ІТМ, к.т.н., с.н.с., затверджені наказом по університету від «06» листопада 2018 р. № 4095-с

2. Термін подання студентом дисертації «10» грудня 2018 р.

3. Об’єкт дослідження: процес розподілення задач в системі розподілених обчислень

4. Предмет дослідження: методи балансування задач в системах розподілених обчислень

5. Перелік завдань, які потрібно розробити:

1. Провести аналіз методів роботи систем балансування
2. Запропонувати метод управління завданнями для розподілених обчислень.

3. Розробити та розгорнути макет робочого серверу по розподіленню завдань.

6. Орієнтовний перелік графічного матеріалу:

1. Тема, актуальність, мета, задачі.

2. Порівняльна характеристика розподілених систем.

3. Схема алгоритму роботи системи. Результат роботи.

4. Загальні висновки.

7. Дата видачі завдання «10» листопада 2017 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Дослідження та вивчення отриманого завдання	10.11.17 – 01.01.18	виконано
2	Аналіз існуючих рішень розподілених обчислень	01.01.18 – 01.02.18	виконано
3	Аналіз підходів до розробки систем розподілених обчислень	01.02.18 – 16.02.18	виконано
4	Підготовка тезисів (ПТ-18)	17.02.18 – 20.03.18	виконано
5	Створення прототипу розподіленої системи, опис компонентів	20.03.18 – 01.08.18	виконано
6	Розробка алгоритму балансування задач між мобільними пристроями	01.08.18 – 01.09.18	виконано
7	Написання алгоритму закінчення роботи над завданням	01.08.18 – 26.09.18	виконано
8	Розробка розподіленої системи	01.08.18 – 01.10.18	виконано
9	Підготовка тексту диплому	15.10.18 – 12.11.18	виконано

Студент _____ В.М. Павленко
(підпис)

Керівник роботи _____ М.О. Алєксєєв
(підпис)

РЕФЕРАТ

Робота містить 82 сторінки, 14 рисунків. Було використано 15 джерел.

Мета роботи: Запропонувати метод розподілення задач гетерогенної системи розподілених обчислень для ефективного використання мобільних вузлів.

Проведено аналіз принципів побудови систем розподілених обчислень та виконано огляд існуючих систем.

Розглянуто особливості застосування розподілених систем для мобільної платформи.

Розроблено ПЗ розподіленої системи враховуючи особливості реалізації на мобільних платформах. Сформовано порівняльну характеристику різних алгоритмів обчислень.

Створено модель для демонстрування роботи розробленої системи та використаних алгоритмів.

Ключові слова: розподілені обчислення, гетерогенна система, балансування задач

ABSTRACT

The paper contains 82 pages, 14 pictures, 15 sources were used.

Objective: To offer distributing tasks of a heterogeneous distributed computing system method for efficient use of mobile nodes.

An analysis of the principles of constructing distributed computing systems was held and an overview of existing systems was carried out.

The peculiarities of the distributed systems for the mobile platform application were reviewed.

Distributed system software was developed taking into account the peculiarities of the implementation on mobile platforms. A comparative characteristic of various algorithms of computing was formed.

A model for demonstrating the work of the developed system and used algorithms was developed.

Keywords: distributed computing, heterogeneous system, task balancing

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

JSON	JavaScript Object Notation
MVC	Model-View-Controller
SQL	Structured Query Language
Wi-Fi	Wireless Fidelity
БД	База даних
ПЗ	Програмне забезпечення
ПК	Персональний комп’ютер

ВСТУП

Актуальність. В сучасному світі з кожним днем зростає обсяг інформації яка потребує обробки. Розробка і використання розподілених систем дозволяє виконувати обчислення над великими масивами даних паралельно, що значно пришвидшує отримання кінцевого результату.

Інформація має неоднорідний зміст і потребує різних методів обробки. Для впровадження цих методів в систему розподілених обчислень існують різні підходи до їх побудови. На даний момент найпопулярнішою є технологія MapReduce яка дозволяє охопити найбільш широкий спектр типів завдань. За даним алгоритмом працюють багато розподілених систем, які мають неоднорідну структуру та абстрагуються від певних залежностей платформи на якій вони виконуються, що в свою чергу дозволяє вільно розповсюджувати ПЗ та з легкістю модифікувати його під деякі особливості поставленого на обробку завдання.

Сучасні мобільні телефони (планшети, плеєри, ультрабуки) мають потужні процесорні можливості що наближає їх до повноцінних ПК. Більшу частину працюючого часу, мобільний пристрій знаходиться в режимі очікування, який можна використовувати в корисних цілях. Через велику їх розповсюдженість і постійний зв’язок з Інтернетом за допомогою популярних Wi-Fi мереж або 3G, мобільні пристрої надають можливість об’єднувати їх у системи розподілених обчислень. Дослідження можливості ефективного використання MapReduce на мобільних платформах є актуальної задачею.

Об’єкт дослідження: процес розподілення задач в системі розподілених обчислень.

Предмет дослідження: методи балансування задач в системах розподілених обчислень.

Мета дослідження: покращення розподілення задач в системах розподілених обчислень за рахунок розробки метода балансування.

Для досягнення мети було поставлено та вирішено такі **основні задачі:**

1. Провести аналіз методів роботи систем розподілених обчислень для персональних комп'ютерів;

2. Розглянути особливості застосування запропонованого метода для мобільної платформи.

3. Розробити ПЗ для встановлення на мобільні пристрої та серверну частину системи для автоматизації роботи над завданнями.

Теоретичний результат дослідження:

1. Проведено аналіз методів роботи систем розподілених обчислень та досліджено можливість їх використання для обчислення інформації.

2. Запропоновано метод реалізації балансування задач в системі розподілених обчислень із застосуванням технології MapReduce.

3. Розглянуто особливості побудови запропонованої системи для мобільних платформ.

Практичний результат дослідження:

1. Розроблено метод балансування задач для системи розподілених обчислень, яка базується на технології MapReduce.

2. Створено мобільний додаток для встановлення на мобільні платформи.

3. Додано покращений алгоритм роботи до основного сервера.

РОЗДІЛ 1
АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ СИСТЕМ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ ТА ОГЛЯД ІСНУЮЧИХ ЗРАЗКІВ ЇХ РЕАЛІЗАЦІЇ.

Розділ розглядає опис існуючих систем розподілених обчислень, та методи балансування завдань між вузлами в їх мережі. Завданням розділу є розглянути та обрати метод який буде максимально задовольняти поставленим умовам.

1.1 Розподілені обчислення. Мета та цілі їх застосування

Розподілені обчислення (розподілена обробка даних) — це спосіб розв'язання трудомістких обчислювальних завдань з використанням двох і більше комп'ютерів, об'єднаних в мережу. Розподілені обчислення є окремим випадком паралельних обчислень, тобто одночасного розв'язання різних частин одного обчислювального завдання декількома процесорами одного або кількох комп'ютерів. Тому необхідно, щоб завдання, що розв'язується було сегментоване — розділене на підзадачі, що можуть обчислюватися паралельно. При цьому для розподілених обчислень доводиться також враховувати можливу відмінність в обчислювальних ресурсах, які будуть доступні для розрахунку різних підзадач. Проте, не кожне завдання можна «розпаралелити» і прискорити його розв'язання за допомогою розподілених обчислень.

У цьому визначенні обмовляються два моменти. Перший відноситься до апаратури: всі машини автономні. Другий стосується програмного забезпечення: користувачі думають, що мають справу з єдиною системою. Важливо обидва моменти. Можливо, замість того щоб розглядати визначення, розумніше буде зосередитися на важливих характеристиках розподілених систем. Перша з таких характеристик полягає в тому, що від користувачів приховані відмінності між комп'ютерами і способи зв'язку між ними. Те ж саме відноситься і до зовнішньої організації розподілених систем. Іншою важливою характеристикою розподілених систем є спосіб, за допомогою якого

користувачі і додатки одноманітно працюють в розподілених системах, незалежно від того, де і коли відбувається їх взаємодія.

Розподілені системи повинні також відносно легко піддаватися розширенню, або масштабуванню. Ця характеристика є прямим наслідком наявності незалежних комп'ютерів, але в той же час не указує, яким чином ці комп'ютери насправді об'єднуються в єдину систему. Розподілені системи зазвичай існують постійно, проте деякі їх частини можуть тимчасово виходити з ладу. Користувачі і додатки не повинні повідомлятися про те, що ці частини замінені або полагоджені або що додані нові частини для підтримки додаткових користувачів або додатків.

Для того, щоб підтримати представлення різних комп'ютерів і мереж у вигляді єдиної системи, організація розподілених систем часто включає додатковий рівень програмного забезпечення, що знаходиться між верхнім рівнем, на якому знаходяться користувачі і додатки, і нижнім рівнем, що складається з операційних систем, як показано на рис. 1.1. Відповідно, така розподілена система зазвичай називається системою проміжного рівня (middleware) [1].

Рис.1.1 Архітектура операційної системи

Метою використання розподілених обчислювальних систем є пришвидшення процесу обробки інформації методом дроблення задачі на частини та паралельного виконання їх на процесорах різних машин.

Поява й розвиток розподілених архітектур пов’язані з інтенсивним розвитком технічних і програмних засобів. У цих архітектурах функціональні компоненти інформаційної системи розподіляються по наявних вузлах залежно від поставлених цілей і завдань. Можна виділити шість основних характеристик архітектури розподілених систем [2]:

1. **Спільне використання ресурсів**. Розподілені системи дозволяють спі-ль-не використання апаратних та програмних ресурсів, наприклад жорстких ди-сків, принтерів, файлів, компіляторів та інше, об’єднаних засобами мережі. Оче-видно, що розподіл ресурсів можливий і в багатокористувацьких системах, але в цьому випадку за надання ресурсів і їх керування повинен керувати цент-ральний процесор.

2. **Відкритість**. Це можливість розширювати систему шляхом добавля-ння нових ресурсів. Розподілені системи – це відкриті системи, до яких приєд-нуються апаратне і програмне забезпечення від різних виробників.

3. **Паралельність**. В розподілених системах декілька процесів можуть одночасно виконуватися на різних комп’ютерах в мережі. Ці процеси можуть взаємодіяти один з одним під час їх виконання.

4. **Масштабованість**. В принципі всі розподілені системи є масштабова-ними: щоб система відповідала новим вимогам, її можна нарощувати за допо-мо-гою добавляння нових обчислювальних ресурсів. Але на практиці нарощу-вання може обмежуватися мережею, яка об’єднує окремі комп’ютери системи. Якщо приєднати багато нових машин, то пропускна здатність мережі може ви-явитися недостатньою.

5. **Відмовостійкість**. Наявність декількох комп’ютерів і можливість дублювання інформації означає, що розподілені системи стійкі до певних апаратних і програмних помилок. Більшість розподілених систем у випадку помилки, як правило, можуть підтримувати хоча б частково функціональність. Повний збій в системі відбувається тільки у випадку мережевих помилок.

6. **Прозорість**. Ця властивість означає, що користувачам надано повніс-тю прозорий доступ до ресурсів і в той же час приховано інформацію про розподіл ресурсів у системі. Однак, в багатьох випадках конкретні знання про організацію системи допомагає користувачу краще використовувати ресурси.

До недоліків розподілених систем варто віднести:

1. **Складність**. Розподілені системи складніші від централізованих. Набагато складніше зрозуміти і оцінити властивості розподілених систем в цілому, а також тестувати ці системи. Наприклад, продуктивність системи залежить від швидкості роботи одного процесора, а від смуги пропускання мережі і швидкодії різних процесорів. Переміщаючи ресурси з одної частини мережі в іншу, можемо радикально вплинути на продуктивність системи.

2. **Безпека**. Як правило доступ до системи можемо отримати з декількох різних машин, повідомлення в мережі можуть переглядатися або перехоплюватися. Тому, в розподіленій системі набагато складніше підтримувати безпеку.

3. **Керованість**. Система може складатися з різнотипних комп'ютерів, на яких можуть бути встановлені різні версії операційних систем. Помилка однієї машини не розповсюджується на інші машини з непередбачуваними наслідками. Тому необхідно значно більше зусиль, щоб керувати і підтримувати систему в робочому стані.

4. **Непередбачуваність**. Як відомо всім користувачам Web-мережі, реак-ція розподілених систем на певні події непередбачувана і залежить від повного завантаження системи, її організації і мереженого навантаження. Оскільки ці всі параметри можуть постійно змінюватися, час, затрачений на виконання запиту користувача, в той чи іншій момент може суттєво різнитися.

Всі вони пов'язані в першу чергу зі складною структурою, різноплановим устаткуванням і складною системою розподілу прав доступу. Необхідно враховувати всі з них, інакше розроблена інформаційна система не зможе функціонувати в рамках очікуваних параметрів.

В розподіленій системі різні системні компоненти можуть бути реалізовані на різних мовах програмування і виконуватись на різних типах процесорів. Моделі даних, подання інформації і протоколи взаємодії – все це необов'язково буде однотипним в розподіленій системі. Отже для розподілених систем необхідне таке програмне забезпечення, яке могло би керувати цими різнотипними частинами та гарантувати взаємодію і обмін даними між ними [3]. Проміжне програмне забезпечення відноситься власне до такого класу ПЗ. Воно знаходиться якби посередині між різними частинами розподілених компонент системи.

1.2 Види розподілених архітектур на портативних пристроях

Існує величезна кількість стандартів для створення правильної й надійної архітектури, а також для розробки й інтеграції програмних систем. Застосування цих стандартів істотно збільшить шанси на успішне створення системи і її подальше безвідмовне функціонування, однак раціональність їхнього застосування повинна визначатися до моменту початку робіт, оскільки складність системи при їхній інтеграції може істотно зрости.

Більшість систем, що опрацьовують великі об'єми даних, організовані на основі бази даних, яка використовується спільно, або репозиторія. Прикладом може бути система управління інформацією, система автоматизованого проектування і CASE-засоби [6]. Для систем штучного інтелекту розроблено альтернативний підхід. Він базується на моделі «робочої області» [8].

На першому етапі процесу розробки архітектури система розбивається на декілька взаємодіючих підсистем. На самому абстрактному рівні архітектуру системи можемо зобразити графічно за допомогою блок-схеми, в якій окремі підсистеми подаються окремими блоками. Якщо підсистему також можемо розділити на декілька частин, то на діаграмі ці частини відображаються в середині великого блоку [4].

Існують такі види розподілених архітектур:

– архітектура репозиторія;

– архітектура клієнт/сервер;
– архітектура абстрактної машини.

1.3 Приклади реалізацій розподілених систем

1.3.1 Система UNICORE

Проект UNICORE (Uniform Interface to Computing Resources – єдиний інтерфейс до розрахунковими ресурсам) зародився в 1997 році, та до справжнього моменту представляє собою комплексне рішення, орієнтоване на забезпечення прозорого безпечного доступу до ресурсів грід.

Архітектура UNICORE формується з клієнтського, сервісного та системного слоїв (рис. 1.2). Верхнім шаром в архітектурі являється клієнтский шар. В ньому знаходяться різні клієнти, які забезпечуюють взаємодію користувачів з грід середовищем:

– UCC (Unicore Command Line Client – клієнт командной строки для UNICORE): клієнт, забезпечуючи інтерфейс командної строки для постановки задач та отримання результатів;

– URC (Unicore Rich Client – багатофункціональний клієнт UNICORE): клієнт, оснований на базі інтерфейса середовища Eclipce, представляється в графічному виді повний набір всіх функціональних можливостей системи UNICORE;

– HiLA (High Level API for Grid Applications – високорівневий програмний інтерфейс для додатків грід): забезпечує розробку клієнтів в системі UNICORE;

– Портали: доступ користувачів до грід-ресурсів через інтернет, через інтеграцію UNICORE в системі інтернет-порталів;

Проміжний сервісний рівень вміщує всі сервіси та компоненти системи UNICORE, основані на стандартах WSRF и SOAP. Шлюз – це компонент, який забезпечує доступ до вузлу UNICORE через аутентифікації всіх вхідних повідомлень. Компонент XNJS забезпечує управління задачами та

використання ядра UNICORE 6. Регістр сервісів забезпечує регистрацію та пошук ресурсів, доступних в грід-середовищі.

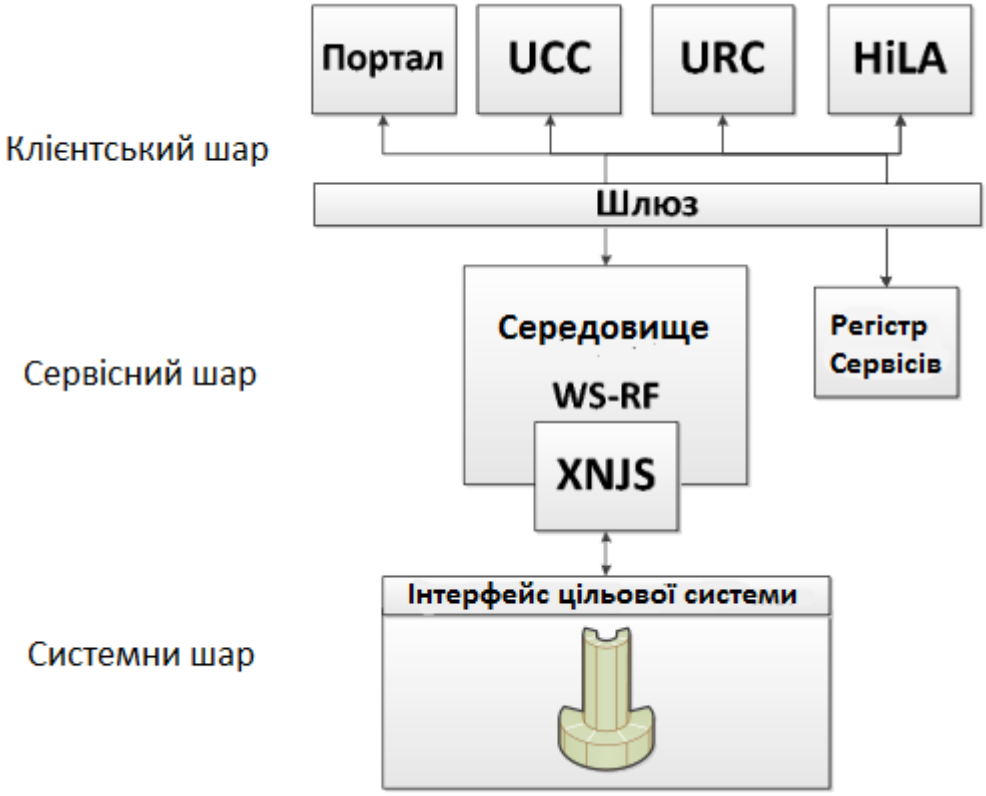

Рис. 1.2 Архітектура UNICORE 6

Також, на рівні сервісного шару забезпечується підтримка безпечних з'єднань авторизації та аутентифікації користувачів.

В основі архітектури UNICORE лежить системний рівень. Інтерфейс цільової системи (TSI – Target System Interface) [13] забезпечує взаємодію між UNICORE та окремим ресурсом грід-мережі. Він забезпечує трансляцію команд, поступаючих з грід-середовища локальної системи.

Основною перевагою використання системи UNICORE 6 для розробки розподілених обчислювальних систем можна брати наявність багатого арсеналу різноманітних клієнтів, забезпечуючи взаємодію користувачів з ресурсами розрахункової мережі, а також розвинутих методів забезпечення безпеки при розробці грід-застосунків.

Принцип роботи та функціональність грід-застосунків значно відрізняються від звичайних послідовних та паралельних систем. Основна відмінність – це можливість агрегування та сумісного використання великих

наборів гетерогенних ресурсів, розподілених між географічно розділеними областями. В багатьох випадках це приносить більше користі, наприклад, коли додаток потребує ресурсів, недоступних в рамках одного вузла, воно може потребувати ресурси у інших вузлів, підключених до грід.

Така складна поведінка несе в собі певні проблеми. Особливо до високо-гетерогенної, динамічно-формованої розподіленої мережі дуже важко напряму застосувати такі традиційні метрики ефективності роботи, як швидкість розрахунку, пропускна здатність каналу та інше. В зв'язку з цим, для оцінки якості запропонованого сервісу потребується використання спеціалізованих метрик.

1.3.2 Система Hadoop

Hadoop - вільна програмна платформа і каркас для організації розподіленої обробки великих обсягів даних (що міряється у петабайтах) з використанням парадигми MapReduce, при якій завдання ділиться на багато дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера. До складу Hadoop входить також реалізація розподіленої файлової системи Hadoop Distributed Filesystem (HDFS), котра автоматично забезпечує резервування даних і оптимізована для роботи MapReduce-застосунків. Для спрощення доступу до даних в сховищі Hadoop розроблена БД HBase і SQL-подібна мова Hive, яка є свого роду SQL для MapReduce і запити якої можуть бути розпаралелені і оброблені кількома Hadoop-платформами.

Ключовий продукт CDH (Cloudera Distribution including Apache Hadoop) — зв'язка найбільш популярних інструментів з інфраструктури Hadoop під управлінням Cloudera Manager. Менеджер бере на себе відповідальність за розгортання кластера, встановлення всіх компонентів і їх подальший моніторинг. Крім CDH компанія розвиває й інші продукти, наприклад, Impala (про це нижче). Відмінною рисою Cloudera також є прагнення першими

надавати на ринку нові фічі, нехай навіть і в збиток стабільності. Ну і так, творець Hadoop — Doug Cutting — працює в Cloudera.

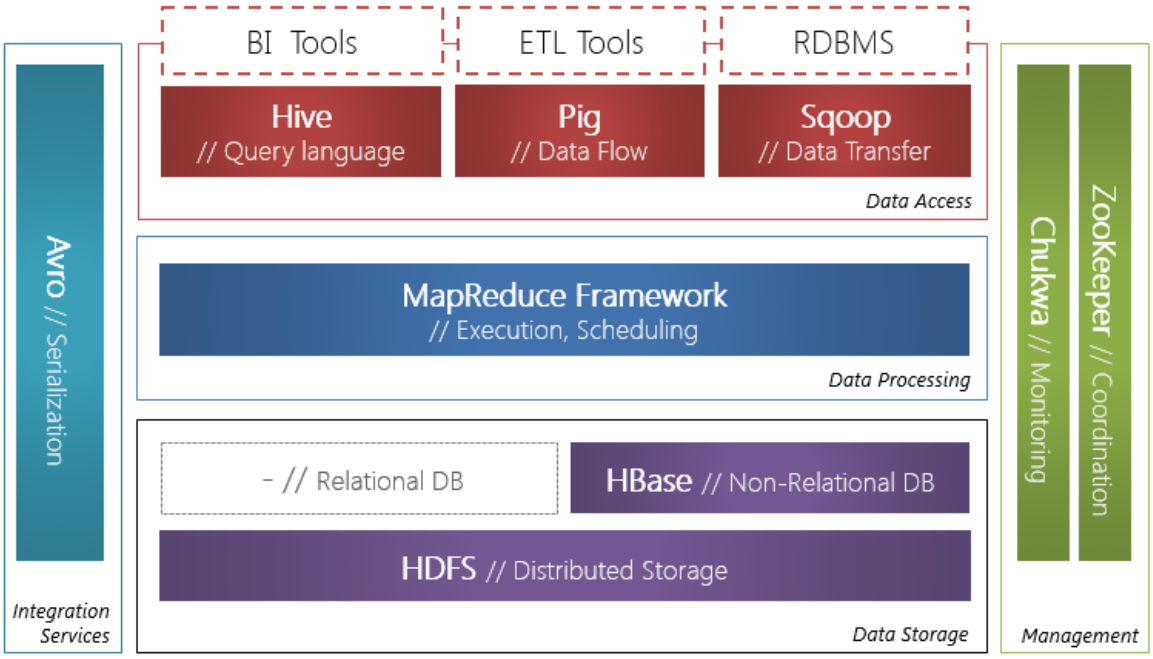

Рис. 1.3 Екосистема Hadoop

Hortonworks. Так само, як і Cloudera, вони надають єдине рішення у вигляді HDP (Hortonworks Data Platform). Їх відмінною рисою є те, що замість розробки власних продуктів вони більше вкладають у розвиток продуктів Apache. Наприклад, замість Cloudera Manager вони використовують Apache Ambari, замість Impala — далі розвивають Apache Hive. Мій особистий досвід з цим дистрибутивом зводиться до парі тестів на віртуальній машині, але за відчуттями HDP ,що на рис. 1.3, виглядає стабільніша, ніж CDH.

MapR. На відміну від двох попередніх компаній, основним джерелом доходів для яких, судячи з усього, є консалтинг і партнерські програми, MapR займається безпосередньо продажем своїх напрацювань. З плюсів: багато оптимізацій, партнерська програма з Amazon. З мінусів: безкоштовно версія (M3) має спрощений функціонал. Крім того, MapR є основним ідеологом і головним розробником Apache Drill.

```
# переглянути кореневу директорію: локально і на HDFS
ls /
hadoop fs-ls /

# оцінити розмір директорії
du-sh mydata
hadoop fs-du-s-h mydata

# вивести на екран вміст всіх файлів в директорії
cat mydata/*
hadoop fs-cat mydata/*
```

Рис. 1.4 Синтаксис Hadoop

Класична конфігурація кластера Hadoop, як на рис. 1.4, складається з одного сервера імен, одного майстра MapReduce (т.зв. JobTracker) і набору робочих машин, на кожній з яких одночасно крутиться сервер даних (DataNode) і воркер (TaskTracker). Кожна MapReduce робота складається з двох фаз:

1. map — виконується паралельно і (по можливості) локально над кожним блоком даних. Замість того, щоб доставляти терабайти даних до програми, невелика, визначена користувачем програма копіюється на сервера з даними та робить з ними все, що не вимагає перемішування і переміщення даних (shuffle).

2. reduce — доповнює map агрегирующими операціями

Насправді між цими фазами є ще фаза combine, яка робить те ж саме, що і reduce, але над локальними блоками даних. Наприклад, уявімо, що у нас є 5 терабайт логів сервера, які потрібно розібрати і отримати повідомлення про помилки. Рядки незалежні один від одного, тому їх аналіз можна перекласти на завдання map. Далі з допомогою combine можна відфільтрувати рядки з повідомленням про помилку на рівні одного сервера, а потім за допомогою reduce зробити те ж саме на рівні всіх даних. Все, що можна було розпаралелити, ми распараллелили, і крім того мінімізували передачу даних

між серверами. І навіть якщо якась задача з якоїсь причини впаде, Hadoop автоматично перезавантажить її, піднявши з диска проміжні результати.

Більшість реальних задач набагато складніше однієї роботи MapReduce. У більшості випадків ми хочемо робити паралельні операції, потім послідовні, потім знову паралельні, потім комбінувати декілька джерел даних і знову робити паралельні і послідовні операції. Стандартний MapReduce спроектований так, що всі результати — як кінцеві, так і проміжні — записуються на диск. У результаті час зчитування і запису на диск, помножене на кількість разів, які воно робиться при рішенні задачі, найчастіше в кілька (та що там декілька, до 100 разів!) перевищує час самих обчислень.

1.3.3 Система BOINC

BOINC (англ. Berkeley Open Infrastructure for Network Computing) — відкрита програмна платформа університету Берклі для GRID обчислень — некомерційне міжплатформне програмне забезпечення для організації розподілених обчислень. Використовується для організації добровольчих обчислень.

Складається з серверної та клієнтської частини, як показано на рис. 1.5. Спочатку розроблявся для найбільшого проекта добровільних обчислень— SETI@home, але потім розробники з Каліфорнійського університету в Берклі зробили платформу доступною для сторонніх проектів. На сьогоднішній день BOINC являється універсальною платформою для проектів в області математики, молекулярної біології, медицини, астрофізики та кліматології. BOINC дає дослідникам можливість задіяти величезні обчислювальні потужності персональних комп’ютерів із всього світу.

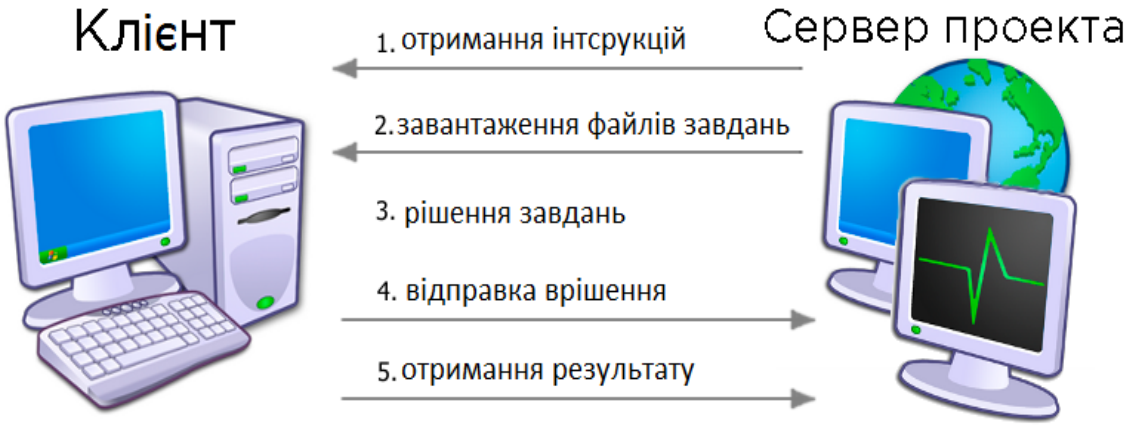

Рис. 1.5 Схема роботи системи BOINC

BOINC розроблений командою в главі з Девідом Андерсоном (David Pope Anderson), очолювавшим також SETI@home, з Space Sciences Laboratory Каліфорнійського університету в Берклі. На 27 березня 2017 року BOINC представляє собою розподілену мережу з більш ніж 830 000 активних комп'ютерів (хостів) із середньою потужністю всієї мережі около 158 петафлопс. Для порівняння, самий потужний суперкомп'ютер на березень 2017 року "СанВей ТайхуЛайт" мав пікову потужність 93 петафлопса. Пікова потужність проекту BOINC зафіксована на рівні 320 петафлопс, що більше ніж як в три рази перевищує пікову потужність самого потужного суперкомп'ютера на Землі.

Серверна частина BOINC. Серверна частина складається з HTTP-сервера з веб-сайтом проекта, бази даних MySQL та набору демонів (генератор задач, планувальник, валідатор, ассимілятор результатів. Сервер — тільки на Linux, переважно Debian.

HTTP сервер представляє собою набір PHP-скриптів та необхідним організаторам проектів для спільного управлення проектом: реєстрація учасників, розподілення завдань для обробки, отримання результатів, управління базами даних проекта.

В базі даних зберігаються користувачі, паролі, записи завдань, результатів, інформація про хости, програмах проекта та інше.

Демони — це набір програм на C++, які виконуються в даному середовищі від імені суперкористувача та виконують прикладні мотиви системи, а саме:

- управління базою даних
- спостереження за активністю користувачів
- запуск необхідних процедур при настанні чекпоінта
- прийом та збір результатів від користувачів, проміжне їх форматування, подальше збереження до бази данних

BOINC-клієнт. Для користувачів поняття BOINC частіше використовується в контексті поняття BOINC-клієнт — універсальний клієнт для роботи з різними (BOINC-сумісними) проектами розподілених обчислень. BOINC-клієнт дозволяє брати участь одночасно в декількох проектах за допомогою однієї загальної програми управленні (boinc або boinc.exe).

Для візуалізації процеса управління BOINC-клієнтом можна використовувати або дану програму по замовчуванню (офіційну програму-менеджер (boincmgr або boincmgr.exe)), або користуватися «неофіційною» програмою для моніторинга та управління BOINC-клієнтом. Слід замітити, що сам BOINC-клієнт в академічному розуміння не має користувацького інтерфейса як такого, а представляє собою сервіс, запуск якого відбувається при запуску системи та управляється за протоколом TCP/IP. Навпаки для кінцевого користувача це не має значення, оскільки дистрибутив програми комплектується програмою-менеджером, яка відразу за замовчуванням установлюється разом з BOINC-клієнтом як єдине ціле та абсолютно прозора для користувачів. В цьому випадку в якості адресу управляючого програмою-менеджером BOINC-клієнта вказується адреса «localhost».

Таким чином, з однієї сторони, ніщо не заважає користувачу використовувати альтернативну програму-менеджер для управління BOINC-клієнтом, а з іншої сторони дає можливість управляти декількома BOINC-клієнтами, які знаходяться на різних комп’ютерах з однієї програми-менеджера. Також така організація управління BOINC-клієнтом наштовхує на

можливість використання BOINC-клієнт в «невидимому» режимі, коли запускається виключно сервіс, без користувацького інтерфейса взагалі.

Настройки. У більш ранніх версіях клієнта відсутні локальні налаштування програми. Майже всю конфігурацію (наприклад, час роботи, час з'єднання, максимальне завантаження і т.д. і т. п .) Учасник вказує на сайті конкретного проекту (для кожного проекту окремо), а оболонка (клієнт) самостійно завантажує конфігурацію разом із завданнями по мірі необхідності. Однак в останніх версіях це можна налаштувати через інтерфейс самого клієнта.

Організація проектів. Створити проект на платформі BOINC може будь-хто - вся платформа BOINC спочатку розроблялася в рамках LGPL, тому будь-хто може ознайомитися з вихідними текстами. В основному цим займаються різні університети і наукові центри для вирішення завдань, що вимагають великих обчислювальних ресурсів, але не мають необхідних матеріальних засобів для покупки суперкомп'ютерів, або потужностей сучасних суперкомп'ютерів недостатньо для вирішення поставленого завдання.

Попри те що, система була розроблена для наукових цілей інституту в Берклі, з часом її широко почали використовувати й інші дослідницькі центри, наукові інститути. Найпопулярнішими проектами під управлінням BOINC є:

1. SETI@home - SETI (Пошук Позаземного Розуму (Search for Extraterrestrial Intelligence)) - область науки, чиєю метою ставиться знаходження розумного позаземного життя. Один з методів, відомий як «радіо SETI», полягає в використанні радіотелескопів для прийому вузькосмугових сигналів з космосу. Сигнали, які не характерні для природних явищ, будуть служити доказом використання позаземних технологій.

2. Rosetta@home - проект спрямований на обчислення тривимірної структури білків, що досить схоже на вирахування тривимірної структури генетичного ланцюга людини. Подібні дослідження можуть привести до створення ліків від таких захворювань як ВІЛ, малярія, рак і хвороба Альцгеймера.

3. Einstein@Home - проект спрямований на визначення місцезнаходження пульсарів, використовуючи дані лазерно-інтерферометричної гравітаційно-хвильової обсерваторії (LIGO), радіотелескопа Аресібо, космічного гамма-телескопа Фермі (GLAST).

4. Climate Prediction - проект прораховує різні симуляції кліматичних моделей, що дозволяє спрогнозувати, як зміниться погода на Землі в майбутньому.

5. MilkyWay@Home - проект спрямований на створення високоточних тривимірних моделей Потоку Стрільця, що дає інформацію про те, як сформувався Чумацький Шлях і як утворюються приливні рукави під час зіткнення галактик.

6. LHC@Home- підпроект SixTrack, створений для допомоги вченим поліпшення роботи ВАК, прораховує різні траєкторії 60 частинок, при яких промінь збереже стабільність в прискорювачі. Кількість циклів від 100000 до мільйона циклів, що відповідає менше 10 секундам реального часу. Цього достатньо, щоб перевірити чи пучок зберігати траєкторію протягом набагато більшого часу або існує ризик втрати стабільності пучка, що може привести до серйозних проблем в реальності, наприклад, до зупинки прискорювача або до виходу з ладу деяких детекторів.

7. Asteroids@home - проект має на меті збільшити обсяг інформації про фізичні характеристики астероїдів. Програма обробляє дані фотометричних спостережень різними приладами за різний час. Ця інформація перетворюється методом інверсії кривої блиску, що дозволяє створити 3D-модель форми астероїда разом з визначенням періоду і напрямком обертання навколо своєї осі.

Останні дані свідчать, що в мережі BOINC налічується близько 300 тисяч активних учасників, що в сумі дає більше 9 мільйонів комп'ютерів і продуктивність більше 8 петафлопс.

1.3.4 Система HTCondor

HTCondor - це відкрите програмне забезпечення для високопродуктивних обчислювальних програм для грубозернистої розподіленої розпаралелювання обчислювальних завдань. Він може бути використаний для управління робочим навантаженням у виділеному кластері комп'ютерів або для виведення роботи на прості настільні комп'ютери - так звану циклічну очищення. HTCondor працює на операційних системах Linux, Unix, Mac OS X, FreeBSD та Microsoft Windows. HTCondor може інтегрувати обидва виділені ресурси (стійки на кластери) та невидані настільні комп'ютери (циклічне викачування) в одне обчислювальне середовище.

HTCondor розроблена командою HTCondor в Університеті Вісконсін-Медісон і вільно доступна для використання. HTCondor іде за філософією відкритого джерела та має ліцензію під ліцензією Apache 2.0. Його можна завантажити з веб-сайту HTCondor або встановити Fedora Linux Distribution. Він також доступний на інших платформах, таких як Ubuntu з репозиторіїв.

HTCondor - це один з механізмів планувальника завдань, який підтримується GRAM (Grid Resource Allocation Manager), компонентом інструментарію Globus. Незважаючи на те, що компанія HTC Mondor ефективно використовує невикористаний обчислювальний час, залишаючи комп'ютери увімкнуті для використання разом із HTCondor, збільшиться споживання енергії та пов'язані витрати. Університет Ліверпуля продемонстрував ефективне рішення цієї проблеми, використовуючи суміш Wake-on-LAN та комерційного керування енергією PowerMAN (Програмне забезпечення). Починаючи з версії 7.1.1, HTCondor може перезапустити і будити машини на основі правил, визначених користувачем, без необхідності сторонніх програм.

Як приклад, база даних NASA Advanced Super Computing (NAS) від HTCondor складається з приблизно 350 SGI та Sun робочих станцій, придбаних та використаних для розробки програмного забезпечення, візуалізації,

електронної пошти, підготовки документів тощо. Кожна робоча станція запускає демон, що дивиться користувацькі введення / виводу і завантаження процесора. Коли робоча станція працювала впродовж двох годин, робота з партійної черги призначається на робочу станцію і буде запускатися, доки демон не виявить натискання клавіші, руху миші або високий рівень використання не-HTCondor процесора. На цьому етапі робота буде вилучена з робочої станції і розміщена назад у черговій партії.

HTCondor може запускати як послідовні, так і паралельні завдання. Послідовні завдання можуть виконуватися в декількох різних "всесвітах", включаючи "ваніль", яка забезпечує можливість запуску більшості "пакетних готових" програм, а також "стандартного універсуму", в якій цільова програма повторно пов'язана з бібліотекою введення / виводу HTCondor що забезпечує віддалене введення / виведення робочого місця та перевірку місця роботи. HTCondor також надає "локальну всесвіт", що дозволяє працювати на "відправці хоста".

У світі паралельних робочих місць компанія HTCondor підтримує стандартні MPI та PVM (Goux, et al., 2000) на додаток до власної бібліотеки Майстра робітника "MW" для надзвичайно паралельних завдань.

HTCondor-G дозволяє роботам HTCondor використовувати ресурси, які не перебувають під його безпосереднім контролем. В основному він використовується для спілкування з Grid та Cloud ресурсами, такими як pre-WS та WS Globus, ARC Nordugrid, UNICORE та Amazon EC2. Але це також може бути використано для спілкування з іншими партійними системами, такими як Torque / PBS та LSF. Підтримка Sun Grid Engine знаходиться в стадії розробки в рамках проекту EGEE.

HTCondor підтримує роботу DRMAA API. Це дозволяє DRMAA-сумісним клієнтам представляти та відстежувати роботу HTCondor. Реалізація довідки SAGA C ++ надає плагін HTCondor (адаптер), який забезпечує доступність та контроль роботи HTCondor за допомогою API Python та C ++ SAGA.

1.3.5 Система Rainbow

Технологія динамічної часткової реконфігурації в поєднанні з операційною системою реконфігурованих систем (OS4RS) дозволяє реалізувати концепцію апаратних завдань, тобто активний обчислювальний об'єкт, який може претендувати на реконфігурувані обчислювальні ресурси та запитувати служби ОС таким чином, щоб виконувати програмне завдання в звичайна ОС. Повну модель та реалізацію легкого OS4RS, що підтримує попередньо вирішальні та масштабовані апаратні завдання. Також це новий, легкий механізм планування, що дозволяє своєчасно і резервувати першочергові резервування реконфігурируемых ресурсів, метою якого є використання переваги лише в той час, коли воно приносить користь для роботи системи.

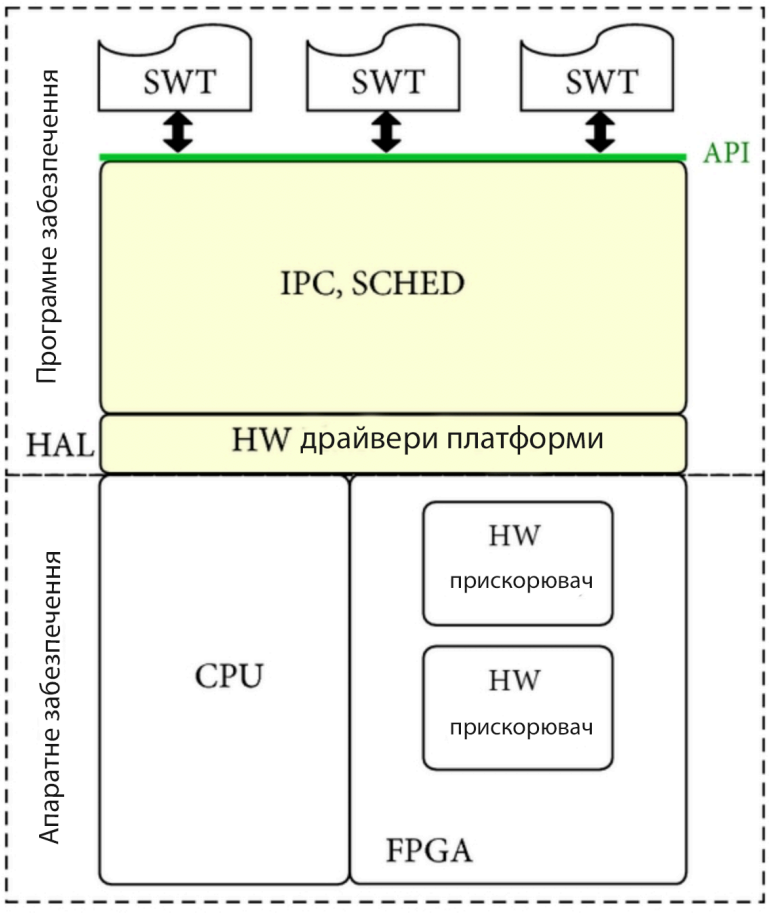

Рис. 1.6 Багатофункціональна FPGA система SW-HW на Rainbow

Розширювач програмного забезпечення з кодуванням, що називається Rainbow, йде за багатопрофільною архітектурою як у програмному, так і в апаратному забезпеченні, що покращує повторне використання та переносимість коду та допомагає групувати різні служби ОС. Отриманий вигляд розробленої архітектури OS4RS показаний на рис. 1.6. HW Task Management and Communication Layer управляє виконанням високого рівня виконання завдань HW, що відображає виклики API, створені шляхом взаємодії завдань SW та HW, а також міжгалузевої комунікації та синхронізації. Стан завдання SW керується самостійно, за базовим ядром ОС. Планування та розміщення планів завдань HW відповідає за планування розподілу, вилучення та попередження завдань HW, а також процес їх динамічного масштабування за часом. Стан завдання SW керується самостійно, за базовим ядром ОС. Планування та розміщення планів завдань HW відповідає за планування розподілу, вилучення та попередження завдань HW, а також процес їх динамічного масштабування за часом.

Програмна частина розширення виконується на ЦП разом з ядром базової ОС, тоді як апаратна частина реалізується на FPGA. Апаратна частина складається з контролера конфігурації, який належить HW Task Configuration Layer і HW Task Wrappers, модулі яких логічно належать до всіх трьох шарів. Конфігураційний контролер дає фізичні засоби розподілу та попередження завдань HW шляхом надання інтерфейсу для доступу до CM FPGA. HW Task Wrapper містить контролер динамічної часткової реконфігурації (DPR), який також належить до того самого шару, що і конфігураційний контролер. Він обробляє фізичні аспекти низького рівня технології DPR, пов'язані з розподілом і попередженням завдання.

Контролер динамічного масштабування (DFS) - це інший модуль, який належить до рівня HW Configuration Layer. Він забезпечує доступ до функції масштабування тактової частоти завдань HW.

1.3.6 Система AppScale

Метою AppScale є уможливити дослідження та експерименти у хмарних обчисленнях та полегшити моделювання програмування "писати один раз, працювати скрізь" для хмар, тобто пришвидшити переносне розроблення додатків і розгортання через неоднорідні хмарні тканини. API AppScale та способи, якими користувачі можуть розгортати Clouds та програми AppScale у загальнодоступних та приватних налаштуваннях. Окрім того, вони описують внутрішні особливості системи, щоб дати уявлення про те, як розробники можуть досліджувати та розповсюджувати AppScale як частину досліджень та розробки обласного програмного забезпечення та послуг нового покоління.

AppScale - це масштабована, розподілена та відмовостійкість системи керування хмарою, яку ми розробляли в Каліфорнійському університеті в Санта-Барбара в рамках нашого дослідження в системах програмування наступного покоління. Зокрема, AppScale - це хмарна платформа, тобто платформа-як-служба (PaaS) хмарної технології, яка виконує над ресурсами кластеру. Ресурси кластерів, що лежать в основі AppScale, можна керувати візуалізацією або без нього, наприклад, Xen, KVM, або через популярні хмарні інфраструктури, включаючи Amazon EC2 та Eucalyptus.

Платформа AppScale здійснює віртуалізацію, тези та мультиплексування хмарних та системних служб у кількох програмах, що дає змогу створювати програми Cloud-on-place, Run-Anywhere (WORA). Окрім спрощення розробки та розгортання додатків за допомогою хмарних систем та сучасних розподілених обчислювальних технологій, AppScale поширює популярні публічні хмарні тканини на замовлення, тобто приватні кластери.

Щоб це дозволити, ми імітуємо ключові хмарні шари у комерційному секторі та роблять це для зародження спільноти користувачів, для отримання доступу до реальних додатків та вивчення їх можливостей та розширення для загальнодоступних хмарних систем, джерельних технологій.

Початкові API, які емулюють AppScale, - це програми Google App Engine. Google App Engine - це загальнодоступна хмарна платформа (повна стека програмного забезпечення), яка експортує масштабовані та еластичні технології веб-сервісу за допомогою чітко визначених API для програм, доступ до мереж. Ці API застосовують обмін повідомленнями, зберігання даних з ключовими значеннями, скорочення карти, пошту та аутентифікацію користувачів, серед інших служб. Платформа полегшує легкий асинхронний багатозадачність, підтримка веб-серверів, еластичність та управління ресурсами.

Використовуючи Google App Engine, розробники налагоджують та перевіряють свої програми, використовуючи відкритий вихідний комплект розробника програмного забезпечення (SDK), який надає Google, який реалізує не масштабовані версії API. Розробники потім завантажують свій код і дані в кластери Google і використовують ресурси та служби кластерів Google на основі безкоштовного (до певного фіксованого набору квот для кожного ресурсу) та зарплати за використання (оренду ресурсів).

AppScale API сумісний з Google App Engine. Таким чином, програми, які виконуються в Google App Engine, також можуть виконуватися в AppScale без змін, використовуючи приватні ресурси кластерів або загальнодоступні хмарні інфраструктури. Наші API та службові реалізації є масштабованими, розповсюдженими, нестабільними, а також забезпечують високопродуктивний та високодоступний доступ до послуг. Щоб це зробити, ми використовуємо зрілі технології з відкритим кодом як можна ширше.

AppScale реалізує декілька мовних інтервалів (Java, Python, Ruby) як інтерфейс програми та широкий спектр технологій баз даних з відкритим вихідним кодом (ключ-значення та реляційні) як параметри для свого внутрішнього, загальносистемного хранилища даних. AppScale - це надійна і розширювана інфраструктура досліджень та приватна хмарна платформа, яка забезпечує ресурси, які вона розподіляється масштабно в різних програмах.

1.3.7 Система Fog computing

Fog computing можна сприймати як у великих хмарних системах, так і в великих структурах даних, вказуючи на зростаючі труднощі в доступі до інформації об'єктивно. Це призводить до недостатньої якості отриманого контенту. Ефекти обчислення туману на хмарних обчисленнях та великих системах даних можуть відрізнятися. Проте загальним аспектом є обмеження точного розповсюдження вмісту, питання, яке було вирішено за допомогою створення показників, які намагаються підвищити точність.

Fog network складається з контрольної площини та площини даних. Наприклад, на площині даних комп'ютерна обробка дозволяє комп'ютерним службам перебувати на краю мережі на відміну від серверів в центрі обробки даних. Порівняно з хмарними обчисленнями, fog computing підкреслює близькість до кінцевих користувачів і цілей клієнта, щільне географічне розподіл та об'єднання локальних ресурсів, зменшення затримки та економію пропускної здатності магістральних каналів для досягнення кращої якості сервісу (QoS) та аналізу краю / видобування потоку, що призводить до вищої якості користувальницький досвід та надмірність у разі виникнення помилки, тоді як його також можна використовувати в сценаріях асистентного життя.

Fog network підтримує концепцію "Інтернету речей" (IoT), в якій більшість пристроїв, що використовуються людьми щодня, будуть пов'язані один з одним. Приклади включають в себе телефони, пристрої для керування охороною здоров'я, пов'язані автомобілі та розширені реалії, використовуючи такі пристрої, як Google Glass. SPAWAR, підрозділ військово-морського флоту США, є прототипом та тестуванням масштабованої, безпечної мережевої сітки для руйнування, яка захищає стратегічні військові ресурси, як стаціонарні, так і мобільні. Програми керування машинами, що працюють на вузлах сітки, "перебирають", коли втрачається інтернет-з'єднання. Використання випадків включає в себе Інтернет речей, наприклад спритні свердловини.

Як хмарні обчислення, так і fog computing забезпечують зберігання, додатки та дані кінцевим користувачам. Проте fog computing має близькість до кінцевих користувачів та більший географічний розподіл.

Cloud Computing - це практика використання мережі віддалених серверів, розміщених в Інтернеті, для зберігання, керування та обробки даних, а не локального сервера або персонального комп'ютера. Cloud Computing може бути важкою і щільною формою обчислювальної потужності.

Fog computing - термін, створений компанією Cisco, що стосується розширення хмарних обчислень до краю мережі підприємства. Також відомий як Edge Computing або туман, обчислення туманів полегшує роботу обчислювальних, зберігання та мережевих сервісів між кінцевими пристроями та центрами обробки даних хмарних обчислень. Хоча граничні обчислення, як правило, посилаються на місце, де послуги інсценируются, обчислення туманів передбачає розподіл ресурсів та послуг зв'язку, обчислень та зберігання на пристроях і системах або поблизу них, що контролюються кінцевими користувачами. Fog computing має середній вага та середній рівень обчислювальної потужності. Замість того, щоб замінити дані, туманні обчислення часто служать доповненням до хмарних обчислень.

Fog computing - легка та рудиментарна форма обчислювальної потужності, яка знаходиться безпосередньо в мережевій тканині на крайньому краю мережевої тканини, використовуючи мікрокомп'ютери та мікроконтролери для подачі в вузли туману та потенційно далі у напрямку до платформ Cloud Computing. Національний інститут стандартів і технологій у березні 2018 року випустив визначення "Туманових обчислень", що приймає більшу частину комерційної термінології компанії Cisco як "NIST Special Publication 500-325", Концептуальну модель "Туманних обчислень", яка визначає обчислення туманів як парадигму горизонтального, фізичного або віртуального ресурсу, що знаходиться між розумними кінцевими пристроями та традиційними хмарними обчисленнями або центром обробки даних.

1.3.8 Система Plan 9

Plan 9 - це розподілена операційна система, створена в Центрі досліджень обчислювальної науки (CSRC) в Bell Labs в середині 1980-х років, і побудована на концепціях UNIX, які вперше були розроблені в кінці 1960-х років. Остаточний випуск був на початку 2015 року. Вона призначена для створення мережі гетерогенних та географічно розділених комп'ютерів як єдиної системи. У типовому Plan 9, користувачі працюють на терміналах під управлінням вікна системи *rio*, і вони отримують доступ до процесорних серверів, які обробляють обчислювальні процеси. Постійне зберігання даних забезпечується додатковими мережевими хостами, що діють як файлові сервери та архівні сховища.

Початкова ідея означає, що, на відміну від більшості операційних систем, процеси (запущені програми) мають власний погляд на простір імен, що відповідає тому, що інші операційні системи викликають файлову систему; назва одного шляху може містити посилання на різні ресурси для різних процесів. Потенційна складність цього налаштування контролюється набором звичайних місць для спільних ресурсів.

Наступна ідея означає, що процеси можуть пропонувати свої послуги іншим процесам, надаючи віртуальні файли, що з'являються в просторі назв інших процесів. Введення / виведення клієнтського процесу на такий файл стає між процесами зв'язку між двома процесами. Таким чином, Plan 9 узагальнює поняття Unix на файловій системі як центральну точку доступу до обчислювальних ресурсів. Він переносить ідею Unix про файли пристроїв для забезпечення доступу до периферійних пристроїв (миші, знімні носії тощо) та можливості монтувати файлові системи, що знаходяться на фізично відмінних файлових системах, в ієрархічне простір імен, але додає можливість підключення до сервера програма, яка говорить стандартизований протокол і розглядає свої послуги як частину простору імен.

Розподілена операція Plan 9 також залежить від простору імен кожного процесу, що дозволяє клієнтським та серверним процесам спілкуватися між комп'ютерами, як описано вище. Наприклад, команда *cpu* запускає віддалений сеанс на обчислювальному сервері. Команда експортує частину свого локального простору імен, включаючи пристрої термінала користувача (mouse, cons, bitblt), на сервер, щоб віддалені програми могли виконувати введення / виведення за допомогою миші, клавіатури та дисплея термінала, об'єднуючи ефекти віддаленої логіни і загальна мережева файлова система.

Всі програми, які хочуть надавати послуги-як-файли в інші програми, говорять про уніфікований протокол, який називається 9P. У порівнянні з іншими системами це зменшує кількість користувальницьких інтерфейсів програмування.

9P - загальний, середньоагностичний, байтовий протокол, який забезпечує повідомлення, що поставляються між сервером і клієнтом. Протокол використовується для позначення та взаємодії з процесами, програмами та даними, включаючи як користувальницький інтерфейс, так і мережу. З випуском 4-го видання, він був змінений і перейменований на 9P2000.

На відміну від більшості інших операційних систем, Plan 9 не надає спеціальних програмних інтерфейсів (таких як сокети Berkeley, ресурси X або системні виклики ioctl) для доступу до пристроїв. Замість цього драйвери пристроїв Plan 9 реалізують свій інтерфейс керування як файлової системи, так що для апаратного забезпечення доступ до звичайних операцій вводу / виводу файлів читається і записується. Отже, спільне використання пристрою через мережу може бути виконане шляхом встановлення відповідного дерева каталогів на цільову машину.

Розповсюджений пакет для Plan 9 містить спеціальні варіанти компіляторів та мови програмування, а також надає спеціальний набір бібліотек разом із віконною системою користувальницького інтерфейсу, специфічною для Plan 9.

1.4 Вибір методу балансування

Серед розглянутих вище систем, потрібно обрати реалізацію методу балансування, який буде максимально задовольняти умови, поставлені перед гетерогенною розподіленою системою та реалізовувати всі необхідні інтерфейси для використання MapReduce.

Таблиця 1.1

Порівняння методів балансування в розглянутих системах

Методи балансування в системах	Сумісність з MapReduce	Перерозподіл завдань на інші вузли	Попередня оцінка часу виконання завдання на вузлі	Реалізація методу як окремого модуля	Сума
	10	8	5	3	
Unicore	-	-	+	-	5
Hadoop	+	+	-	+	21
BOINC	-	+	-	+	11
HTCondor	+	-	+	-	15
Rainbow	-	-	+	+	8
AppScale	-	-	+	-	5
Fog Computing	+	+	-	-	18
Plan 9	+	-	+	-	15

З таблиці зрозуміло, що найбільш відповідним методом балансуванням володіє система Hadoop тому, що вона набрала найбільшу суму балів – відповідно задовольняє основним вимогам, поставленим до методу балансування який потрібно реалізувати в системі.

Висновки

1. Найбільш відповідним методом балансуванням володіє система Hadoop тому, що вона виконує одну з найважливіших вимог, а саме можливість її інтеграції з MapReduce, адже саме він використовується в системі як основний метод ділення та склеювання завдань.

2. Перерозподіл завдань між вузлами є також досить важливою вимогою до метода балансування, адже саме цей пункт надає можливість системі адаптуватися під змінну топологію мережі.

3. Hadoop є досить динамічною та гнучкою системою розподілених обчислень. Вона дозволяє різнопланові підходи до вирішення завдань та розподілу ресурсів. Також завдяки парадигмі MapReduce яку легко адаптувати під різноманітні завдання, цю систему можна назвати достатньо універсальною. Але сам метод балансування системи Hadoop не має можливості для виконання на мобільних платформах через немале споживання ресурсів та несумісністю платформ.

РОЗДІЛ 2
РЕАЛІЗАЦІЇ МЕТОДІВ БАЛАНСУВАННЯ

Розділ розглядає опис створених методів, результати їх застосування в розподілених системах. Також приведені деталі реалізації окремих алгоритмів. Завданням розділу є адаптація обраного методу до задач поставлених перед гетерогенною розподіленою системою з використанням мобільних вузлів.

2.1 Дослідження завдань по обслуговуванню вузлів

Для дослідження завдань з обслуговування вузлів припустимо, що число агрегованих по класах або пріоритетам потоків в структурі відомо і дорівнює M, що відповідає прийнятим на практиці рішенням в рамках відомих методів маркування завдань. Також приймемо до уваги, що максимальне число завдань у мережевих пристроях також відомо і становить (N). Наприклад , в алгоритмі пріоритетного обслуговування PQ виділяється 4 черги , в алгоритмі зваженого справедливого обслуговування WFQ не перевищує 256 , а в алгоритмі CBWFQ дорівнює кількістю класів обслуговуваних потоків і дорівнює 64 . Таким чином, позначимо через a_i (i =1,2, ...M) – інтенсивність вхідного трафіку і -го класу, який надходить на обслуговування мережевим пристроєм. Крім цього, нехай b_j (j =1,2, ... N) – частина пропускної здатності вихідного каналу зв’язку, яка є виділеною для j - ї черги (j =1, ... N), що є типовим для алгоритму CBWFQ . Одне з ключових відмінностей пропонованого рішення полягає в тому, що змінні b_j (j= 1, ... N) у нашому випадку матимуть змогу при можливості розраховуватись динамічно з контролем за часом перебування в черзі пріоритетних пакетів, адаптуючись до зміни стану мережного вузла, а не адміністративно, як, наприклад, в CBWFQ. Під час управління чергами забезпечити виконання умови (2.1) відсутності перевантаження каналу зв'язку в момент часу t:

$$\sum_{j=1}^{n} b_j(t) \le b \ , \quad (2.1)$$

де b - пропускна здатність вихідного каналу зв’язку, $b_j(t)$ – частина пропускної здатності вихідного каналу зв’язку, яка є виділеною для j - ї черги в момент часу t (j =1, ... N). Крім того, з метою запобігання перевантаження мережевого пристрою в момент часу t потрібно, щоб виконувалася наступна умова:

$$\sum_{i=1}^{m} a_i(t) \leq b, \quad (2.2)$$

Виконання умови (2.2) забезпечується шляхом превентивного обмеження інтенсивності вхідного трафіку, що поступає на мережевий пристрій телекомунікаційної мережі, з метою не перевищення пропускної здатності вихідного каналу зв’язку. Це функція на практиці виконується алгоритмами довільного раннього виявлення перевантаження та обмеження довжини черги RED і WRED. Забезпечити динамічний характер процесу обслуговування черг в рамках запропонованого алгоритму можна шляхом введення керуючої змінної x_{ij} , під якої мовити частка і -го трафіку, що надходить для обслуговування в j - ю чергу.

Виконання умови гарантує відсутність втрат пакетів на досліджуваному мережевому пристрої. Умова вводиться для уникнення перевантаження пропускної здатності каналу зв’язку, що виділяється для передавання пакетів тієї чи іншої черги мережевого пристрою в процесі управління ресурсами. У ході виконання умов за рахунок непередбачуваної зміни характеру мережевого трафіку на вузлі виникають черги та пов'язані з ними затримки пакетів. Для забезпечення допустимих затримок у вузлах вибирають оптимальну загальну буферну ємність. На цьому етапі починають з’являтися проблеми щодо вибору оптимальних відносно мінімальних затримок та втрат пакетів ємностей мережевих буферів. Таким чином, для кожної черги визначимо її поточну завантаженість і максимальну ємність, позначивши їх відповідно через n_j і n_j^{max} (j = 1, ... , N). Запишемо умови запобігання перевантаження окремих черг по їх пропускній здатності умовами уникнення перевантаження черг по їх довжині і тепер завдання зводиться лише до вибору

аналітичного виразу для розрахунку середньої довжини черги в процесі обслуговування пакетів у мережевих вузлах.

При цьому кожному типу трафіку, а значить і кожній черзі, може відповідати своя модель обслуговування, не обов'язково відповідаючи переліком існуючих варіантів СМО. З точки зору забезпечення гарантій QoS за параметром середньої затримки в ряді випадків зручніша сумарна умова (2), де середня затримка обслуговування в тій чи іншій черзі може бути розрахована за відомою середній довжині черги на основі формули Літтла для будь-якої СМО. Використання системи умов є актуальною у випадку, коли чисельні значення необхідної середньої затримки (як параметра QoS) нормовані по окремих ділянках мережі. Тоді в процесі управління чергами важливо не перевищити ці задані для окремо взятої пари вузлів каналу значення середньої затримки пакетів, що особливо характерно при вирішенні задач щодо забезпечення гарантованого QoS в рамках архітектурної моделі досліджуваної мережі.

2.2 Модель системи оброблення завдань

Представлений алгоритм керування завданнями, що базується на одному з відомих алгоритмів CBWFQ та запропонованого методу. Розробка алгоритму полягає у забезпеченні максимальної доступності високопріоритетної послуги зв’язку при збереженні ефективного розподілу мережного ресурсу іншим потоком, який забезпечується зваженим механізмом кругового обслуговування черг. З метою підвищення якості обслуговування та мінімізації втрати пакетів інформації вводиться контроль за часом перебування в черзі пріоритетних пакетів. Узагальнений алгоритм обслуговування трафіку для моделі системи з m пріоритетними групами виглядатиме аналогічно. Утворюється m черг відповідно до пріоритетної групи. На початку кожної ітерації передачі інформації для кожного пакету пріоритетних черг оцінюється час перебування в черзі, якщо час перебування в черзі більше ніж Δt_m , тоді ці пакети класифікуються як прострочені та

ставляться в першу позицію найкоротшої черги, що має пріоритет менший. Такі дії виконуються послідовно для всіх прострочених пакетів всіх пріоритетних груп. Отже, розміри черг повинні бути такими, щоб виконувалась умова, що час який буде чекати пакет, що останній став в чергу, повинен бути не більше Δt_m^{max}, (часу максимальної затримки рафіку p-ї пріоритетної групи).

Після того, як розмір черг був встановлений є потреба роз'яснити поведінку системи в цілому. При надходженні пакетів до вузла вони ставляться в чергу відповідно до пріоритетної групи. Як вже було з'ясовано для пріоритетних пакетів задаються інтервали часу Δt_m та Δt_m^{max} , на основі даних про них формуються черги. Відомо, що $\Delta t_m < \Delta t_m^{max}$. Пакети, що знаходяться в черзі довше Δt_m вважаються простроченими. Для того, щоб на кожній ітерації не виявляти прострочені пакети (виконання зайвих дій), є можливість ставити мітки тим пакетам, що потрапили в хвіст черги і явно стануть простроченими (отримають мітку lim), якщо будуть послідовно обслуговуватися в своїй черзі.

Тобто всі пакети, що потрапили на місця черг в інтервалі $[N_p(m), N_p^{lim}(m)]$, отримають мітку lim (прострочені). Якщо запропонувати перенесення цих пакетів одразу після їх надходження до черг наслідування, тоді збережеться цілісність задачі, суттєво спростяться розрахунки часу затримки повідомлень, та зменшиться кількість втрачених пакетів з пріоритетних груп, тому що останні позиції черг пріоритетних пакетів будуть порожніми. Робота запропонованого алгоритму для системи з трьома чергами: відео потоків (Video), голосових потоків (Voice) та потоків Інтернет трафіку (Data). Група черг наслідування складається з черги Інтернет трафіку. Група наслідування складається з черг непріоритетних пакетів або низько пріоритетних пакетів. Якщо Інтернет трафік класифікований деяким чином, і пакети різних класів Інтернет трафіку мають різні пріоритети, тоді всі або деякі з них можуть потрапити до групи наслідування. В рамках даної роботи робиться

припущення, що події пов’язані з потраплянням пакетів до однієї з черг групи наслідування вважаються рівноймовірними.

Модель системи оброблення інформаційних потоків за удосконаленим алгоритмом управління черг у вузлах корпоративної мережі алгоритм дає змогу ефективно передавати пакети пріоритетних типів трафіку, при цьому не завдаючи суттєвої шкоди для передачі непріоритетних потоків.

2.3 Системи з удосконаленим алгоритмом управління завданнями

Модифікований метод управління завданнями в мультисервісних вузлах телекомунікаційної мережі. Одна з ключових відмінностей пропонованого рішення полягає в тому, що ведеться контроль за часом перебування пакетів в системі, в умовах перевищення допустимого часу очікування, пакет передається на чергу із нижчим пріоритетом та першочерговим обслуговування, що дає змогу підвищити ефективність розподілу мережевих ресурсів за критерієм якості обслуговування. Розроблено імітаційну модель маршрутизатора, який володіє функцією адаптивного вибору алгоритму обслуговування черг в умовах виникаючого явища випадкового сплеску трафіку характерних для мультисервісних мереж нового покоління. Адекватність розроблених моделей підтверджено на основі дослідження імовірнісних властивостей трафіку мультисервісної корпоративної мережі. В результаті проведення імітаційного моделювання доведено, що застосування розробленого методу управління трафіком у вузлах мережі надання інфокомунікаційних послуг в мережах нового покоління, призведе до покращення якості обслуговування потоків реального часу. А саме зменшено затримку обслуговування інформаційних послуг реального часу з кінця в кінець до 2 разів та зменшено ймовірність втрат пакетів на 3% із застосуванням удосконаленого алгоритму зваженого кругового обслуговування черг в мережевих вузлах мультисервісної мережі.

Це в свою чергу призводить до вимушеного відкидання пакетів згідного алгоритму WRED та виникнення значних втрат пакетів голосу, що вплине на

якість розмови користувачів в NGN мережах. Внаслідок застосування удосконаленого алгоритму та із вище запропонованою моделю обслуговування, зменшується кількість пакетів у буфері голосового потоку до допустимого значення. А прострочені пакети надсилаються у вільно доступні черги нищої пріоритетності із вищою пропускною здатністю, що в свою чергу призводить до зменшення втрат та середньої затримки обслуговування у вузлі NGN мережі у 2 рази. На практиці застосування запропонованого модифікованого алгоритму обслуговування черг у мережевих вузлах NGN мережі, дасть змогу зменшити тривалість оброблення пакетів та ймовірність втрат даних потоку реального часу із високим пріоритетом не погіршуючи якості низько пріоритетним.

Результат оцінки тривалості оброблення пакетів маршрутизатора рівня ядра, щляхом пінгування від сервера на якому встановлена система моніторингу. Внаслідок чого спостерігається затримка, що перевищує 10,2 мс та 2% втрат пакетів. Що при передаванні потоків реального часу є не допустим, якщо на шляху до адресата стоять аналогічні вузли з однотипним навантаження та кількість їх перевищує 10. Отже, забезпечити гарантовану якість потоків реального часу в даній мережі при кількості хопів більше 10 із застосуванням існуючих алгоритмах CBWFQ є неможливим. Що підтверджує доцільність впровадження у NGN мережах запропонованих рішень розглянутих у дисертаційній роботі.

2.4 Вимоги до інтерфейсу для розпаралелювання завдань

Система DDCI, що реалізує підхід прозорого автоматичного динамічного розпаралелювання обчислень для блокових послідовних алгоритмів, вже зарекомендовула себе як ефективне і зручне рішення для автоматичного перетворення послідовних алгоритмів в паралельні аналоги. Обмеження системи, пов'язані з розпаралелюванням тільки блокових послідовних алгоритмів, є обґрунтованими і не являють незручностей для програмування, що було доведено в теоретичних і експериментальних дослідженнях. У

системі, процес динамічного побудови графа распараллелівать програми в головній мірі залежить від обраного методу інтегрування DDCI-системи в код послідовних програм. Запропонований метод інтеграції DDCI-системи має багато спільного з підходом, використовуваним в Т-Системі, але ті відмінності, які присутні, вносять серйозні зміни в процеси динамічного побудови і розпаралелювання графа послідовної програми.

Розгляд проблеми динамічного побудови графа послідовної програми з вибору завдання, на якій наочно можна буде продемонструвати всі висунуті методи. Блоковий алгоритм твори двох квадратних матриць:

- має поліноміальних складність завдання $O(n^3)$ в залежності від розмірності матриць
- для завдання часто доводиться використовувати паралельну реалізацію
- блок-схему алгоритму легко аналізувати (рис. 2.1)

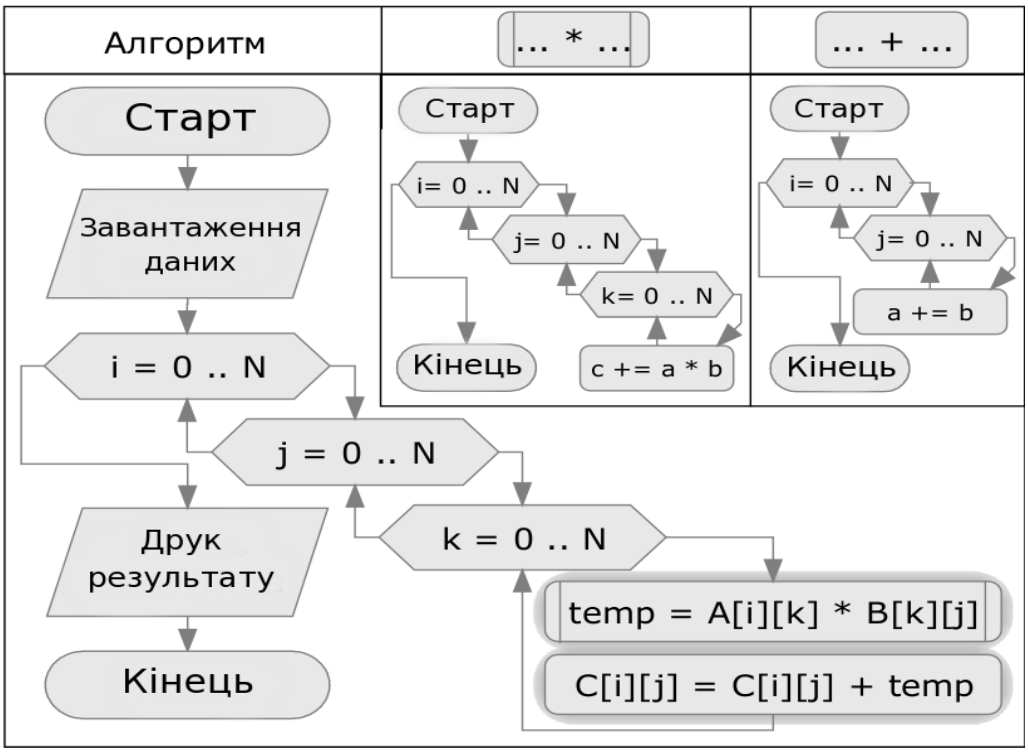

Рис. 2.1 Блок-схема блочного алгоритму перемноження двох квадратних матриць

На рис. 2.1 демонструються три блок-схеми, сумарно реалізують алгоритм твори двох матриць. Основна блок-схема програми позначена як «Алгоритм». Дві малих блок-схеми з лівої частини малюнка реалізують

алгоритми функцій, що відповідають за твір і складання двох матриць. Виклики даних функцій можна побачити в основний блок-схемі в правому нижньому кутку. Таке розбиття алгоритму на три блок-схеми не випадково, кожна частина алгоритму має свої унікальні особливості.

«Алгоритм», в подальшому «керуючий код». Дана частина алгоритму в після послідовно реалізації завдання відповідає в першу чергу за зв'язок виконуваної програми з різними локальними сховищами даних, наприклад: параметри запуску програми, доступ до консолі, доступ до файлової системи, мережеві протоколи, спеціалізоване дослідницьке обладнання. Наслідком є неможливість виконання деяких команд даної частини алгоритму на віддалених обчислювальних станціях, а, значить, і неможливість неконтрольованого автоматичного розпаралелювання програми.

Другою відмітною особливістю даної частини алгоритму є споживана обчислювальна потужність. Якщо в блок-схемі виклики розрахункових функцій

«Множення» і «Додавання» замінити заглушками, то стане видно, що в незалежності від розмірності матриць ця частина алгоритму завжди виконується однаково швидко і не вносить відчутної надбавки до часу роботи алгоритму в цілому. Такі особливості по даному конкретному прикладу означають тільки одне: розпаралелювання керуючої частини коду не є першочерговим завданням, так як це не принесе відчутного зростання в швидкості роботи паралельного аналога. Тепер можна зробити деякі проміжні висновки:

1. Автоматичне розпаралелювання розглянутої частини алгоритму ускладнене через проблеми, пов'язані з локальними дескрипторами ресурсів, що використовуються для зв'язку розпаралелювань послідовної програми із зовнішніми сховищами даних.
2. Існує велика кількість різновидів локальних дескрипторів, і навіть якщо більшість з них є стандартними, то завжди залишається можливість установки на комп'ютері спеціалізованого програмного забезпечення

або дослідницького обладнання, що використовує свої «унікальні» дескриптори. В результаті детектування локальних дескрипторів, не вводячи в інтерфейс розпаралелювання системи спеціальних директив, неможливо.

3. Якщо всі основні обчислення розпаралелювати блочного алгоритму знаходяться в чистих функціях, то виклики функцій можна замінити «заглушками», а автоматичне розпаралелювання цій частині алгоритму є необов'язковим.
4. Вимога винесення основних обчислень з основного графа програми в окремі чисті функції є не більше ніж частиною інтерфейсу рішення, а, значить, можна досягти для переважної числа блокових послідовних алгоритмів, що підлягають автоматичному розпаралеленню.

Результатом зроблених висновків є формулювання першого вимоги для послідовних алгоритмів: все автоматично динамічно розпаралелені програми повинні використовувати основну частину графа алгоритму для зв'язку з зовнішніми джерелами даних і використовувати мінімум обчислень, що не інкапсульованих в окремі чисті функції.

«Множення», в подальшому «розрахункові функції». У запропонованому завданні багаторазове обчислення функції, що відповідає за множення двох блоків від початкових матриць, займає близько 99% часу роботи всього алгоритму. Надалі, при розгляді розрахункових функцій, буде матися на увазі безліч помічених чистих функцій з послідовного блочного алгоритму будь-якого завдання, які виконують всі основні обчислення, і при цьому їх виконання займає близько 99% часу роботи всій завдання. Для всіх розрахункових функцій в межах реалізованого підходу також висуваються певні вимоги, і їх повний виклад вже наводилося в попередніх роботах. Коротко можна сказати, що розглянута операція «перемноження блоків матриць» повністю відповідає всім висунутим раніше вимогам:

1. Враховано всі обмеження для «чистої функції», згідно з якими функція може працювати тільки з тими даними, які були передані в якості параметрів.
2. Функція відповідає всім нормам по «ефективності чистих функцій», так як кількість переданих даних в функцію зростає квадратично, а кількість обчислень - кубічно щодо боку перемножуєте квадратних матриць.

На закінчення можна сказати, що даний клас функцій найбільше потребує ефективного розпаралелювання, так як сумарний час роботи розрахункових функцій - це основна складова будь-якої распараллелівать завдання.

«Додавання», в подальшому сполучні функції. Такі функції в попередніх авторських роботах розглядалися як окремі випадки рідко використовуваних низько ефективних чистих функцій. На рішення виділити їх в окремий клас вплинули наступні моменти:

1. Такі функції завжди мають низьку ефективність, яка призводить до уповільнення роботи розпаралелювань програми, причому твердження вірне, тільки якщо вважати їх розрахунковими функціями і використовувати для них такі ж методи планування.
2. Так як складність використовуваного в них алгоритму близька до лінійної щодо кількості переданих даних, то немає явної необхідності обчислювати їх всього один раз і надалі використовувати результати їх роботи. Більш ефективним підходом для них можна вважати багаторазовий перерахунок на різних станціях, так як це не додасть суттєвих змін до спільного часу роботи всього алгоритму.
3. Відмова від обліку таких функцій як повноправних вершин графа розпаралеленої частини програми економить кількість використовуваної розподіленої пам'яті для зберігання проміжних розрахунків. Зменшення кількості розподіленої пам'яті, в свою чергу, призводить до зниження кількості міжпроцесорного трафіку.

В результаті можна зробити наступний висновок: виділення класу сполучних функцій може підвищити ефективність роботи автоматичного

розпаралелення програм за умови, що надасть права мати окремі методи планування для виконання функцій даного типу. Також це призводить до додаткових інтерфейсних вимог, які полягають в самостійної позначці програмістом всіх сполучних функцій в його послідовній програмі, так як система автоматичного розпаралелювання самостійно не зможе відрізнити сполучні і розрахункові функції.

2.5 Управління динамічним графом розпаралелювання

Розглянемо процес формування графа розпаралелювання частини послідовного алгоритму з боку планувальника. Планувальник влаштований так, що він є незалежною системою і пов'язаний з керуючою частиною коду через механізми формування і модифікації динамічного графа. Як говорилося раніше, керуючий код відповідає за процес динамічного додавання нових елементів в граф програми і при необхідності може очікувати закінчення виконання заданої вершини графа. Плануюча система відповідає за запуск чистих функцій (вершин графа) на доступних обчислювальних станціях, а також за подальше видалення вже порахованих елементів з графа. Плануюча система відповідає за запуск чистих функцій (вершин графа) на доступних обчислювальних станціях, а також за подальше видалення вже порахованих елементів з графа. При такому підході динамічно будується граф має багато схожого з буфером зворотного магазинного типу (FIFO). Різниця полягає лише в тому, що плануюча система не зобов'язана вибирати елементи динамічного графа в тому ж порядку, в якому вони були додані в граф. При цьому практичне використання системи показало, то для більшості блокових алгоритмів мінімізація міжпроцесорного трафіку і динамічне балансування навантаження призводять до того, що планувальник подає на виконання чисті функції приблизно в такому ж порядку, в якому вони додавалися в динамічний граф.

Тепер розглянемо граф паралельної частини алгоритму, який реалізує блочне твір двох квадратних матриць. Для випадку розбиття кожної матриці

на 4 однакових квадратних блоку граф буде виглядати наступним чином (рис. 2.2).

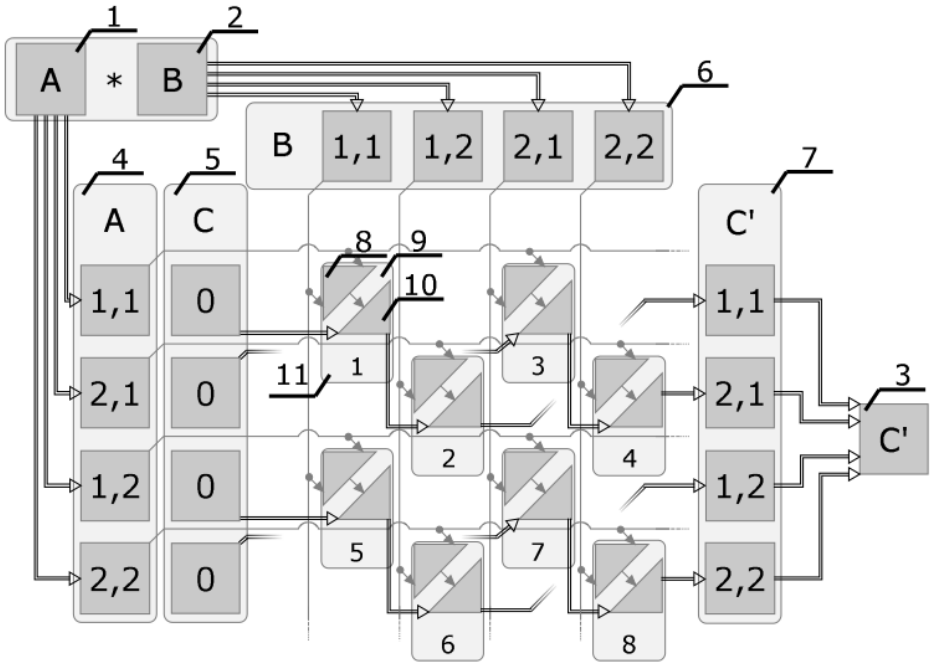

Рис. 2.2: Граф розпаралелювання частини блочного алгоритму перемноження двох матриць

Центральна частина графа, що складається з однотипних елементів під номером «9», формується в процесі виконання трьох циклів з «Міткою-3» в блоксхемі алгоритму. Позначки «8» і «10» відповідають функціям перемноження і складання двох квадратних блоків матриць відповідно. Порядок, в якому елементи будуть додаватися в динамічний граф програми, також показаний на схемі (нумерація з позначкою «11»). З порядку додавання елементів в динамічний граф відразу видно серйозна проблема, з якою стикається плануюча система. Проблема полягає в тому, що проведення будь-якого ефективного розподілу обчислень для всіх розрахункових станцій під час інтенсивного формування графа програми безглуздо, тому що згодом додавання нових елементів в динамічний граф програми призведе до несумісності кінцевого графа із запланованими раніше операціями.

Більшість сучасних систем динамічного розпаралелювання використовують для планування тільки пасивні алгоритми аналізу графа. Такі алгоритми проводять спрощену оптимізацію розподілу обчислень для

розрахункових станцій, так як проводити якісний аналіз по розпаралелюванню безглуздо через мінливої форми графа.

Набір елементів під номером «7» відповідає результатам перемноження двох матриць. Тут результат представлений у вигляді набору блоків, з яких складається результуюча матриця. Всі чотири блоки до закінчення обчислень будуть знаходитися в області розподіленої пам'яті на різних обчислювальних станціях. Об'єднання їх в цілісну матрицю під номером «3» проводиться на етапі блок-схеми з «Міткою-6». Процес перенесення даних з розподіленого сховища в область керуючого коду, як говорилося вище, включає в себе призупинення керуючого коду на тривалий проміжок часу. Саме цю «припинення» і пропонується використовувати для запуску активних алгоритмів розпаралелювання, так як саме в цей проміжок часу за зміни динамічного графа відповідає тільки планувальник.

2.6 Планувальник використання пасивних і активних алгоритмів

На основі системи динамічного розпаралелювання обчислень можна використовувати пасивні та активні алгоритми розпаралелювання в межах одного завдання. Проблему одночасного використання обох підходів з метою поліпшення ефективності роботи комплексного планувальника для динамічного розпаралелювання обчислень.

Виділимо два основних види ресурсів, за ефективне використання яких відповідає планувальник:

- обчислювальні ресурси включають в себе всю обчислювальну потужність використовуваної багатопроцесорної системи;
- транспортні ресурси включають в себе можливості транспортної системи, використовуваної для передачі даних між обчислювальними станціями.

З метою ефективного управління розглянутими ресурсами пропонується створити кілька черг з різним пріоритетом для можливості ефективного розподілу всіх наявних ресурсів і мінімізації простоїв.

Обчислювальні ресурси:

- високий пріоритет - використовується для всіх завдань, запланованих активним планувальником. Це необхідно для того, щоб активний планувальник в разі нестачі вільних обчислювальних станцій міг витісняти завдання пасивного планувальника;
- середній пріоритет - використовується для задач, запланованих пасивним планувальником. Черги високого і середнього пріоритетів використовують один і той же обчислювальний ресурс, і якщо чергу високого пріоритету непорожній, то все завдання з черги середнього пріоритету припиняються або можуть бути скасовані і перенесені на інші обчислювальні станції.

Транспортні ресурси:

- високий пріоритет - використовується для активного планувальника, дана чергу витісняє всі наступні черги;
- середній пріоритет - використовується для пасивного планувальника, а також для витіснення черзі низького пріоритету;
- низький пріоритет - черга використовується для фонового алгоритму дублювання розподілених даних. Черга активна під час простою каналів, коли черги високого і середнього пріоритетів порожні.

Для ефективного використання всіх ресурсів також пропонується розбити планувальник на три незалежні системи: активний планувальник, пасивний планувальник і система фонового дублювання даних.

Активний планувальник. Використовується тільки під час призупинення керуючого коду, так як тільки в цей момент часу можна проводити складний аналіз графа распараллелівать програми, не побоюючись за серйозні і непередбачені наступні зміни динамічного графа з боку керуючого коду. При цьому під час своєї роботи алгоритм може використовувати вільні обчислювальні ресурси, виділені раніше для виконання керуючого коду. Пропонується використовувати даний планувальник для ефективного

розпаралелювання тільки тієї частини динамічного графа, результати розрахунку якої очікує керуючий потік, а інша частина графа буде як і раніше распараллелівать пасивним планувальником. Такий підхід спрямований на мінімізацію часу простою керуючого коду і, отже, на мінімізацію часу роботи паралельної реалізації програми в цілому.

Пасивний планувальник. Використовується постійно і не використовує складні алгоритми аналізу графа. Пропонується для реалізації планувальника використовувати основні ідеї Т-Системи: планувальник повинен стежити, щоб якомога менше обчислювальних ресурсів простоювало. При цьому, на відміну від Т-Системи, пропонується ввести критерій, який буде відображати, скільки подальших розрахунків залежить від результатів виконання кожної окремої вершини графа. Критерій пропонується використовувати для того, щоб поліпшити початкову ідею Т-Системи за рахунок того, що на виконання спочатку будуть ставитися ті вершини графа, результатів виконання яких очікує більше число наступних чистих функцій. Таке нововведення дозволить змусити пасивний планувальник в першу чергу слідувати по шляху мінімізації діагоналі графа, що, в свою чергу, призведе до підвищення ефективності розпаралелювання.

Система фонового дублювання даних для розподіленого сховища. Метою алгоритму є кероване дублювання проміжних даних між обчислювальними станціями. Такий підхід, по-перше, підвищує ймовірність того, що велика частина даних, необхідна для запуску чистої функції, заздалегідь буде перебувати на потрібній обчислювальної станції. По-друге, при запланованій передачі даних між обчислювальними станціями з'являється можливість вибрати для передачі відсутніх даних обчислювальну станцію з найбільш низьким завантаженням транспортного каналу на даний момент, а не передавати дані з тією єдиною станції, на якій дані були пораховані, без можливості адекватної балансування навантаження на транспортні канали. При цьому такий алгоритм не заважає роботі пасивного і активного планувальників, так як використовує тільки простоюють транспортні ресурси.

Висновки

1. Розглянуто методи балансування навантаження в розподіленій системі, алгоритми перерозподілу завдань та прикладну програму (планувальник) для обробки вхідних завдань.

2. Проведено покращення та адаптацію методу для роботи в гетерогенній системі розподілених обчислень з використанням мобільних вузлів.

РОЗДІЛ 3
РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

В даному розділі викладено маркетинговий аналіз перспектив реалізації системи розподілених обчислень на мобільній платформі, а також оцінено можливості її ринкового впровадження.

3.1 Опис ідеї проекту

Проект направлений на підвищення використання розподілених систем обчислень за рахунок використання мобільних пристроїв в якості вузлів мережі. Така система володіє високим рівнем гетерогенності, завдяки властивостям які їй надає технологія MapReduce. В межах її функціоналу є можливість використовувати різнорідні вхідні данні для створення завдань на обчислення.

Таблиця 3.1

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Системи розподілених обчислень, що дає можливість проведення обчислень на мобільних пристроях	1.Розподілені обчислення	Можливість виконати необхідні обчислення паралельно на великій кількості пристроїв одночасно

При порівнянні з конкурентами в першу чергу увага надається архітектурному підходу, що забезпечує гетерогенність системи і дозволяє використовувати мобільні пристрої в якості вузлів, що виконують обчислення.

Також система дає змогу обчислювати різнотипові завдання за допомогою сучасного алгоритму ділення та склеювання завдань.

Порівняння з конкурентами, а також визначення переваг і недоліків наведено у наступні таблиці.

Таблиця 3.2

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

No п/п	Техніко- економічні характеристики ідеї	(потенційні) товари/концепції конку-рентів			W (слабка сторона)	N (нейтра - льна сторона)	S (сил ьна стор она)
		Мій проек т	Hado op	Google Cloud Platform			
1	Можливість переконатись у тому чи зараховано голос користувача і кому саме	так	ні	так			так
2	Непередбачуваність зростання вартості підтримки системи в випадку розгортуваності в основній мережі	так	ні	ні	так		
3	Доступність проміжних результатів	ні	так	так			так
4	Використання різнотипових завдань	так	ні	ні			так

Конкуренти володіють лише частковим функціоналом, який реалізований в даному проекті. Серед сильних сторін визначені гетерогенність побудови системи та використання різнотипових завдань, що підвищує рівень розповсюдження додатку серед користувачів. Слабкою стороною є неможливість отримання проміжних результатів обчислень, через архітектуру системи, а саме реалізація паралельного обчислення на незв'язаних між собою вузлах мережі.

3.2 Технологічний аудит ідеї проекту

Таблиця 3.3

Технологічна здійсненність ідеї проекту

No п/п	Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
1	Розподілена система обчислень на мобільній платформі	Технологія MapReduce	Існуючі платформи, Hadoop, Google Cloud Platform, що дозволяють виконувати обчислення на ПК	Є доступними та безкоштовними для використання
2		Централізований мобільний застосунок	Мови програмування, фреймворки, що дозволяють побудувати веб-системи	Велика кількість фреймворків вільно розповсюджуються і є доступними
Обрана технологія реалізації проекту: MapReduce				

Для реалізації проекту доступні дві технології: традиційна централізована веб-система з реляційним або NoSQL сховищем даних та розподілена система з технологією MapReduce. Для виконання розподілених обчислень на ринку представлено декілька платформ, таких як Hadoop, Google Cloud Platform та інші, що дозволяють виконувати розподілені обчислення на ПК.

3.3 Аналіз ринкових можливостей запуску стартап-проекту

При дослідженні ринкових можливостей, в першу чергу проведений аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку. Дані наведені у таблиці нижче.

Таблиця 3.4

Попередня характеристика потенційного ринку стартап-проекту

No п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж	?
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу	Немає
5	Специфічні вимоги для специфікації	Немає
6	Середня норма рентабельності в галузі, %	?

Враховуючи сьогоднішню зацікавленість суспільства до технології MapReduce, розподілених обчислень та використання мобільних пристроїв у якості вузлів гетерогенних мереж, а також інтерес інвесторів до технології, за попереднім оцінюванням ринок є привабливим для входження.

Таблиця 3.5

Характеристика потенційних клієнтів стартап-проекту

No п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Обчислення даних	Приватні підприємства	Приватні підприємства в свою чергу можуть визначати самостійно формат взаємодії та внести свої коригування	● Впровадження систем розподілених обчислень ● Можливість швидко отримати результат ● Можливість встановлювати обмеження на розповсюдження певної інформації
2	Використання потенціалу мобільних пристроїв	Поодинокі користувачі	Поодинокі користувачі бажають використовувати простій пристрою з користю	● Встановлення мобільного додатку для обчислень ● Керування виконуваними завданнями

Таблиця 3.6

Фактори загроз

No п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Відсутність зацікавленості у організацій, які не прагнуть до розголоження корпоратиної інформації	Успіх системи багато в чому залежить від підтримки з приватних структур, адже з великою імовірністю організації, які мають в своїй роботі не прозорі процеси, не прагнуть до розголошення або викриття корпоративної інформації	Надання прикладів роботи з її перевагами рядовим працівникам, які в свою чергу будуть спонукати керівництво до впровадження даної системи
2	Новизна технології	Технологія MapReduce, недивлячись на успіх у сфері розподілених обчислень, ще є молодою і великі гравці ринку лише придивляються до неї. У великій мірі вони ще досліджують можливість реального її застосування і орієнтуються на успішні приклади впровадження, яких ще недостатньо	Розробка широкої інтеграції з існуючими популярними технологіями за для більш плавного впровадження системи

Таблиця 3.7

Фактори можливостей

No п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Споживча готовність населення	В наш час суспільство стрімко рухається в бік розподілених обчислень як можливості отримати результат швидше традиційних засобів	Підтримка надійного іміджу компанії, продовження політики інформування населення про можливості децентралізованої технології MapReduce
2	Зростання популярності технології MapReduce у світі	2017 рік став піком популярності технології MapReduce завдяки пожвавленню розподілених обчислень.	Висвітлення альтернативних сфер використання технології
3	Зростання швидкості процесу отримання результату	Технологія MapReduce, що лежить в основі стартап-проекту, є децентралізованою і дозволяє виконувати різнотипові завдання	Надати користувачам гарантію того що їх завдання буде конфіденційним і дані буде оброблено і збережено в належному вигляді

Одночасно і можливістю і загрозою є новизна технології MapReduce — вона набуває все більшої популярності серед суспільства, але використання її у інших сферах, окрім як для розподілених обчисленнях, все ще не стало достатньо широким.

Таблиця 3.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства
1. Тип конкуренції: олігополія	На ринку представлені декілька компаній, що поставляють подібні послуги керування даними	Акцентування переваг продукту, що забезпечує використання технології MapReduce
2. Рівень конкурентної боротьби: національний/інтернаціональний	Першим етапом є боротьба за ринок України, де послуги конкурентів не запропоновані з подальшим виходом на ринки інших країн	Маркетингова компанія в першу чергу орієнтована на захоплення місцевого ринку
3. Галузева ознака: внутрішньогалузева	Економічна боротьба з конкурентами відбувається в одній галузі економіки, але є відмінності у функціонуванні	Пропозиція суттєвих переваг у порівнянні з продуктами конкурентів у визначеній галузі економіки

Продовження таблиці 3.8		
4. Конкуренція за видами товарів: товарно-видова	Конкуренція відбувається між послугами одного виду. За такої конкуренції значення набуває марка товару	Постійна робота над забезпеченням високого рівня іміджу компанії
5. За характером конкурентних переваг: нецінова	Передбачається ведення конкурентної боротьби не за рахунок зниження ціни на аналогічні послуги, а за рахунок новизни та унікальних характеристик технології, на якій базується функціонування системи	Акцент на унікальних характеристиках пропонованого товару
6.За інтенсивністю: марочна	Виведення товару на ринок передбачається під власною маркою, а також створення асоціації між назвою фірми і технологія MapReduce	Просування продукту компанії під визначеним брендом

Таблиця 3.9

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Hadoop, Google Cloud Platform	Гнучкі ціни, Патент на продукт	Змінні витрати постачальників	Рівень чутливості до зміни цін	Ціна, лояльність споживачів
Висновки	Конкуренція не інтенсивна, кожен працює в окремому регіоні	Можливість входу в ринок висока. Потенційні конкуренти присутні	Постачальник може диктувати умови: ціни на послуги	Кожен з клієнтів потребує індивідуального підходу для вирішення його задач	Обмежень для роботи на ринку з боку товарів замінників на даний момент не існує

В результаті проведення аналізу таблиці 3.9, можна зробити висновок, що можливість виходу на ринок з огляду на конкурентну ситуацію є високою. Для виходу на ринок товар в першу чергу повинен пропонувати унікальні характеристики, які відсутні у продуктах конкурентів.

На основі аналізу конкуренції, проведеного в таблиці 3.9, а також із урахуванням характеристик ідеї проекту (таблиця 3.2), вимог споживачів до товару (таблиця 3.5) та факторів маркетингового середовища (таблиці 3.6 та 3.7), визначається та обґрунтовується перелік факторів конкурентоспроможності, що надається у таблиці 3.10.

Таблиця 3.10

Обґрунтування факторів конкурентоспроможності

№ п/п	Фактори конкурентоспроможності	Обґрунтування
1	Динаміка галузі	Технологія MapReduce наразі є дуже популярною, тому підприємства зацікавлені у ній
2	Концепція товару і послуги	Система розподілених обчислень з використанням MapReduce дозволяє звільнитися від посередника
3	Після продажне обслуговування	Підтримка щодо використання системи після її продажу

Таблиця 3.11

Порівняльний аналіз сильних та слабких сторін системи розподілених обчислень з використанням технології MapReduce

№ п/п	Фактори конкуренто спроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з власною системою						
			-3	-2	-1	0	1	2	3
1	Динаміка галузі	5		✓					
2	Концепція товару і послуги	12			✓				
3	Після продажне обслуговування	4						✓	

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (таблиця 3.12).

Таблиця 3.12

SWOT-аналіз стартап проекту

Сильні сторони:	Слабкі сторони:
- Інноваційні технології - Висока якість	- Слабкий імідж компанії - Мало оборотних коштів
Можливості:	Загрози:
- Нові технології - Нові потреби клієнтів - Тенденції попиту	- Продукти-замінники - Законодавче регулювання - Зміна тенденцій

Таблиця 3.13

Альтернативи ринкового впровадження стартап-проекту

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії	Переважно готові	Дуже високий	Низька	Легко
2	Держустанови	Готові	Дуже високий	Висока	Важко

Базові стратегії в обраних сегментах ринку представлені у таблиці 3.14.

Таблиця 3.14

Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції	Базова стратегія розвитку
1	Динамічний розвиток з використанням маркетингу та встановлення бізнес-контактів	Підняття рейтингу компанії шляхом маркетингу, встановлення конкурентоспроможних цін	Незалежність від посередника, який утримує кошти за свої послуги	Стратегія лідерства по витратах
2	Динамічний розвиток завдяки висвітленню унікальних характеристик наданих послуг	Унікальність послуг, що надає розподілені обчислення	Використання технології MapReduce, що дозволяє зробити процес прозорим та анонімним	Стратегія диференціації

Залежно від міри сформованості галузевого ринку, характеру конкурентної боротьби, необхідно обрати одну з трьох стратегій конкурентної поведінки: розширення первинного попиту, оборонну або наступальну стратегію або ж застосувати демаркетинг або диверсифікацію (таблиця 3.15).

Таблиця 3.15

Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект першопрохідцем на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів	Чи буде компанія копіювати основні характеристики	Стратегія конкурентної поведінки
1	Проект не є першопрохідцем	Компанія буде шукати нових користувачів	Компанія буде копіювати найкращі з характеристик конкурентів	Стратегія наслідування лідеру задля економії фінансових ресурсів

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (таблиця 3.5), а також в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки була розроблена стратегія позиціонування (таблиця 3.16).

Таблиця 3.16

Визначення стратегії позиціонування

№ п/п	Вимоги до товару	Базова стратегія розвитку	Ключові позиції	Вибір асоціацій
1	Доступність, захищеність	Стратегія диференціації	Використання технології MapReduce	Доступність, захищеність

3.4 Розроблення маркетингової програми стартап-проекту

Маркетингова програма - це намічений для планомірного здійснення, об'єднаний єдиною метою та залежний від певних строків комплекс взаємопов'язаних завдань і адресних заходів соціального, економічного, науково-технічного, виробничого, організаційного характеру з визначенням ресурсів, що використовуються, а також джерел одержання цих ресурсів. Основну увагу слід приділяти вибору, значенню та формі інструментів маркетингу, їх об'єднанню в найбільш оптимальний з погляду визначеної мети комплекс, а також розподілу фінансових ресурсів у межах бюджетування маркетингу.

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару (таблиця 3.17).

Таблиця 3.17

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентом
1	Можливість виконання розподілених обчислень	Доступність, що забезпечується технологією MapReduce	Абстрагування від фізичних властивостей вузлів
2	Виконання різнотипових завдань	Уніфікація алгоритмові ділення і склеювання, абстрагування від метода балансування від типу завдання	MapReduce дозволяє кожному автору задачі на свій лад модифікувати алгоритм якщо потрібно

Надалі розробляється трирівнева маркетингова модель товару: уточняється ідея продукту, складові, процесу надання (таблиця 3.18).

Таблиця 3.18

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
1. Товар за задумом	Товар забезпечує проведення розподілених обчислень на мобільних пристроях
2. Товар у реальному виконанні	Властивості: доступність, цілісність, конфіденційність, гнучкість, зручність
	Товар представляє собою програмний комплекс з двох модулів: сервер управління з MapReduce та мобільний додаток для виконання обчислень.
	Поставляється у вигляді застосунку в форматі apk
	Назва: Distributed calculation system Reckon
3. Товар із підкріпленням	До продажу: відбувається інсталяція та конфігурування системи, проводяться тренінги для клієнта
	Після продажу: відбувається підтримка програмного забезпечення та його доопрацювання під потреби клієнта

Аналіз системи збуту передбачає визначення ефективності кожного елемента цієї системи, оцінювання діяльності апарату працівників збуту. Аналіз витрат обігу передбачає зіставлення фактичних збутових витрат за кожним каналом збуту і видом витрат із запланованими показниками для того,

щоб виявити необґрунтовані витрати, ліквідувати затрати, що виникають у процесі руху товарів і підвищити рентабельність наявної системи збуту.

Дані щодо визначення системи збуту надаються в таблиці 3.19.

Таблиця 3.19

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Мобільні магазини Google Play та App Store	Реєстрація користувача в системі та надання йому відповідних прав на створення завдань	Канал нульового рівня, продаж товару відбувається безпосередньо споживачам через онлайн маркети	Оптимальною системою збуту є прямий збут з каналом нульового рівня за відсутності посередників

У якості концепції маркетингових комунікацій були обрані інтегровані маркетингові комунікації, де компанія ретельно обмірковує і координує роботу своїх численних каналів комунікації, рекламу в засобах масової інформації, особистий продаж, стимулювання збуту, пропаганду, прямий маркетинг, упаковку товару.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО МЕТОДУ

Розділ розглядає опис реалізації методу балансування та результат застосування методу в розподіленій системі на мобільній платформі. Завданням розділу є отримання результатів обчислення завдань введених в систему з використанням покращеного методу балансування та співставлення із результатами обчислень без використання цього методу.

4.1 Реалізація архітектури мережі

На першому етапі рішення задачі був спроектований план відповідної архітектури для глобальної мережі розподіленої системи на рис. 4.1.

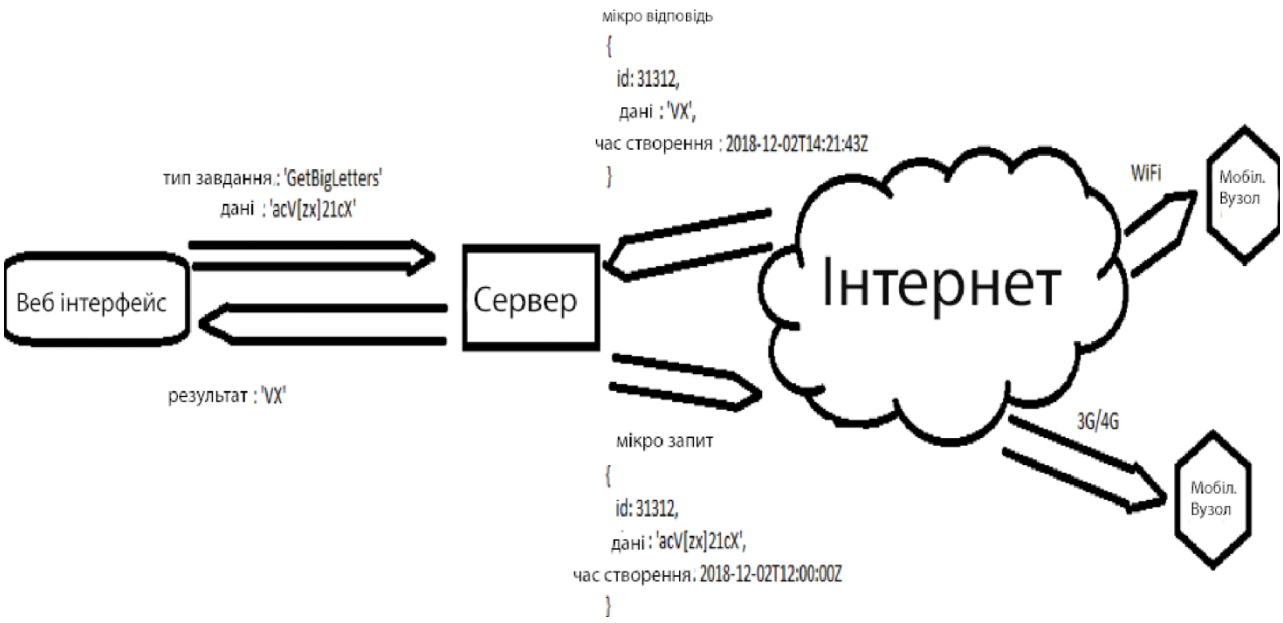

Рис. 4.1 Архітектура мережі для розподіленої системи

Данна схема показує спосіб з’єднання та систему обміну даними по маршруту «сервер-вузол» та «вузол-сервер». База даних також включена в сервер. Сервер має стабільне з’єднання з мережею Інтернет, через яку він спілкується (віддає мікро завдання та отримує мікро відповіді) з мобільними вузлами. Самі мобільні вузли можуть бути рознесені територіально, адже вони географічно незалежні та не мають конкретної вимоги до типу з’єднання з мережею Інтернет (WiFi, 3G,4G).

Така архітектура мережі дає змогу абстрагуватися від таких залежностей:

- Географічна прив’язаність
- Визначений тип з’єднання
- Балансування трафіку
- Використання спеціальних проколів передачі даних
- Різнотиповість формату даних
- Налаштування спеціального тунелю для доступу до серверу.

З даної архітектури виділимо два типи прогам які потрібні для реалізації успішної роботи розподіленої мережі :

- Серверний код - програма яка встановлюється на сервер та виконує основні цикли роботи розподіленої системи
- Клієнтський код – програма (додаток) для мобільного пристрою, що реєструє пристрій в мережі, отримує частину завдання та обробивши, відправляє результат на сервер.

Таким чином для кожної із сторін розроблялося своє ПЗ націлене на ефективне використання ресурсів та найшвидше отримання відповіді на поставлене завдання.

4.2 Реалізація алгоритму роботи метода

Для роботи розподіленої системи було вибрано парадигму MapReduce і за нею побудований алгоритм для ділення вхідних завдань, обробки частин завдань на мобільних вузлах та збір відповідей в єдиний результат для видачі через веб-інтерфейс.

Даний алгоритм реалізований на серверній частині, що відносить його до “backend” частини всієї системи.

На рис. 4.2 приведено приклад виконання розробленого алгоритму для пошуку великих літер в словах (текстах).

Рис. 4.2 Алгоритму роботи метода

Для прикладу було вибрано речення «Приклад роботи MapReduce». Ці вхідні данні були отримані на сервері. Потім речення було поділене на слова та збережено в БД. Далі на підключені пристрої були надіслані слова, де за спеціальним правилом виділяються та відправляються назад на сервер великі букви. Де вже сам сервер збирає результати воєдино, видаляє проміжні значення та зберігає кінцевий результат.

4.3 Обробка завдань на сервері

На сервер завдання потрапляють через веб-інтерфейс, який дозволяє додавати, переглядати завдання та отримувати інформацію про пристроях підключених до системи та завдання які вони виконують.

При створенні завдання можна вибрати тільки зараніше визначені типи завдань, які можна додавати динамічно до системи через збереження нового алгоритму для обробки завдання в БД розподіленої системи.

На рис. 4.3 показано сторінку створення завдання з додаванням «заголовку» , «автора», «опису» завдання, вибрати із випадаючого списку тип завдання та ввести вхідну інформацію у відповідне поле.

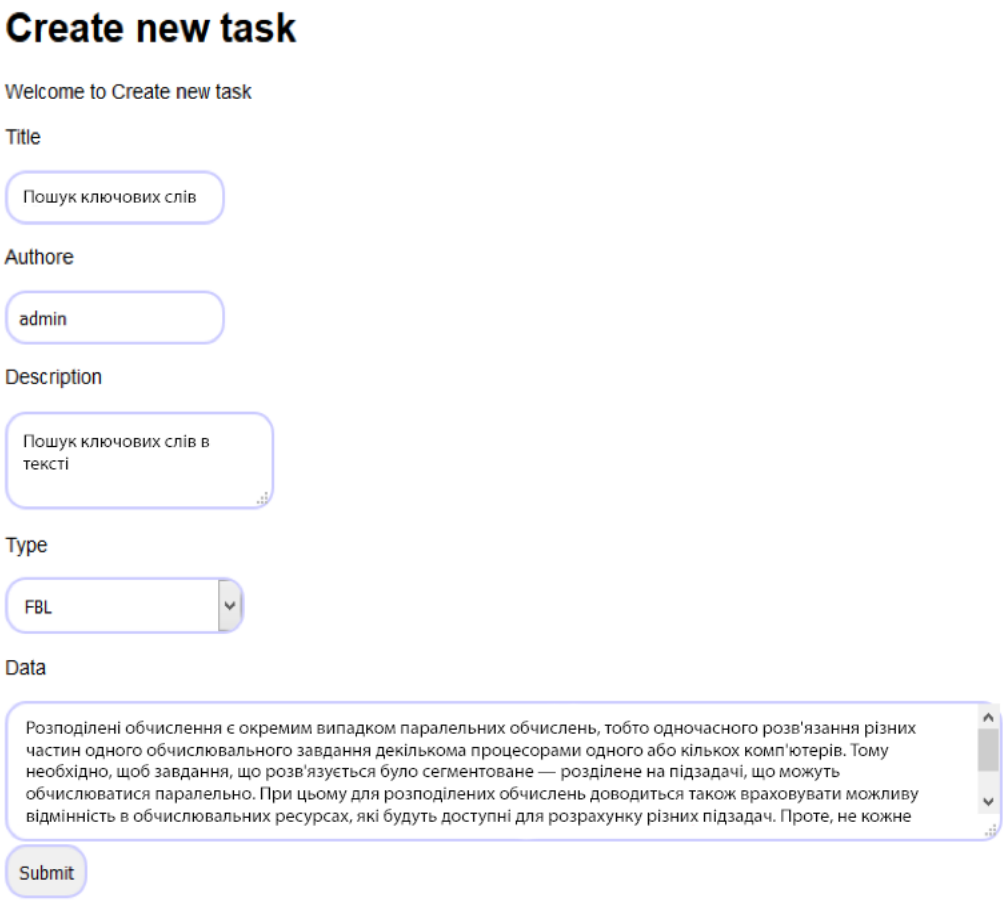

Рис. 4.3 Сторінка створення завдання

Сторінка перегляду завдання зберігає ту саму інформацію, що вводилась при створенні завдання, але з додаванням інформації про статус завдання, процентне значення виконання та результат обчислень.

На сторінці також додані посилання на сторінки із списком всіх завдань, на сторінку додавання нового завдання та перегляду всіх пристроїв що є зареєстрованими у системі.

На рис. 4.4 показано приклад створеного завдання із типом завдання «FBL», що означає «Find Big Letters». В нового завдання статус «New» як завдання яке ще не оброблює жоден пристрій. Відповідно і процентне відношення виконання завдання рівне нулю.

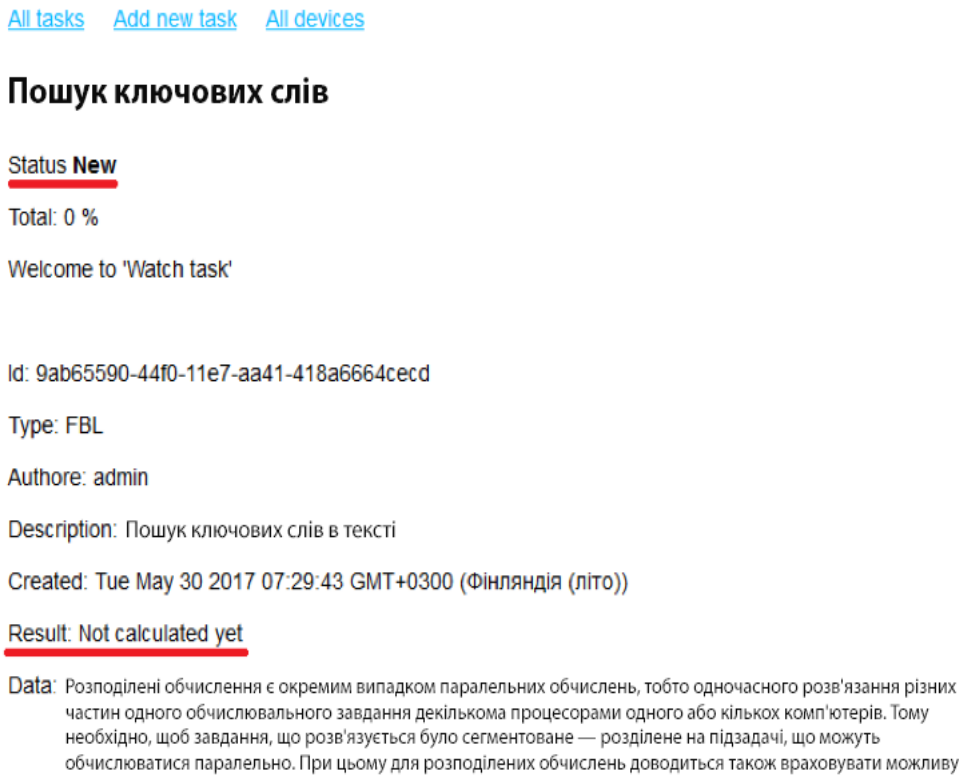

Рис. 4.4 Сторінка завдання після її створення

Веб-інтерфейс дає змогу відійти від особливостей роботи на різних комп’ютерах та операційних системах, і бути доступним будь де, навіть на мобільному пристрої. Завдяки цьому користувач може в будь-який момент переглянути стан виконання поставленого ним завдання та дізнатися кількість оброблених завдань в відсотковому відношенні до загальної суми завдань та окремо (за бажанням) отримати відсоткове відношення до кількості ще не виконаних завдань.

4.4 Обробка завдань на мобільному пристрої

Впроваджуючи розподілену систему на мобільний пристрій, постає необхідність створення окремого додатку, що буде нативно виконувати необхідні алгоритми обробки отриманих даних, та зберігати цей проміжний результат для відправлення на сервер.

Мобільний додаток повинен виконуватися у фоновому режимі, щоб не заважати користувачу виконувати інші повсякденні завдання та мати змогу, при закінченні обробки завдання повідомляти користувача про це та надавати можливість вибрати інше завдання із списку яке ще не перейшло у статус завершено та має вільні «частинки» які потребуються обробки.

На рис. 4.5 приведено приклади такого додатку на мобільний пристрій де є список завдань та вкладка із переглядом статусу виконання теперішнього завдання.

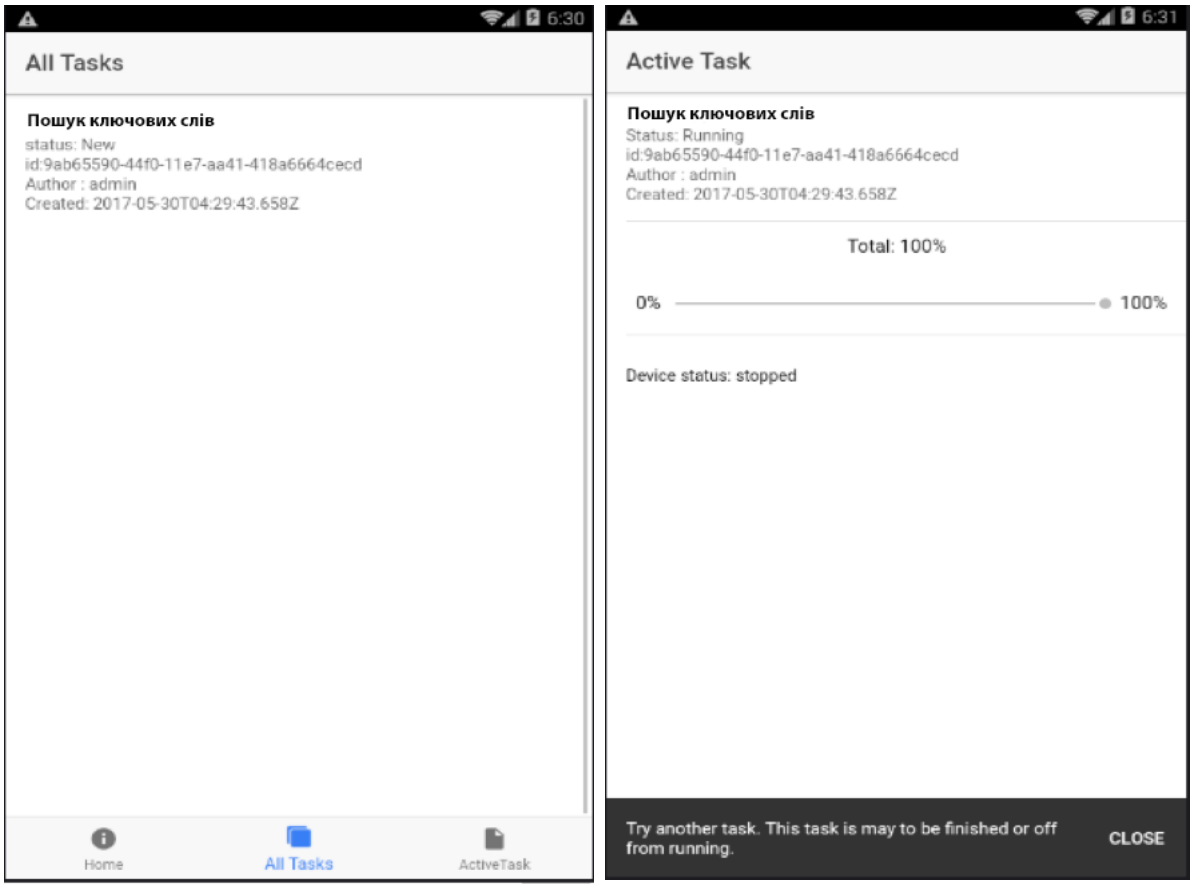

Рис. 4.5 Приклад додатку на мобільному пристрої

4.5 Порівняння роботи системи з покращеним методом

Після вдосконалення методу балансування завдань між вузлами розподіленої системи, було проведено повторні заміри часу обробки загального завдання, результати якого показані на рис. 4.6.

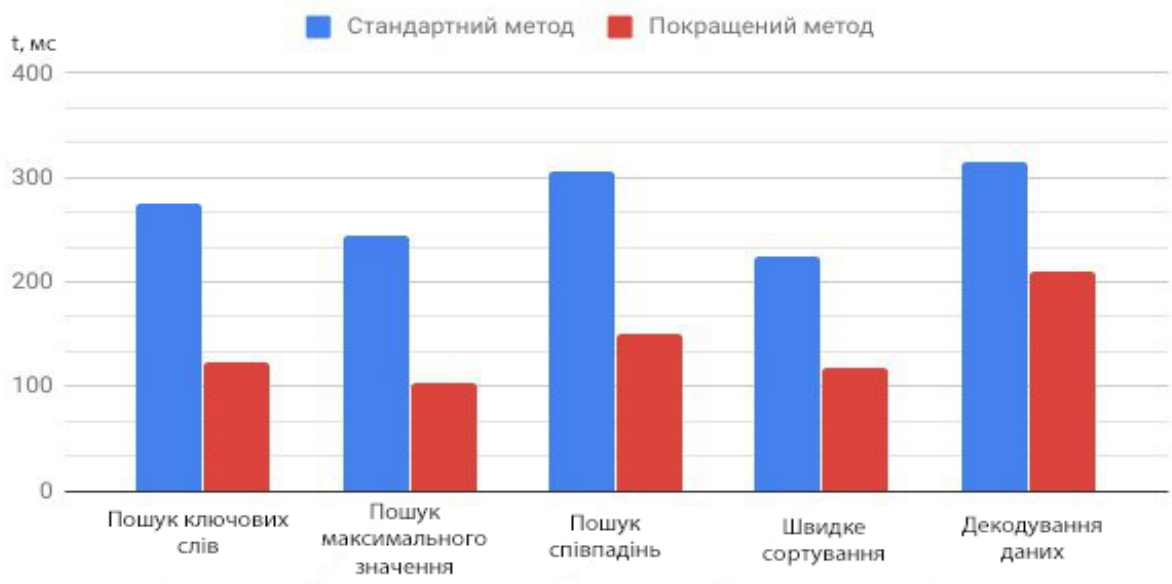

Рис. 4.6 Приклад роботи системи за різними методами

На діаграмі зображені середні значення знаходження підзавдань в системі. Різний час оброки різних типів завдань викликаний складністю самого алгоритму обчислення підзавдань на мобільному вузлі.

Висновки

1. Розроблено та адаптовано метод для організації балансування завдань в розподіленій системі, прикладну програму для обробки вхідних завдань та «склеювання» результату воєдино на сервері.

2. Проведено порівняння між швидкістю оброблення завдань в системі, які показали що з покращеним методом балансування, завдання обробляються майже в 2 рази швидше (в залежності від типу завдання та складності алгоритму обробки даних на робочому вузлі мережі).

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1. Проведено детальний аналіз методів розподілення задач в гетерогенних системах розподілених обчислень для ефективного використання мобільних вузлів.

2. Було досліджено різни алгоритми роботи розподілених систем. Серед цих систем була виділена Hadoop та її метод балансування, адже він задовольняє ряду вимог, які були поставлені перед системою, як гетерогенною мережею із мобільними пристроями в якості робочих вузлів.

3. Обраний метод добре підходить для реалізації розподіленої системи адже він вирішує питання паралелізації завдання між робочими вузлами; розмежування сфер роботи пристроїв, тобто встановлення незалежних зв’язків між пристроями та зменшення ризику втрати великої частини обчислень через раптове відключення вузла від мережі Інтернет.

4. Було розроблено та адаптовано метод балансування для його використання та встановлення на сервер. Серверна програма представляє собою веб-інтерфейс який дозволяє користувачу створювати та переглядати стан завдань поставлених на обчислення. Також на в цій програмі відображаються всі пристрої підключені до розподіленої системи та інформація по них (стан, номер задачі яку обробляє, технічні характеристики). Для мобільний пристроїв було розроблено свій додаток, який відображає поточний стан пристрою як вузла в системі, надає повний список всіх завдань з їх статусами, там відображає інформацію по завданню що виконується на пристрої (статус, процентне відношення, кількість виконаних операцій).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zhukov I. Integral telecommunication environment implementation concept for harmonized air traffic control with scalable display systems // Aviation – 2010. – vol.16

2. Іванкевич О. В. Засоби керування потоками даних у розподілених обчислювальних системах // Проблеми інформатизації та управління: Зб. наук. пр.–К.: Вид-во нац. авіац. «НАУ-друк», 2010.–Вип. 3(31).–С. 65-69.

3. Мадяр А. Й. Інноваційні методи дослідження в розподілених : ИМЕДИС, 2015. – 120 с.

4. Guth L. S.: The effect of wavelength of visual perception latency. Vis. Res. 4, 567 , 1964.

5. Mary Jo Foley. Hortonworks delivers beta of Hadoop big-data platform for Windows. (англ.). ZDNet (17 February 2013). — «In 2011, Microsoft announced it was partnering with Hortonworks to create both a Windows Azure and Windows Server implementations of the Hadoop big data framework».

6. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика / Ф. Уоссермен — Мир, 1992. — 240 с. — ISBN 5-03-002115-9.

7. Джарратано Дж. Экспертные системы: принципы разработки и программированив: Издательский дом «Вильямс», 2006. — 1152 с.

8. Джексон П. Введение в экспертные системы / П. Джексон — М. : Издательский дом «Вильямс, 2001. — С. 624. — ISBN 0-201-87686-8.

9. Таунсенд К. Проектирование и программная реализация экспертных систем на персональных ЭВМ / К. Таунсенд, В. А. Кондратенко, С. В. Трубицына. — М.: Финансы и статистика, 1990. — 320 с.

10. Гаврилова Т. А. Базы знаний интеллектуальных систем. Учебник. / Т. А. Гаврилова, В. Ф. Хорошевский — СПб.: Питер, 2000.

11. Клименко І . А . Класифікація та архітектурні особливості програмованих мультипро - цесорних систем - на - кристалі // Вісник НТУУ «

КПІ ». Інформатика, управління та обчислюваль – на техніка : Зб. наук. пр. – К.: Видавництво «ВЄК+», 2011. – No 55.

12. Воеводин В. В. Параллельные вычисления. — СПб: БХВ-Петербург, 2002. — 608 с

13. Hachul S. An Experimental Comparison of Fast Algorithms for Drawing General Large Graphs / S. Hachul , M. Jünger // 13th International Symposium, GD 2005: Conf., September 12-14 2005, Limerick, Ireland: proc. of conf. / LNCS, Springer. – 2006. – Vol. 3843. – P. 235-250.

14. Кормен Т. Алгоритмы. Построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. –2-е изд. – М.:Вильямс, 2005. – 1328 с.

15. Barnes J. A hierarchical O(N log N) force-calculation algorithm / J. Barnes, P. Hut // Nature. – 1986. – Vol. 324. – No 4.

Додаток А

Код класу MapReduce мови JavaScript

```
var PartsModel = require('../libs/mongoose').PartsModel;
var TaskModel    = require('../libs/mongoose').TaskModel;
var log         = require('../libs/log')(module);

module.exports.FBL={
  map :  function(task){
    var data = task.data;
    var step = 40;
    for(var i=0; i < data.length; i= i + step){
      var part_data = data.substring(i,i+step);
      var part = new PartsModel({task_id:task.id,data:part_data});
      part.save(function(err){
        if(err)errorHandler(err,'FBL map error',task);
        else console.log('saved: ',part.id)
      })
    }
  },
  reduce : function(task,incomdata){
    var data = incomdata || [];

    PartsModel.findOne({task_id:task.id},function(err,part){
      if(err || !part){
        err && errorHandler(err,'FBL reduce error',task);
        if(!err && incomdata){

TaskModel.findOneAndUpdate({id:task.id},{result:data,status:'Done'},function(err){
              if(err)errorHandler(err,'FBL reduce error result save task',task);
            })
          }
      }
      else if(!part.result)errorHandler({},'FBL there is part without result',part);
      else {
```

```
          data.push(part.result);
          PartsModel.remove({id:part.id},function(err){
            console.log('removed');
            if(err)errorHandler(err,'FBL error removing part',task);
            else module.exports.FBL.reduce(task,data);
          })
        }
      })
    }
}
```

Код функції балансування

```
module.exports.ballance =
function(task,incom,res,sendPartError,sendPart){//sendPartError(err,res,incom) ||
sendPart(part,incom,res)
   return
PartsModel.findOne({task_id:task.id}).where('result').equals(null).exec(function(err,part){
      if(err)return sendPartError(err,res,incom);
      else if(!part){
         sendPartError({error:'There is no parts'},res,incom);

TaskModel.findOneAndUpdate({id:task.id},{status:'Reducing'},function(err,data){if(err)err
orHandler(err,'Ballance error updating status',task)})
         module.exports[task.type].reduce(task);
      } else {
         sendPart(task,part,incom,res);
         return
PartsModel.findOneAndUpdate({id:part.id},{device:incom.device_id},function(err){if(err)e
rrorHandler(err,'updating device id in part error',part)});
      } })}
```