**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

Інститут телекомунікаційних систем

(повна назва інституту/факультету)

Кафедра «Інформаційно-телекомунікаційних мереж»

(повна назва кафедри)

|  |  |
| --- | --- |
| «На правах рукопису»  УДК \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | До захисту допущено»  Завідувач кафедри  \_\_\_\_\_\_\_\_\_\_ Л.С.Глоба  (підпис) (ініціали, прізвище)  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_2018 р. |

**Магістерська дисертація**

з спеціальності 6.050903 «Телекомунікації та радіотехніка»

(код і назва)

на тему: Метод балансування задач між мобільними вузлами розподілені системи обчислень

Виконав (-ла): студент (-ка) \_6\_\_\_ курсу, групи \_\_\_ТІ-71мп\_\_\_\_

(шифр групи)

Павленко Владислав Миколайович

(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Доцент , к.т.н., старший науковий співробітник,

Алєксєєв М. О.

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

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

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2018 року

**Національний технічний університет України**

**«Київський політехнічний інститут ім. Ігоря Сікорського»**

Інститут телекомунікаційних систем

(повна назва)

Кафедра «Інформаційно-телекомунікаційних мереж»

(повна назва)

Рівень вищої освіти – другий (магістр) за освітньо-професійною

Напрям підготовки 6.050903 «Телекомунікації та радіотехніка»

(код і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_ Л.С.Глоба

(підпис) (ініціали, прізвище)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2018 р.

**ЗАВДАННЯ**

**на магістерську дисертацію студенту**

Павленку Владиславу Миколайовичу

(прізвище, ім’я, по батькові)

1. Тема дисертації: Метод балансування задач між мобільними вузлами розподілені системи обчислень ,

науковий керівник дисертації Алєксєєв Микола Олександрович, доцент , к.т.н., старший науковий співробітник ,

затверджені наказом по університету від «11» квітня 2018 р. №1321-с

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

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

4. Вихідні дані:

1. Інформація про системи розподілених обрахунків.
2. Принципи розподіленої роботи гетерогенних систем
3. Порівняльні статистики алгоритмів розподіленої роботи

5. Перелік завдань, які потрібно розробити:

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

2. Запропонувати метод управління завданнями для розподіленого обрахування.

3. Розробити та розгорнути макет робочого серверу по розподіленню завдань та клієнтський додаток для виконання обчислень.

6. 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 «12» вересня 2018 року.

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання  дипломної роботи | Термін виконання  етапів роботи | Примітка |
| 1 | Дослідження та вивчення отриманого завдання | 12.09.17 – 01.10.17 | виконано |
| 2 | Аналіз існуючих рішень розподілених обрахунків | 01.10.16 – 12.01.17 | виконано |
| 3 | Аналіз підходів до розробки систем розподілених обрахунків | 12.01.17 – 16.02.17 | виконано |
| 4 | Підготовка тезисів (ПТ-18) | 01.02.17 – 01.03.17 | виконано |
| 5 | Створення прототипу розподіленої системи, опис компонентів | 01.03.17 – 01.04.17 | виконано |
| 6 | Розробка алгоритму балансування задач між мобільними пристроями | 01.11.16 – 01.03.17 | виконано |
| 7 | Написання алгоритму закінчення роботи над завданням | 01.03.17 – 26.03.17 | виконано |
| 8 | Розробка розподіленої системи | 01.11.16 – 01.05.17 | виконано |
| 9 | Підготовка тексту диплому | 15.04.17 – 12.06.17 | виконано |

Студент \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали, прізвище)

**РЕФЕРАТ**

Робота містить 58 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 work contains 58 pages, 14 figures, 15 sources have been used.

**Objective**: Suggest a method for distributing tasks of a heterogeneous distributed computing system for the efficient use of mobile nodes.

Analyzed the principles of construction of distributed computing and complete review of existing systems.

Considered features of the distributed systems for mobile platforms.

Developed software distributed system implementation including features on mobile platforms. Formed comparative characteristics of different calculation algorithms.

Created model for the demonstration of the developed system and implemented algorithms.

**Keywords**: distributed calculations, heterogeneous system, balancing tasks

ЗМІСТ

[ВСТУП 9](#_Toc530692280)

[РОЗДІЛ 1 11](#_Toc530692281)

[Аналіз принципів побудови систем розподілених обчислень та огляд існуючих зразків їх реалізації. 11](#_Toc530692282)

[1.1 Розподілені обчислення. Мета та цілі їх застосування 11](#_Toc530692283)

[1.2 Види розподілених архітектур на портативних пристроях 15](#_Toc530692284)

[1.2.1 Архітектура репозиторія 15](#_Toc530692285)

[1.2.2 Архітектура клієнт/сервер 17](#_Toc530692286)

[1.2.3 Архітектура абстрактної машини 22](#_Toc530692287)

[1.3 Приклади реалізацій розподілених систем 26](#_Toc530692288)

[1.3.1 Система UNICORE 26](#_Toc530692289)

[1.3.2 Система Hadoop 28](#_Toc530692290)

[1.3.3 Система BOINC 31](#_Toc530692291)

[Висновки 36](#_Toc530692292)

[РОЗДІЛ 2 37](#_Toc530692293)

[Реалізації системи розподілених обчислень із застосуванням технології MapReduce 37](#_Toc530692294)

[2.1 Парадигма MapReduce 37](#_Toc530692295)

[2.2 Паралелізм в MapReduce 38](#_Toc530692296)

[2.3 Рівні паралелізму в MapReduce 39](#_Toc530692297)

[2.3.1 Паралелізм бітового рівня 39](#_Toc530692298)

[2.3.2 Паралелізм рівня інструкцій 40](#_Toc530692299)

[2.3.3 Паралелізм даних 40](#_Toc530692300)

[2.3.4 Паралелізм задач 41](#_Toc530692301)

[2.4 Реалізація алгоритмів в MapReduce 41](#_Toc530692302)

[2.4.1 Продуктивність розподілених алгоритмів в MapReduce 44](#_Toc530692303)

[2.4.2 Надійність розподілених алгоритмів в MapReduce 45](#_Toc530692304)

[2.5 Функціональне програмування в MapReduce 46](#_Toc530692305)

[Висновки 47](#_Toc530692306)

[РОЗДІЛ 3 48](#_Toc530692307)

[РЕАЛІЗАЦІЇ МЕТОДІВ БАЛАНСУВАННЯ 48](#_Toc530692308)

[РОЗДІЛ 3 50](#_Toc530692309)

[РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 50](#_Toc530692310)

[3.1 Опис ідеї проекту 50](#_Toc530692311)

[3.2 Технологічний аудит ідеї проекту 52](#_Toc530692312)

[3.3 Аналіз ринкових можливостей запуску стартап-проекту 53](#_Toc530692313)

[3.4 Розроблення маркетингової програми стартап-проекту 64](#_Toc530692314)

[РОЗДІЛ 4 67](#_Toc530692315)

[ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО МЕТОДУ 67](#_Toc530692316)

[4.1 Реалізація архітектури мережі 67](#_Toc530692317)

[4.2 Реалізація алгоритму MapReduce 68](#_Toc530692318)

[4.3 Обробка завдань на сервері 69](#_Toc530692319)

[4.4 Обробка завдань на мобільному пристрої 71](#_Toc530692320)

[Висновки 72](#_Toc530692321)

[ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 73](#_Toc530692322)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 74](#_Toc530692323)

[Додаток А 76](#_Toc530692324)

**перелік скорочень**

|  |  |
| --- | --- |
| JSON | JavaScript Object Notation |
| MVC | Model-View-Controller |
| SQL  Wi-Fi | Structured Query Language  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 обрахунки 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 обрахунків 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 обрахунків. 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 обрахунків та досліджено можливість їх використання для обчислення інформації.
2. Запропоновано метод реалізації балансування задач в системі розподілених обрахунків із застосуванням технології MapReduce.
3. Розглянуто особливості побудови запропонованої системи для мобільних платформ.

**Практичний результат дослідження:**

1. Розроблено метод балансування задач для системи розподілених обрахунків, 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.

## 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 спі­ль­не використання апаратних та програмних ресурсів, наприклад жорстких ди­сків, принтерів, файлів, компіляторів та інше, об’єднаних засобами мережі. Оче­видно, що розподіл ресурсів можливий і в багатокористувацьких системах, але в цьому випадку за надання ресурсів і їх керування повинен керувати цент­ральний процесор.
2. **Відкритість**. Це можливість розширювати систему шляхом добавля­ння нових ресурсів. Розподілені системи – це відкриті системи, до яких приєд­нуються 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 масштабова­ними: щоб система відповідала новим вимогам, її можна нарощувати за допо­мо­гою добавляння нових обчислювальних ресурсів. Але на практиці нарощу­вання може обмежуватися мережею, яка об’єднує окремі комп’ютери системи. Якщо приєднати багато нових машин, то пропускна здатність мережі може ви­явитися 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 повніс­тю 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, реак­ція 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 забезпеченнявідноситься власне до такого класу ПЗ. Воно знаходиться якби посередині між різними частинами розподілених компонент системи.

## 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.

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.

## Архітектура репозиторія

Для того щоб підсистеми, які входять в систему, працювали ефективно, між ними повинен проходити обмін інформацією. Обмін можемо організувати двома способами [5] :

* Всі дані які використовуються спільно зберігаються в центральній ба­зі даних, доступній всім підсистемам. Модель системи, основана на спільно­му використання бази даних, часто називають моделлю репозиторія.
* Кожна підсистема має свою власну базу даних. Взаємо обмін даними між підсистемами відбувається за допомогою передачі повідомлень.

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].

На рис. 1.2 подано приклад архітектури інтегрованого набору CASE-інструментів, яка базується на сумісному використанні репозиторію.

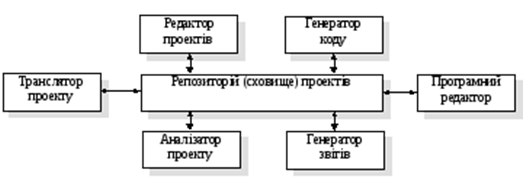


Рис. 1.2 Архітектура інтегрованого набору CASE-засоби

Широке використання дана модель отримала після того, як була застосована для підт­ри­мки розробки систем, написаних на мові Ада.

Спільне використання репозиторію має як переваги, так і недоліки [7]:

* Очевидно, спільне використання великих об’ємів даних ефективне, оскільки непотрібно передавати дані із однієї підсистему в іншу.
* З іншої сторони, підсистеми повинні бути узгоджені з моделлю репозиторію даних. Це завжди потребує компромісу між вимогами, поставленими до кожної підсистеми. Компромісне рішення може знизити їх продуктивність. Якщо форми даних нових підсистем не підходять під узгоджену модель поданих даних, інтегрувати такі підсистеми складно або неможливо.
* Підсистемі в якій створюються дані, не потрібно знати, як ці дані використовуються в інших підсистемах.
* Оскільки у відповідності з узгодженою моделлю даних генеруються великі об’єми інформації, модернізація таких систем ускладнена. Переведення системи на нову модель даних буде дорогим і складним, а іноді неможливим.
* В системах з репозиторієм такі засоби, як резервне копіювання, забезпечення безпеки, керування доступом та відновлення даних, централізовані, оскільки входять в систему управління репозиторієм. Ці засоби виконують тільки свої основні операції і не займаються іншими операціями.
* З іншої сторони, до різних підсистем ставляться різні вимоги, відносно безпеки, відновлення та резервування даних. В моделі репозиторію до всіх підсистем застосовується однакова політика.
* Модель спільного використання репозиторію прозора: якщо нові підсистеми сумісні із узгодженою моделлю даних, їх можемо безпосередньо інтегрувати в систему.
* Однак складно розмістити репозиторій на декількох машинах, оскільки можуть виникнути проблеми, пов’язані з надлишковістю та порушенням цілісності даних.

В даній моделі репозиторій є пасивним елементом, а управління покладене на підсистеми, які використовують дані з репозиторію. Для систем штучного інтелекту розроблено альтернативний підхід. Він базується на моделі «робочої області» [8], яка ініціює підсистеми тоді, коли конкретні дані стають доступними. Такий підхід можливо використовувати до систем, в яких форма даних добре структурована.

## Архітектура клієнт/сервер

Модель архітектури клієнт/сервер – це модель розподіленої системи, в якій показано розподіл даних і процесів між декількома процесорами.

Модель містить три основних компоненти [9]:

1. Набір автономних серверів, які надають сервіси іншим підсистемам. Наприклад, сервер друку, який надає послуги друку, файлові сервери, які надають сервіси управління файлами, і сервер-компілятор, який надає сервіси компіляції вихідних кодів програми.
2. Набір клієнтів, які викликають сервіси, що надаються серверами. В контексті системи клієнти є звичайними підсистемами. Дозволяється паралельне виконання декількох екземплярів клієнтської програми.
3. Мережа, за допомогою якої клієнти отримують доступ до сервісів. В принципі немає ніякої заборони на те, щоб клієнти і сервери запускались на одній машині. На практиці модель клієнт/сервер в такій ситуації не використовується.

Клієнти повинні знати імена доступних серверів та сервісів, які вони надають. В той же час серверам непотрібно знати ні імена клієнтів, ні їх кількість. Клієнти отримують доступ до сервісів, які надає сервер, за допомогою віддале­ного виклику процедур [10].

Під час роботи сервери і клієнти обмінюються даними, але при обміні великими об’ємами даних можуть виникнути проблеми, пов’язані з пропускною здатністю мережі. Правда, з розвитком все більш швидких мереж ця проблема втрачає своє значення.

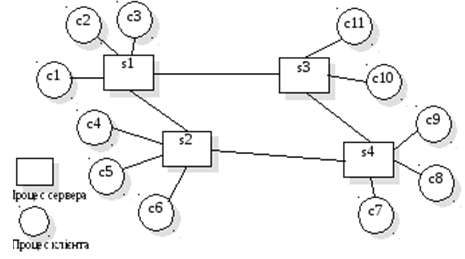
Приклад системи, організованої за типом моделі клієнт/сервер, зображено на рис. 1.3.

Це багатокористувацька гіпертекстова система, призначена для підтримки бібліотеки відеофільмів і фотографій. В ній міститься декілька серверів, які розміщують різні типи медіа файлів і керують ними. Відео файли необхідно передавати швидко і синхронно, але з відносно малою роздільною здатністю. Вони можуть зберігатись у стисненому стані. Фотографії повинні передаватись з високою роздільною здатністю. Каталоги повинні забезпечувати роботу з множиною запитів і підтримувати зв'язок із використанням гіпертекстової системи [11]. Тут клієнтська програма є просто інтегрованим інтерфейсом користувача.

Рис. 1.3 Архітектура бібліотечної системи файлів

Підхід клієнт/сервер можна використовувати при реалізації систем, що базуються на репозиторії, який підтримується як сервер системи. Підсистеми, які мають доступ до репозиторію, є клієнтами. Але звичайно кожна підсистема керує власними даними.

Найбільш важлива перевага моделі клієнт/сервер полягає в тому, що вона є розподіленою архітектурою. ЇЇ ефективно використовувати в мережевих системах з множиною розподілених процесорів.

В дану систему легко добавити новий сервер і інтегрувати її з останньою частиною системи або ж обновити сервери, не діючі на інші частини системи.

В архітектурі клієнт/сервер програмні додатки моделюються як набір се­рвісів, що надаються серверами, і множина клієнтів, які використовують дані сервіси. Клієнти повинні знати про доступні сервери, хоча і не можуть мати уя­в­лення про існування інших клієнтів. Як видно з рис. 1.4, на якій зображено мо­дель розподіленої архітектури клієнт/сервер, клієнти і сервери подають різні процеси.

Рис. 1.4 Система клієнт/сервер

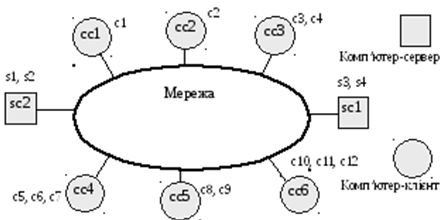
В системі між процесорами і процесами не обов’язково повинно втри­муватися відношення «один до одного». На рис. 1.5 зображено фізичну архітек­туру системи, що складається із шести клієнтських машин і двох серверів. В загаль­ному випадку, говорячи про клієнтів і сервери , маємо на увазі скоріше логічні процеси, ніж фізичні машини, на яких виконуються процеси.

Рис. 1.5 Комп’ютери в мережі клієнт/сервер

Архітектура клієнт/сервер повинна відображати логічну структуру прог­рамного додатку, який проектується .

На рис. 1.6 пропонується ще один погляд на програмні додатки, структурований у вигляді трьох рівнів.

Рівень подання забезпечує інформацію для користувачів і взаємодію з ними. Рівень виконання додатку реалізує логіку роботи додатку. Нарівні управління даними виконуються всі операції з даними. В централізованих системах між даними рівнями немає чіткого розподілу. Однак при проектуванні розподілених систем необхідно розділити ці рівні, щоб розкласти кожен рівень на різні комп’ютери.

Рис. 1.6 Рівні програмного додатку

Самою простою архітектурою клієнт/сервер є дворівнева, в якій додатки складаються із сервера (або множини серверів) і групи клієнтів. Існує два види такої архітектури рис. 1.7.

Рис. 1.7 Моделі товстого та тонкого клієнта

**Модель тонкого клієнта**. В даній моделі всі робота додатку і керування даними виконуються на сервері. На клієнтській машині виконується тільки ПЗ рівня подання.

**Модель товстого клієнта**. В даній моделі сервер тільки керує даними. На клієнтській машині реалізована робота додатку і взаємодія з користувачем системи.

Тонкий клієнт дворівневої структури – самий простий спосіб переведення існуючих централізованих систем в архітектуру клієнт/сервер. Користувацький інтерфейс в даних системах «переселяється» на персональний комп’ютер, а сам програмний додаток виконує функції сервера, тобто виконує всі процеси додатку і керує даними. Модель тонкого клієнта можемо також реалізувати там , де клієнти подають собою звичайні мережеві пристрої, а не персональні комп’ютери або робочі станції. Мережеві пристрої запускають Internet-браузер і користувацький інтерфейс, реалізований в середині системи.

Головний недолік моделі тонкого клієнта – велика завантаженість сервера та мережі. Всі обчислення виконуються на сервері, а це може призвести до значного мереженого трафіку між клієнтом і сервером. В сучасних комп’ютерах достатньо обчислювальної потужності, але вона практично не використовується в моделі тонкого клієнта.

Модель товстого клієнта використовує обчислювальну потужність ло­кальних машин: і рівень виконання додатку, і рівень подання розміщуються на клієнтський сервер. Сервер в даному випадку, за сутністю, є сервером транзакцій, який керує всіма трансакціями баз даних. Прикладом архітектури такого типу бути системи банкоматів, в яких банкомат є клієнтом, а сервер – центральним комп’ютером, який обслуговує базу даних розрахунку з клієнтами.

## Архітектура абстрактної машини

Модель абстрактної машини (багаторівнева модель) моделює взаємодію підсистем. Вона організовує систему у вигляді переліку рівнів, кожен з яких надає свої сервіси. Кожен рівень визначає абстрактну машину, машинна мова якої (сервіси, які надаються рівнем) використовуються для реалізації наступного рівня абстрактної машини. Наприклад, найбільш поширений спосіб реалізації мови програмування полягає у визначенні «мовної машини» і компіляції програм, які написані на даній мові, в код цієї машини. На наступному кроці трансляції код абстрактної машини конвертується на реальний машинний код.

Відомим прикладом такого підходу може бути OSI (Open SyStem Iterconnection – взаємодія відкритих систем, на основі семирівневої моделі протоколів передачі даних у відкритих системах) мережевих протоколів. Іншим прикладом є трирівнева модель середовища програмування на мові Ада.

Система адміністрування версій базується на управлінні версіями об’є­к­тів і надає засоби для повного управління конфігурацією системи. Для підтрим­ки засобів управління конфігурацією використовують систему адміністрування об’єктів, яка підтримує систему баз даних і сервіси управління об’єктами.

На рис. 1.8 зображена подібна модель і показано, як за допомогою моделі абстрактної машини можемо подати систему адміністрування версій.

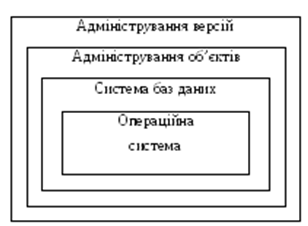


Рис. 1.8 Модель абстрактної машини

В свою чергу, в системі баз даних підтримуються різні сервіси, наприклад керування транзакціями, зміщення назад, відновлення та керування доступом. Для керування базами даних використовують засоби операційної системи і її файлова система.

Багаторівневий підхід забезпечує покроковий розвиток системи – при розробці будь-якого рівня сервіси які надаються є доступними користувачам. Крім того, така архітектура легко змінюється і встановлюється на різні платформи. Зміни інтерфейсу довільного рівня впливають тільки на суміжний рівень. Так в багаторівневий системах залежність від машинної платформи локалізовано на внутрішніх рівнях.

Недоліком багаторівневого підходу є доволі складна структура системи. Основні засоби, такі як керування файлами, необхідні всім абстрактним машинам, надаються внутрішніми рівнями. Тому сервісам, що запитуються користувачем, можливо, необхідний доступ до внутрішніх рівнів абстрактної машини. Така ситуація викликає руйнування моделі, оскільки зовнішній рівень залежить не тільки від попереднього рівня, але і від біль низьких рівнів.

Об’єкти можуть розміщуватися на різних комп’ютерах в мережі і взаємодіяти за допомогою проміжного ПЗ. За аналогією з системною шиною, яка дозволяє приєднувати різні пристрої і підтримувати взаємодію між апаратними засобами, проміжне ПЗ можемо розглядати як шину програмного забезпечення. Вона являє собою набір сервісів, які дозволяють об’єктам взаємодіяти один з одним, добавляти або видаляти їх із системи. Проміжне ПЗ називають брокером запитів до об’єктів. Його задача – забезпечити інтерфейс між об’єктами.

Нижче перераховано переваги архітектури розподілених об’єктів:

1. Розробники системи можуть не поспішати з прийняттям рішення відносно того, де і як будуть надаватися сервіси. Об’єкти, що надають сервіси, в довільному вузлі мережі. Тобто, різниця між моделями товстого і тонкого кліє­нтів стають несуттєвими, оскільки немає необхідності планувати розміщен­ня об’єктів для виконання об’єкту.
2. Системна архітектура достатньо відкрита, що дозволяє при необхідності добавляти в систему ресурси. Окрім того, зауважимо, що стандарти програмної шини постійно вдосконалюються, що дозволяє об’єктам, які написані на різних мовах програмування, взаємодіяти та надавати сервіси один одному.
3. Гнучкість та масштабованість системи. Для того щоб справитися з системними навантаженнями, можна створювати екземпляри системи з однаковими сервізами, які будуть надаватися різними об’єктами або різними екземплярами об’єктів. При збільшені навантаження, в систему можемо добавити нові об’єкти, не припиняючи при цьому роботу інших її об’єктів.
4. Існує можливість динамічної пере конфігурації системи засобами об’єктів, які мігрують в мережі за запитами. Об’єкти, які надають сервіси, можуть мігрувати на той же процесор, що і об’єкти, які запитують сервіси, тим самим підвищуючи продуктивність системи.

В процесі проектування систем архітектуру розподілених об’єктів можемо використовувати двояко:

* У вигляді логічної моделі, яка дозволяє розробникам структурувати і планувати систему. В цьому випадку функціональність додатку описується тільки в термінах і комбінаціях сервісів. В подальшому розробляються способи надання сервісів за допомогою декількох розподілених об’єктів. На цьому рівні, як правило, проектуються великомодульні об’єкти, які надають сервіси, що відображають специфіку конкретної області додатку. Наприклад, в програму обліку роздрібної торгівлі можемо включити об’єкти, які б вели облік стану запасів, відслідковували б взаємодію з клієнтами, проводили класифікацію товару та інше.
* Як гнучкий підхід до реалізації системи клієнт/сервер. В даному випадку логічна модель системи – це модель клієнт/сервер, в якій клієнти і сервери реалізовані як розподілені об’єкти, які взаємодіють засобами програмної шини. При такому підході легко замінити систему, наприклад дворівневу на багаторівневу. В даному випадку ні сервер, ні клієнт не можуть бути реалізовані в одному об’єкті, однак можуть складатися з множини невеликих об’єктів, кожен з яких надає певний сервіс.

Головним недоліком архітектур розподілених об’єктів є те, що їх складніше проектувати, ніж системи клієнт/сервер. Тобто, системи клієнт/сервер надають більш природний підхід до створення розподілених систем. В ньому відображаються взаємовідношення між людьми, при яких одні люди користуються послугами інших людей, які спеціалізуються на наданні конкретних послуг. Набагато складніше розробити систему у відповідності з архітектурою розподілених об’єктів, оскільки індустрія створення ПЗ поки що не має достатнього досвіду в проектуванні та розробці велико модульних об’єктів.

## Приклади реалізацій розподілених систем

## Система UNICORE

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

Архітектура UNICORE формується з клієнтського, сервісного та системного слоїв (рис. 1.9). Верхнім слоєм в архітектуру являєтся клієнтский слой. В ньому располагаются різні клієнти, забезпечувані взаїємодію 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 регістрацію та пошук ресурсів, доступних в грід-середовищі.



Рис. 1.9 Архітектура UNICORE 6

Також, на рівні сервисного слою 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, забезпечуючих взаємодію користувачів з ресурсами розрахункової мережі, а також розвинутих методів забезпечення безпеки при розробці грід-додатків.

Принцип роботи та функціональність грід-додатків 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.

## 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. 1.20 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. 1.20, 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.



Рис. 1.21 Cинтаксис Hadoop під управлінням MapR

Класична конфігурація кластера Hadoop, як на рис. 1.21, 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.

## 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. 1.22. Спочатку розроблявся для найбільшого проекта добровільних обрахунків— SETI@home, але потім розробники з Каліфорнійського університета 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 обрахунки потужності персональних комп’ютерів із всього світу.



Рис. 1.22 Схема роботи системи BOINC

BOINC розроблений командою в главі з Девідом Андерсоном (David Pope Anderson), очолювавшим також SETI@home, з Space Sciences Laboratory Каліфорнійського університета 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 — тілько 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 обрахунків. 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. 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.

## Висновки

1. З переглянутих вище систем, клієнт-серверні рішення мають найкращу пристосованість до змінної кількості вузлів під’єднаних до мережі та є невразливими до раптового відключення клієнтів. Також така модель дозволяє використовувати пристрої різних технічних характеристик.
2. Клієнт-серверні системи як Hadoop та BOINC є найкраще пристосовані до змінної топології мережі, але остання потребує специфічного графічного інтерпретатора і не має можливості до виконання різнопланових завдань, а лише «заточена» під обрахунки конкретних запрограмованих алгоритмів.
3. Hadoop є досить динамічною та гнучкою системою розподілених обрахунків. 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 Hadoop не має можливості для виконання на мобільних платформах через немале споживання ресурсів та несумісністю платформ.

# РОЗДІЛ 2

# Реалізації системи розподілених обчислень із застосуванням технології MapReduce

## Парадигма MapReduce

MapReduce - парадигма розподілених обчислень, представлена компанією Google, яка використовується для паралельних обчислень над дуже великими, кілька петабайт, наборами даних в комп'ютерних кластерах.

Робота MapReduce складається з двох кроків: Map і Reduce, названих так за аналогією з однойменними функціями вищого порядку, map і reduce.

Перевага MapReduce полягає в тому, що він дозволяє розподілений виробляти операції попередньої обробки і згортки. Операції попередньої обробки працюють незалежно один від одного і можуть проводитися паралельно (хоча на практиці це обмежена джерелом вхідних даних і / або кількістю використовуваних процесорів).

Аналогічно, безліч робочих вузлів можуть здійснювати згортку - для цього необхідно лише щоб все результати попередньої обробки з одним конкретним значенням ключа оброблялися одним робочим вузлом в один момент часу. Хоча цей процес може бути менш ефективним в порівнянні з більш послідовними алгоритмами, MapReduce може бути застосований до великих обсягів даних, які можуть оброблятися великою кількістю серверів. Так, MapReduce може бути використаний для сортування петабайта даних, що займе всього лише кілька годин.

Програми, що використовують реалізацію MapReduce, автоматично розпаралелювать і виконуються на кластері, що складається з безлічі пов'язаних між собою комп'ютерів. Виконавча система сама піклується про деталі розбиття вхідних даних на частини, плануванні виконання програми на наборі машин, обробці збоїв і управлінні необхідним сполученням між машинами. Це дозволяє програмістам навіть без досвіду роботи з паралельними і розподіленими системами з легкістю використовувати ресурси великих розподілених систем.

## Паралелізм в MapReduce

Паралелізм дає деякі можливості відновлення після часткових збоїв серверів: якщо в робочому вузлі, що виробляє операцію попередньої обробки або згортки, виникає збій, то його робота може бути передана іншому робочому вузлу (за умови, що вхідні дані для проведеної операції доступні).

До паралелізму MapReduce легко можна застосовувати відому парадигму інформатики «розділяй і володарюй», поділивши інформаційний масив на частини і віддавши кожну з них індексувати єдиного сервера. Ну а після виконання індексації по частинах залишається зібрати знайдене рішення воєдино. Таке архітектурне рішення забезпечує [14]:

* автоматичне розпаралелювання даних з величезного масиву по безлічі вузлів обробки, виконують процедури Map / Reduce;
* ефективну балансування завантаження цих обчислювальних вузлів, що не дає їм простоювати або бути перевантаженими надміру;
* технологію відмовостійкої роботи, що передбачає той факт, що при виконанні загального завдання частина вузлів обробки може вийти з ладу або з якої-небудь іншої причини перестати обробляти дані.

Таким чином, MapReduce, з одного боку, надає користувачеві процедури обробки його даних, а з іншого - робить для нього прозорим процес розпаралелювання цієї обробки на могутньому кластері Google. Найбільш світлою думкою при проектуванні MapReduce була ідея розмістити модулі, що реалізують процедури map і reduce, на тих самих чанк-серверах - основі файлової системи GFS. Такий підхід наближає зберігаються в GPS модулі до функцій їх обробки. Економія мережевого трафіку в наявності.

## Рівні паралелізму в MapReduce

Програми часто класифікуються відповідно до того, як часто їхні підзадачі мають синхронізуватись чи спілкуватись один з одним. Програма проявляє дрібнозернистий паралелізм якщо її підзадачі мають обмінюватись даними багато разів на секунду; вона проявляє крупнозернистий паралелізм, якщо вони не мусять обмінюватись даними багато разів на секунду, і вона проявляє приголомшливий паралелізм, якщо вони рідко, чи взагалі ніколи не мають обмінюватись даними. Приголомшливо паралельні програми вважаються такими що розпаралелюються найлегше.

## Паралелізм бітового рівня

З винайденням у 1970-тих технології створення надвеликих інтегральних схем прискорення в комп'ютерній архітектурі відбувалось з допомогою подвоєння розміру машинного слова — кількості інформації, яку комп'ютер може обробляти за один цикл [15]. Збільшення розміру слова зменшує кількість інструкцій необхідних виконати операцію над даними чий розмір більший ніж розмір вхідного слова. Наприклад коли восьмибітний процесор має додати два шістнадцятирозрядні числа, процесор має спочатку додати 8 біт нижчого розряду з кожного числа, використовуючи стандартну інструкцію додавання, потім додати 8 бітів вищого розряду, використовуючи інструкцію додавання з переносом та біт переносу від виконання попереднього додавання. Тому восьмибітний процесор потребує дві інструкції для виконання однієї операції, в той час як шістнадцятибітний лиш одну.

Історично, чотирьохрозрядні процесори були замінені на восьмирозрядні, потім на шістнадцятирозрядні, потім на 32-х розрядні. Ця тенденція припинилась з введенням тридцятидвохрозрядних процесорів, які стали стандартом для персональних комп'ютерів на два десятиліття. Аж поки недавно (2003—2004), з винайденням архітектури x86-64, не з'явились 64-x розрядні процесори.

## Паралелізм рівня інструкцій

Комп'ютерна програма, по суті, є потоком інструкцій, що виконуються процесором. Іноді ці інструкції можна перевпорядкувати, та об'єднати в групи, які потім виконувати паралельно, без зміни результату роботи програми, що відомо як паралелізм на рівні інструкцій. Такий підхід до збільшення продуктивності обчислень переважав з середини 80-тих до середини 90-тих.

Сучасні процесори мають багатоетапні конвеєри команд. Кожен етап конвеєра відповідає іншій дії, що виконує процесор. Процесор що має конвеєр з N-ступенями, може одночасно обробляти N інструкцій, кожну на іншій стадії обробки. Класичним прикладом процесора з конвеєром є процесор архітектури RISC, що має п'ять етапів: завантаження інструкції, декодування, виконання, доступ до пам'яті, та запис результату. Процесор Pentium 4 має конвеєр з 35 етапами.

На додачу до паралелізму на рівні інструкцій деякі процесори можуть виконувати більш ніж одну інструкцію за раз. Вони відомі як суперскалярні процесори. Інструкції групуються разом, якщо між ними не існує залежності даних. Щоб реалізувати паралелізм на рівні інструкцій використовують алгоритми Scoreboarding та Tomasulo algorithm (який аналогічний до попереднього, проте використовує перейменування регістрів).

## Паралелізм даних

Паралелізм даних — це паралелізм властивий циклам програм, які фокусуються на доставці даних різним обчислювальним вузлам для паралельної обробки. "Розпаралелювання циклів часто приводить до подібних (не обов'язково ідентичних) послідовностей операцій, чи обчислення функцій над елементами великих структур даних. Багато наукових, та інженерних програм проявляють паралелізм даних.

Циклічна залежність — залежність ітерації циклу, від результатів попередньої, чи кількох попередніх ітерацій. Циклічні залежності перешкоджають розпаралелювання циклів. Розглянемо псевдокод, що обчислює кілька перших чисел Фібоначчі:

Такий цикл не може бути розпаралелений, бо CUR залежить від себе (PREV2), та PREV1, які обчислюються в кожній ітерації. Тому, кожна ітерація залежить від результатів попередньої, вони не можуть виконуватись паралельно. Коли розмір задачі стає більшим, кількість доступних для розпаралелювання даних зазвичай теж зростає.

## Паралелізм задач

Паралелізм задач — характеристика паралельної програми, яка полягає в тому, що «цілком різні обчислення можуть виконуватись над одними, чи різними даними». Це відрізняє паралелізм задач від паралелізму даних, при якому одне і те ж обчислення виконується над одними і тими ж даними. Паралелізм задач, зазвичай не зростає зі зростанням розміру задачі.

## Реалізація алгоритмів в MapReduce

На вхід MapReduce надходить вимагає обробки масив, «розрізаний» на M (по числу мепперов) частин розміром від 16 до 64 мегабайт (варто нагадати, що саме такий розмір має чанк в файлової системі GFS). Отримавши адреси M елементів масиву, Master MapReduce формує приватні завдання для M функцій мепперов і роздає кожній з них адреса чанка, який належить піддати процедурі map. Оскільки меппери працюють паралельно і незалежно один від одного, потрібно в M разів менше часу, ніж при лінійної обробці.

Канонічний приклад програми, написаної за допомогою MapReduce, - це процес, що підраховує, скільки разів різні слова зустрічаються в наборі документів:

// Функція, яка використовується робітниками нодами на Map-кроці

// для обробки пар ключ-значення з вхідного потоку

void map(String name, String document):

// Вхідні дані:

// name - назва документу

// document - вміст документа

for each word w in document:

EmitIntermediate(w, "1");

// Функція, яка використовується робітниками нодамі на Reduce-шаге

// для обробки пар ключ-значення, отриманих на Map-шаге

void reduce(String word, Iterator partialCounts):

// Вхідні дані:

// word - слово

// partialCounts - список групувати проміжних результатів. Кількість записів в partialCounts і є

// необхідне значення

int result = 0;

for each v in partialCounts:

result += parseInt(v);

Emit(AsString(result));

У цьому коді на Map-кроці кожен документ розбивається на слова, і повертаються пари, де ключем є саме слово, а значенням - «1». Якщо в документі одне і те ж слово зустрічається кілька разів, то в результаті попередньої обробки цього документа буде стільки ж цих пар, скільки разів зустрілося це слово.

Бібліотека об'єднує всі пари з однаковим ключем і передає їх на вхід функції reduce, якій залишається скласти їх, щоб отримати загальну кількість входжень даного слова до всіх документів. Виклики операції Map розподілені між безліччю машин за допомогою автоматичного розподілу вхідних даних на набір з M частин. Вхідні частини, що на рис. 2.1, можуть оброблятися паралельно декількома машинами. Виклики операції Reduce розподілені поділом проміжного простору ключів на R частин за допомогою функції розподілу (наприклад, hash (key) modR). Кількість частин R і функція поділу задаються користувачем. На схемі показаний повний процес операції MapReduce в реалізації Google. Коли програма користувача викликає функцію MapReduce, відбувається наступна послідовність операцій (їх номери в схемі відповідають списку нижче):

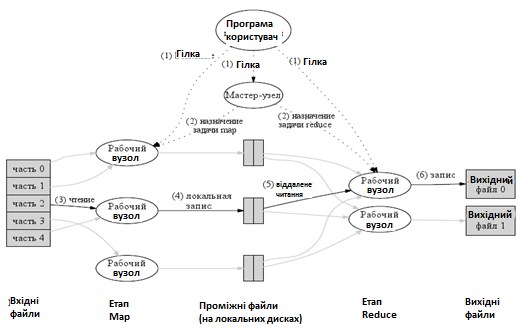


Рис. 2.1 Детальний огляд роботи реалізації MapReduce

Стадії обробки нумеруються від отримання завдання і до видачі кінцевого результату за таким алгоритмом:

1. Спочатку бібліотека MapReduce в програмі користувача ділить вхідні файли на M частин, кожна з яких зазвичай займає від 16 до 64 мегабайт (цей параметр може задаватися користувачем). Потім вона запускає безліч копій програми на кластері.
2. Одна з копій програми (master) особлива, вона задає роботу іншим екземплярам (workers). Всього потрібно задати M завдань map і R завдань reduce. Майстер шукає неактивні робочі екземпляри і призначає кожній з них одне завдання.
3. Робочий екземпляр програми, якому було присвоєно завдання map, читає вміст відповідної частини вхідних даних, розбирає його і передає кожен елемент функції користувача Map. Проміжні пари «ключ-значення» потім зберігаються в пам'яті.
4. Через деякі проміжки часу збережені пари записуються на локальний диск і розбиваються на R областей функцією розподілу. Місцезнаходження цих пар на диску передаються назад майстру, який відповідає за подальше повідомлення цих місць розташування робочим екземплярів.
5. Коли робочий екземпляр оповіщається про місцезнаходження проміжних даних, він читає дані з локальних дисків примірників, які застосовували функцію map. Коли всі дані прочитані, вони сортуються по ключу і групуються разом. Якщо обсяг даних занадто великий, використовується зовнішня сортування.
6. Робочий екземпляр проходить по відсортованим проміжним даними і передає функції Reduce кожен унікальний ключ і відповідний йому список значень. Результат приєднується до кінцевого файлу для цієї частини проміжних даних.
7. Коли всі завдання map і reduce будуть виконані, виклик MapReduce буде завершений, і відбудеться повернення назад до призначеного для користувача коду.

Після успішного завершення обробки вихідні дані будуть доступні в R результуючих файлів (один файл на кожну задачу reduce) з назвами, заданими користувачем. Зазвичай користувачам не потрібно з'єднувати ці R файлів в один - часто буває, що ці файли передаються черговому виклику MapReduce вже в якості входу або обробляються в інших розподілених додатках, яким на вхід також передаються кілька файлів.

## Продуктивність розподілених алгоритмів в MapReduce

Програми, що використовують MapReduce, не завжди будуть працювати швидко. Головною перевагою цієї моделі програмування є оптимізоване розподіл даних між вузлами і невелика кількість коду, яке потрібно написати програмісту. Однак на практиці користувач програми повинен прийняти до уваги етап розподілу даних, зокрема, функція поділу даних і кількість даних на виході функції Map можуть дуже сильно впливати на продуктивність. Додаткові модулі, такі як функція Combiner, можуть допомогти зменшити кількість даних, що записуються на диск і через мережу.

При написанні програми користувач повинен знайти і вибрати хороший компроміс між обчислювальної та комунікаційної складністю. Комунікаційна складність перевершує над обчислювальною складністю, і багато реалізації MapReduce були розроблені, щоб записувати відомості про всі комунікаціях в розподілене зберігання для аварійного відновлення.

Для завдань, які вирішуються швидко на нерозподілених системах, а вхідні дані містяться в оперативну пам'ять одного комп'ютера або невеликого кластера, використання фреймворка MapReduce неефективно. Так як ці фреймворки розроблені, щоб мати можливість відновлення цілих вузлів кластера під час обчислень, вони записують в розподілене сховище проміжні результати роботи. Такий захист від збоїв - дуже дорога процедура і окупається, тільки коли в обчисленнях бере участь безліч комп'ютерів, а при виході одного з них з ладу найпростіше перезапустити присвоєну йому завдання на іншому вузлі.

## Надійність розподілених алгоритмів в MapReduce

Надійність MapReduce досягається за рахунок розподілу операції обробки даних по всіх вузлах мережі. Майстер періодично опитує кожен робочий вузол для отримання статусу або результату роботи. Якщо вузол не відповідає протягом встановленого часу, майстер починає вважати його аварійним і призначає його роботу іншого вузла.

У вузлах застосовуються атомарні операції іменування вихідних файлів для перевірки, що в кожен момент не запущені конфліктуючі паралельні потоки. Так як операція згортки погано розпаралелюється, і саме в вузлі зберігається обробляється частина даних, майстер робить спроби запуску згортки на тому ж самому або на прилеглих вузлах.

Не всі реалізації MapReduce обов'язково мають високу надійність. Наприклад, в ранніх версіях Hadoop примітив майстер-вузла NameNode був єдиною точкою відмови розподіленої файлової системи.

## Функціональне програмування в MapReduce

Функціональне програмування - розділ дискретної математики і парадигма програмування, в якій процес обчислення трактується як обчислення значень функцій в математичному розумінні останніх (на відміну від функцій як підпрограм в процедурному програмуванні).

Парадигма MapReduce підтримує концепцію функціонального програмування, адже вона має в собі дві функції вищого порядку попередньої обробки і згортки, які викликаються за необхідністю (за появою завдання і за появою результатів обробки).

Функціональне програмування передбачає обходитися обчисленням результатів функцій від вихідних даних і результатів інших функцій, і не передбачає явного зберігання стану програми. Відповідно, не передбачає воно і змінність цього стану (на відміну від імперативного, де однією з базових концепцій є змінна, що зберігає своє значення і дозволяє змінювати його в міру виконання алгоритму).

На практиці відмінність математичної функції від поняття «функції» в імперативному програмуванні полягає в тому, що імперативні функції можуть спиратися не тільки на аргументи, а й на стан зовнішніх по відношенню до функції змінних, а також мати побічні ефекти і змінювати стан зовнішніх змінних. Таким чином, в імперативному програмуванні при виклику однієї і тієї ж функції з однаковими параметрами, але на різних етапах виконання алгоритму, можна отримати різні дані на виході через вплив на функцію стану змінних. А в функціональному мовою при виконанні функції з одними і тими ж аргументами ми завжди отримаємо однаковий результат: вихідні дані залежать тільки від вхідних. Це дозволяє середах виконання програм на **функціональних мовами** кешувати результати функцій і викликати їх в порядку, що не визначається алгоритмом і розпаралелювать їх без будь-яких додаткових дій з боку програміста (що забезпечують функції без побічних ефектів - чисті функції).

## Висновки

1. Парадигма MapReduce задовольняє основні вимоги представлені для гетерогенної системи з перемінним числом вузлів та змінним станом мережі.
2. Реалізація алгоритму роботи розподіленої системи за MapReduce дозволяє ділити завдання на які завгодно маленькі частинки, що пришвидшує отримання результату від мобільного вузла. Дрібністю завдань можна легко керувати кодом, описавши функцію map.
3. Принцип паралельності даних допоможе архітектурно відділити один вузол від іншого, що в свою чергу зводить до нуля вплив одного пристрою на інший. Це допоможе домогтися необхідної надійності системи, щоб при відключенні одного вузла, це ніяк не буде відображатися на інших.
4. Використання функціонального програмування в коді розподіленої системи дозволить збільшити читабельність коду та інкапсулювати данні виконання одного завдання від іншого. Це вирішує проблему фіктивних даних, коли при виконанні програми значення змінних можуть підмінитися іншою програмою та спричинити до виведення неправдивих результатів в систему. Також це дає змогу обробляти частину завдання незалежно від результату виконання в інших частинах (якщо це необхідно) , забезпечивши покращене розпаралелювання завдання між вузлами.

# РОЗДІЛ 3

# РЕАЛІЗАЦІЇ МЕТОДІВ БАЛАНСУВАННЯ

Розділ розглядає опис створених класів для реалізації методу MapReduce. Результат застосування методу в розподіленій системі на мобільній платформі. Результати обрахунків завдань введених в систему і використані вхідні параметри.

Для дослідження та моделювання завдань з обслуговування черг припустимо, що число агрегованих по класах або пріоритетам потоків в структурі NGN відомо і дорівнює M, що відповідає прийнятим на практиці рішенням в рамках відомих методів маркування пакетів у полях заголовків Tos, DSCP. Також приймемо до уваги, що максимальне число черг 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 ai (i =1,2, ...M) – інтенсивність вхідного трафіку i -го класу, який надходить на обслуговування мережевим пристроєм. Крім цього, нехай bj (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 bj (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 відсутності перевантаження каналу зв'язку в момент часу t:

,(1)

де b - пропускна здатність вихідного каналу зв’язку, bj(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:

, (2)

Виконання умови (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 xij , під якої мовити частка i -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 nj і njmax (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.

**Модель системи оброблення інформаційних потоків за удосконаленим алгоритмом управління черг**

1. Представлений оригінальний алгоритм керування чергами інформаційних потоків, 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 ∆tm , 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 ∆tmmax, (часу максимальної затримки рафіку p-ї пріоритетної групи).
2. 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 ∆tm та ∆tmmax , на основі даних про них формуються черги. Відомо, що ∆tm < ∆tmmax. Пакети, що знаходяться в черзі довше ∆tm 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.
3. Тобто всі пакети, що потрапили на місця черг в інтервалі [Np(m), Nplim(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.
4. 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.

**Імітаційна модель системи оброблення інформаційних потоків за удосконаленим алгоритмом управління черг у маршрутизаторах**

1. Модифікований метод управління чергами в мультисервісних вузлах телекомунікаційної мережі. Одна з ключових відмінностей пропонованого рішення полягає в тому, що ведеться контроль за часом перебування в чергах пакетів і, 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.

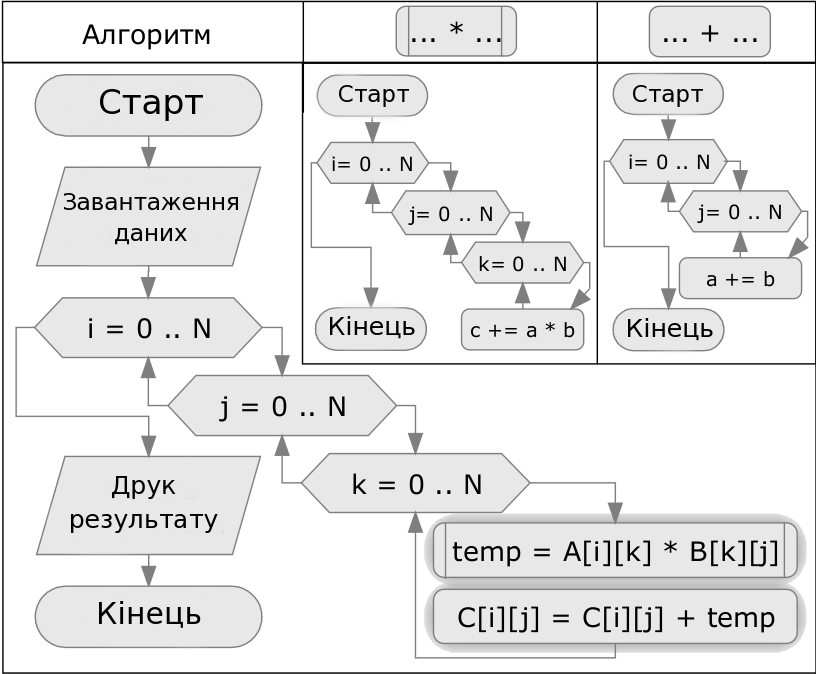
Це в свою чергу призводить до вимушеного відкидання пакетів згідного алгоритму WRED та виникнення значних втрат пакетів голосу, що вплине на якість розмови користувачів в NGN мережах. Внаслідок застосування удосконаленого алгоритму та із вище запропонованою моделю обслуговування, зменшується кількість 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.

1. 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.

**Вимоги до оформлення інтерфейсу для розпаралелювання послідовних програм**

1. Система DDCI, що реалізує підхід прозорого автоматичного динамічного розпаралелювання обчислень для блокових послідовних алгоритмів, вже зарекомендовала 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.
2. 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:

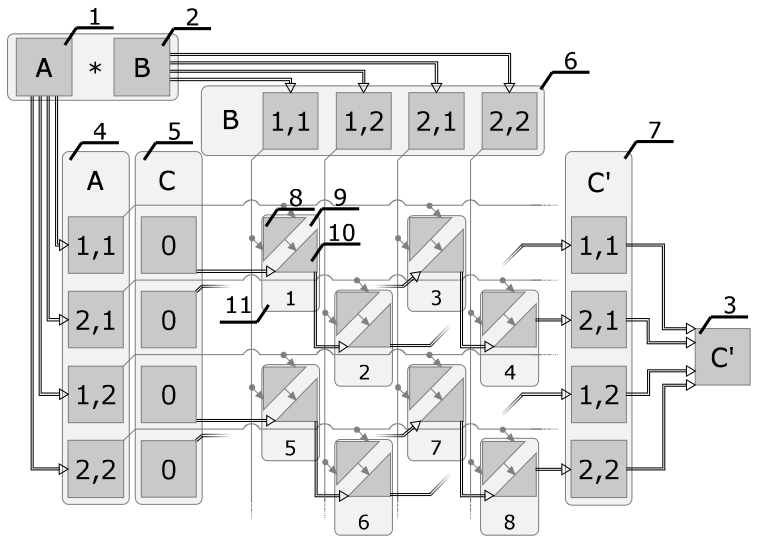
* має поліноміальних складність завдання O(n3) в залежності від розмірності матриць
* для завдання часто доводиться використовувати паралельну реалізацію
* блок-схему алгоритму легко аналізувати (рис. 1)

  
Illustration 1: Блок-схема блочного алгоритму перемноження двох квадратних матриць

1. На рис. 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.
2. «Алгоритм», в подальшому «керуючий код». Дана частина алгоритму в післяпослідовно 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, специализированне дослідницьке обладнання. Наслідком є ​​неможливість виконання деяких команд даної частини алгоритму на віддалених обчислювальних станциях, а, значить, і неможливість неконтрольованого автоматичного розпаралелювання програми.
3. Другою відмітною особливістю даної частини алгоритму є споживана обчислювальна потужність. Якщо в блок-схемі виклики розрахункових функцій
4. «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:
5. 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.
6. 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 распараллелівать системи спеціальних директив, неможливо.
7. Якщо всі основні обчислення распараллелівать блочного алгоритму знаходяться в чистих функціях, то виклики функцій можна замінити «заглушками», а автоматичне розпаралелювання цій частині алгоритму є необов'язковим.
8. 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 распараллеливанию.
9. Результатом зроблених висновків є формулювання першого вимоги для послідовних алгоритмів: все автоматично динамічно распаралелені 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.
10. «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:
11. Враховано всі обмеження для «чистої функції», згідно з якими функція може працювати тільки з тими даними, які були передані в якості параметрів.
12. 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.
13. 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.
14. «Додавання», в подальшому сполучні функції. Такі функції в попередніх авторських роботах розглядалися як окремі випадки рідко використовуваних низькоефективних чистих функцій. На рішення виділити їх в окремий клас вплинули наступні моменти:
15. Такі функції завжди мають низьку ефективність, яка призводить до уповільнення роботи распараллелівать програми, причому твердження вірне, тільки якщо вважати їх розрахунковими функціями і використовувати для них такі ж методи планування.
16. 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.
17. 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 межпроцессорного трафіку.
18. В результаті можна зробити наступний висновок: виділення класу сполучних функцій може підвищити ефективність роботи автоматично распараллелівать програм за умови, що надасть права мати окремі методи планування для виконання функцій даного типу. Також це призводить до додаткових інтерфейсним вимогам, 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.

**Управління динамічним графом розпаралелювання частини програми**

1. Розглянемо процес формування графа распараллелівать 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 межпроцессорного 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.
2. 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. 3).

  
рис 3: Граф розпаралелювання частини блочного алгоритму перемноження двох матриць

1. Центральна частина графа, що складається з однотипних елементів під номером «9», формується в процесі виконання трьох циклів з «Міткою-3» в блокхеме 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.
2. 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 распараллеливанию безглуздо через мінливої форми графа.
3. 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.

**Планувальник**

1. 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.
2. 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.

1. З метою ефективного управління розглянутими ресурсами пропонується
2. створити кілька черг з різним пріоритетом для можливості ефективного розподілу всіх наявних ресурсів і мінімізації простоїв.
3. 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.

1. 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.

1. Для ефективного використання всіх ресурсів також пропонується розбити планувальник на три незалежні системи: активний планувальник, пасивний планувальник і система фонового дублювання даних.
2. 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.
3. 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
4. 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.
5. 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.

# 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.

## 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 нетральних характеристик ідеї̈ проекту

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No п/п | Техніко- економічні характеристики ідеї | (потенційні) товари/концепції конку- рентів | | | W (слабка сторона) | N (нейтра- льна сторона) | S (сильна сторона) |
| Мій проект | Hadoop | 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.

## 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.

## 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 обрахунків, 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 ідеї̈ 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 |  |  | ✓ |  |  |  |  |  |
| 2 | Концепція товару і послуги |  |  |  | ✓ |  |  |  |  |
| 3 | Після продажне обслуговування |  |  |  |  |  |  | ✓ |  |

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

Таблиця 3.12

SWOT-аналіз стартап проекту

|  |  |
| --- | --- |
| Сильні сторони:   * Інноваційні технології * Висока якість | Слабкі сторони:   * Слабкий імідж компанії * Мало оборотних коштів |
| Можливості:   * Нові технології * Нові потреби клієнтів * Тенденції попиту | Загрози:   * 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 конкурентної̈ боротьби, необхідно обрати одну з трьох стратегій̆ 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 за для економії фінансових ресурсів |

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії̈) та до продукту (таблиця 3.5), а також в залежності від обраної̈ базової̈ стратегії̈ розвитку та стратегії̈ конкурентної̈ 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 |

## 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 маркетингової̈ концепції̈ товару, який̆ 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 обрахунків на мобільних пристроях |
| 2. Товар у реально-му 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 сalculation 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

# ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО МЕТОДУ

Розділ розглядає опис створених класів для реалізації методу MapReduce. Результат застосування методу в розподіленій системі на мобільній платформі. Результати обрахунків завдань введених в систему і використані вхідні параметри.

## Реалізація архітектури мережі

На першому етапі рішення задачі був спроектований граф 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». Сервер має стабільне з’єднання з мережею Інтернет, через яку він спілкується (віддає завдання та отримує 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.

## Реалізація алгоритму MapReduce

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).

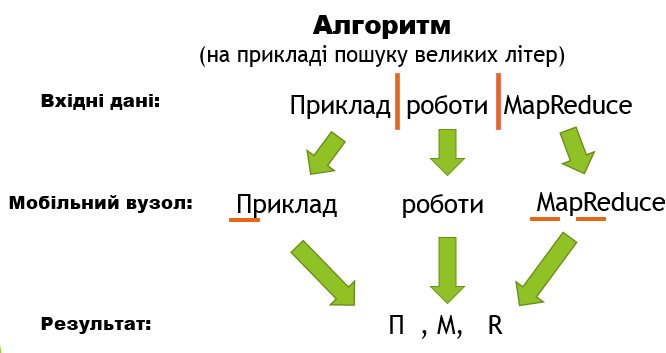


Рис. 4.2 Приклад виконання алгоритму для пошуку великих літер

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.

## 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.

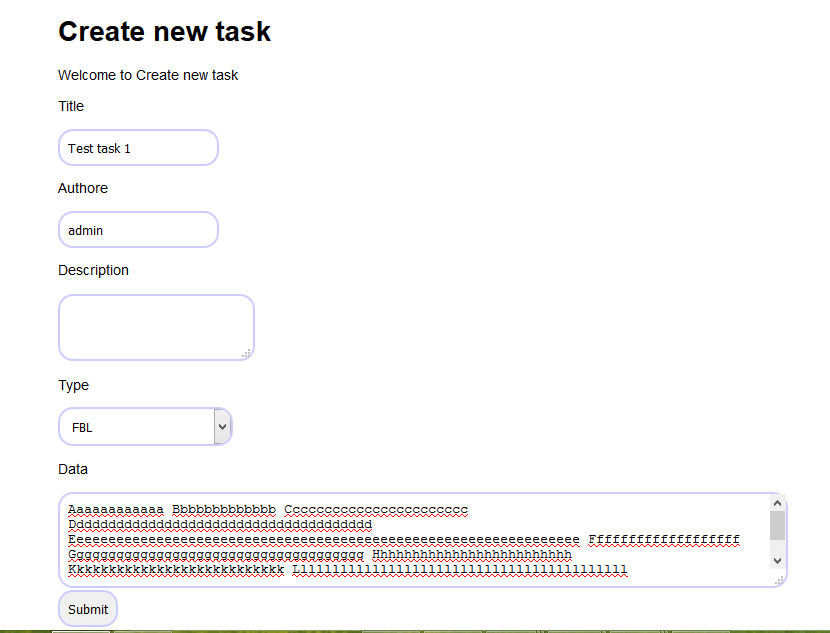


Рис. 4.3 Сторінка створення завдання

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

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.

## 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.

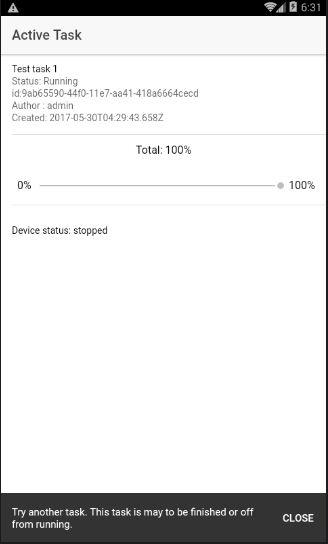


Рис. 4.5 Приклад додатку на мобільному пристрої

## Висновки

1. Створено ПЗ для організації розподіленої системи, прикладну програму для обробки вхідних завдань та «склеювання» результату воєдино на сервері, та окремий мобільний додаток, який виконує функції реєстрації пристрою, отримання інформації по завданню, відправлення обробленої інформації на сервер.

# ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1. Проведено детальний аналіз метод розподілення задач гетерогенної системи розподілених обчислень для ефективного використання мобільних вузлів.
2. Було досліджено різни алгоритми роботи розподілених систем. Серед цих алгоритмів був виділений MapReduce, адже він задовольняє ряду вимог, які були поставлені перед системою, як гетерогенною мережею із мобільними пристроями в якості робочих вузлів.
3. Обраний алгоритм 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; керування дрібністю ділення завдання, для досягання виконання мобільним пристроєм найпростіших операцій, задля зменшення навантаження на акумулятор пристрою та зменшення ризику втрати великої частини обрахунків через раптове відключення вузла від мережі Інтернет.
4. Було розроблено ПЗ для встановлення на сервер та на кожен мобільний вузол, що знаходиться в мережі. Серверна програма представляє собою веб-інтерфейс який дозволяє користувачу створювати та переглядати стан завдань поставлених на обрахування. 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 ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Zhukov І. 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](http://evrika.tsi.lv/index.php?name=texts&file=show&f=410) / Ф. Уоссермен — Мир, 1992. — 240 с. — [ISBN 5-03-002115-9](https://uk.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0:%D0%94%D0%B6%D0%B5%D1%80%D0%B5%D0%BB%D0%B0_%D0%BA%D0%BD%D0%B8%D0%B3/5030021159).
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](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/0201876868).
9. Таунсенд К. [Проектирование и программная реализация экспертных систем на персональных ЭВМ / К. Таунсенд, В. А. Кондратенко, С. В. Трубицына.](http://web.archive.org/web/20060525230925/http:/www.forth.org.ru/~kp/taunsend.zip) — 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-e 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)errorHandler(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)errorHandler(err,'updating device id in part error',part)});

} })}