**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

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

Інститут телекомунікаційних систем

(повна назва інституту/факультету)

Кафедра «Інформаційно-телекомунікаційних мереж»

(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_ Л.С.Глоба

(підпис) (ініціали, прізвище)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2017 р.

**Дипломна робота**

**на здобуття ступеня бакалавра**

з напряму підготовки 6.050903 «Телекомунікації»

(код і назва)

на тему: Розподілені обрахунки методом MapReduce на мобільній платформі

Виконав (-ла): студент (-ка) \_4\_\_\_ курсу, групи \_\_\_ТІ-32\_\_\_\_

(шифр групи)

Павленко Владислав Миколайович

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

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 дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ – 2017 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 – перший (бакалаврський)

Напрям підготовки 6.050903 «Телекомунікації та радіотехніка»

(код і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_ Л.С.Глоба

(підпис) (ініціали, прізвище)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 р.

**ЗАВДАННЯ**

**на дипломну роботу студенту**

Павленку Владиславу Миколайовичу

(прізвище, ім’я, по батькові)

1. Тема роботи: Розподілені обрахунки методом MapReduce на мобільній платформі ,

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

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «11» квітня 2017 р. №1321-с

2. Термін подання студентом роботи 13 червня 2017 р.

3. Вихідні дані до роботи:

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

4. Зміст роботи

1. Провести аналіз вже створених систем розподілених обрахунків для персональних комп’ютерів.

2. Запропонувати модель управління завданнями для розподіленого обрахування.

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

5. Дата видачі завдання 10 вересня 2016 року.

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання  дипломної роботи | Термін виконання  етапів роботи | Примітка |
| 1 | Дослідження та вивчення отриманого завдання | 10.09.16 – 01.10.16 |  |
| 2 | Аналіз методології побудов систем розподілених обрахунків | 01.10.16 – 09.11.16 |  |
| 3 | Аналіз існуючих рішень розподілених обрахунків | 09.11.16 – 12.01.17 |  |
| 4 | Аналіз підходів до розробки систем розподілених обрахунків | 12.01.17 – 16.02.17 |  |
| 5 | Підготовка тезисів (ПТ-17) | 01.02.17 – 01.03.17 |  |
| 6 | Створення прототипу розподіленої системи, опис компонентів | 01.03.17 – 01.04.17 |  |
| 7 | Розробка алгоритму балансування задач між мобільними пристроями | 01.11.16 – 01.03.17 |  |
| 8 | Розробка алгоритму визначення закінчення роботи над завданням | 01.03.17 – 26.03.17 |  |
| 9 | Розробка розподіленої системи | 01.11.16 – 01.05.17 |  |
| 10 | Підготовка тексту диплому | 15.04.17 – 12.06.17 |  |

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

(підпис) (ініціали, прізвище)

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

(підпис) (ініціали, прізвище)

**РЕФЕРАТ**

Робота містить 60 сторінки, 14 рисунків. Було використано 15 джерел

**Мета роботи**: Запропонувати модель гетерогенної системи розподілених обрахунків для використання на мобільних платформах.

Проведено аналіз алгоритмів балансування задач, розглянуто переваги та приклади їх використання в існуючих програмних продуктах.

Розробити модель розподіленої системи враховуючи особливості реалізації на мобільних платформах. Сформовано порівняльну характеристику різних алгоритмів розрахунку.

Створити модель для демонстрування роботи розробленої системи та використаних алгоритмів.

**Ключові слова:** розподілені обрахунки, гетерогенна система, алгоритм балансування завдань.

**ABSTRACT**

The work contains 60 pages, 14 figures, 15 sources have been used.

**Objective**: Suggest distributed heterogeneous system model calculations for using on mobile platforms.

The analysis of balancing algorithms, discussed advantages and examples of their use in existing software products.

Develop distributed computing model including analyzed information using the principles of distributed calculation. Formed a comparison characteristics of different calculation algorithms.

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

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

Зміст

[ВСТУП 10](#_Toc484511784)

[РОЗДІЛ 1](#_Toc484511785) [Провести аналіз принципів побудови систем розподілених обрахунків та виконати огляд існуючих зразків їх реалізації. 12](#_Toc484511786)

[1.1 Розподілені обчислення. Мета та цілі їх застосування 12](#_Toc484511787)

[1.2 Види розподілених архітектур 17](#_Toc484511788)

[1.2.1 Модель репозиторія MapReduce 17](#_Toc484511791)

[1.2.2 Модель клієнт/сервер 20](#_Toc484511792)

[1.2.3 Модель абстрактної машини 25](#_Toc484511793)

[1.3 Приклади реалізацій розподілених систем 29](#_Toc484511794)

[1.3.1 Система UNICORE 29](#_Toc484511795)

[1.3.2 Модель репозиторія 31](#_Toc484511796)

[1.4 Система BOINC 34](#_Toc484511797)

[Висновки 39](#_Toc484511798)

[РОЗДІЛ 2](#_Toc484511799) [Запропонувати алгоритм реалізації системи розподілених обрахунків із застосуванням технології MapReduce. 40](#_Toc484511800)

[2.1 Парадигма MapReduce 40](#_Toc484511803)

[2.2 Паралелізм в MapReduce 41](#_Toc484511804)

[2.3 Рівні паралелізму в MapReduce 42](#_Toc484511805)

[2.3.1 Паралелізм бітового рівня 42](#_Toc484511806)

[2.3.2 Паралелізм рівня інструкцій 43](#_Toc484511807)

[2.3.3 Паралелізм даних 43](#_Toc484511808)

[2.3.4 Паралелізм задач 44](#_Toc484511809)

[2.4 Реалізація алгоритмів в MapReduce 44](#_Toc484511810)

[2.4.1 Продуктивність таких алгоритмів 47](#_Toc484511811)

[2.4.2 Надійність таких алгоритмів 48](#_Toc484511812)

[2.5 Функціональне програмування в MapReduce 49](#_Toc484511813)

[Висновки 50](#_Toc484511814)

[РОЗДІЛ 3](#_Toc484511815) [Програмна реалізація запропонованого методу 51](#_Toc484511816)

[3.1 Реалізація архітектури мережі 51](#_Toc484511818)

[3.2 Реалізація алгоритму MapReduce 52](#_Toc484511819)

[3.3 Обробка завдань на сервері 53](#_Toc484511820)

[3.4 Обробка завдань на мобільному пристрої 55](#_Toc484511821)

[Висновки 56](#_Toc484511822)

[ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 57](#_Toc484511823)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 58](#_Toc484511824)

[Додаток А Код класу MapReduce мови JavaScript 60](#_Toc484511825)

**перелік скорочень**

|  |  |
| --- | --- |
| MongoDB | Mongoose Data Base |
| 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. Беручи до уваги неоднорідність структури та різні ОС на пристроях, технологія MapReduce добре підходить для реалізації розподіленої системи.

**Об’єкт роботи:** мобільна платформа.

**Предмет роботи:** побудова системи розподілених обрахунків.

**Мета роботи**: підвищення ефективності розподілених систем за рахунок використання мобільних пристроїв.

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

1. Провести аналіз принципів побудови систем розподілених обрахунків та виконати огляд існуючих зразків їх реалізації;
2. Розглянути особливості застосування запропонованої реалізації 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 принципів побудови систем розподілених обрахунків та досліджено можливість їх використання для обрахунку інформації.
2. Запропоновано алгоритм реалізації системи розподілених обрахунків із застосуванням технології MapReduce.
3. Розглянуто особливості побудови запропонованої системи для мобільних платформ .

**Практичний результат роботи:**

1. Розроблено систему розподілених обрахунків, яка базується на технології MapReduce.

2. Створено мобільний додаток для встановлення на мобільні платформи.

# РОЗДІЛ 1

# Провести аналіз принципів побудови систем розподілених обрахунків та виконати 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].



Рисунок 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 забезпечення* відноситься власне до такого класу ПЗ. Воно знаходиться якби посередині між різними частинами розподілених компонент системи.

## Види розподілених архітектур

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:

* Модель репозиторія;
* Модель клієнт/сервер;
* Модель абстрактної машини;



## Модель репозиторія MapReduce

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.



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.

## Модель 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.

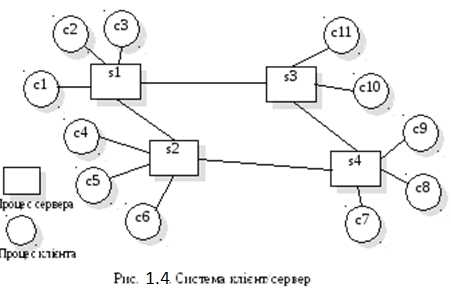
Клієнти повинні знати імена доступних серверів та сервісів, які вони на­дають. В той же час серверам непотрібно знати ні імена клієнтів, ні їх кількість. Клієнти отримують доступ до сервісів, які надає сервер, за допомогою віддале­ного виклику процедур[10].

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.

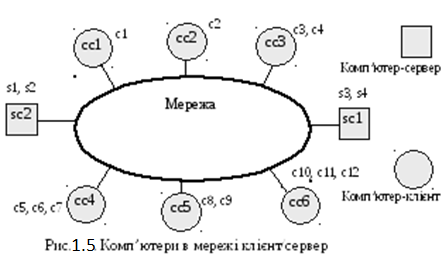
Підхід клієнт/сервер можна використовувати при реалізації систем, що базуються на репозиторії, який підтримується як сервер системи. Підсистеми, які мають доступ до репозиторію, є клієнтами. Але звичайно кожна підсистема керує власними даними. Під час роботи сервери і клієнти обмінюються даними, але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

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

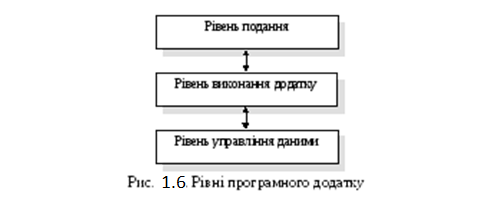
В архітектурі клієнт/сервер програмні додатки моделюються як набір се­рвісів, що надаються серверами, і множина клієнтів, які використовують дані сервіси. Клієнти повинні знати про доступні сервери, хоча і не можуть мати уя­в­лення про існування інших клієнтів. Як видно з рис. 1.4, на якій зображено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



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. На них запускаються клієнтські і серверні процеси, зображені на рис. 1.5. В загаль­ному випадку, говорячи про клієнтів і сервери , маємо на увазі скоріше логічні процеси, ніж фізичні машини, на яких виконуються процеси.

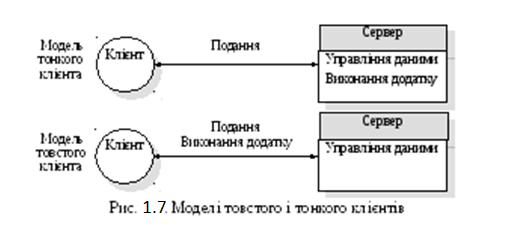


Архітектура клієнт/сервер повинна відображати логічну структуру прог­рамного додатку, який проектується .



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.

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.



**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-броузер 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.

## Модель 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.



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

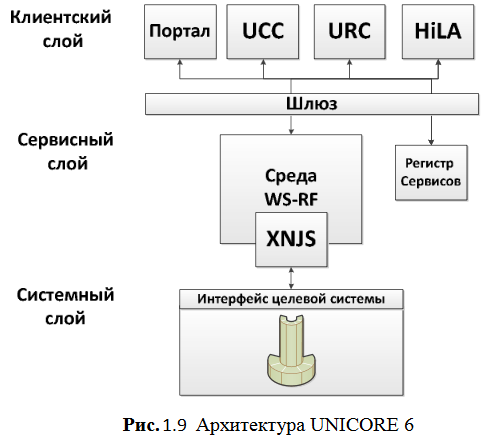
## Система UNICORE

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 (см. 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.

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.

## Модель репозиторія

**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.

**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.

Рисунок 1.20 Екосистема Hadoop

**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. 



Рисунок 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.



Рисунок 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

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.

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.

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

# Запропонувати алгоритм реалізації системи розподілених обрахунків 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):

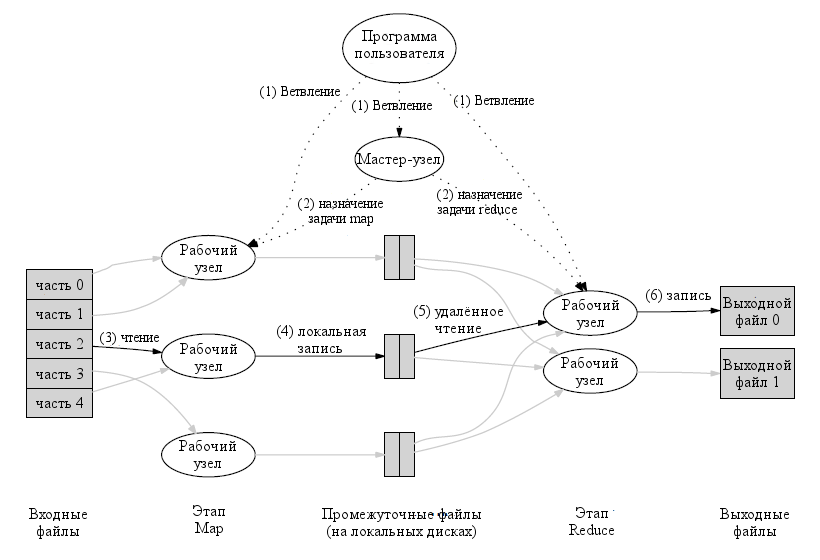


Рисунок 2.1 Детальний огляд роботи реалізації MapReduce

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 таких алгоритмів

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 таких алгоритмів

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

Парадигма MapReduce задовольняє основні вимоги представлені для гетерогенної системи з перемінним числом вузлів та змінним станом мережі.

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.

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.

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

# 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. 3.1.



Рисунок 3.1 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.

На рис.3.2 приведено приклад виконання розробленого алгоритму для пошуку великих літер в словах (текстах).

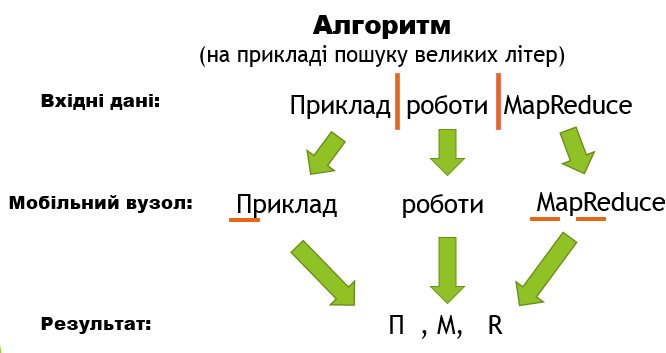


Рисунок 3.2 Приклад виконання алгоритму для пошуку великих літер

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

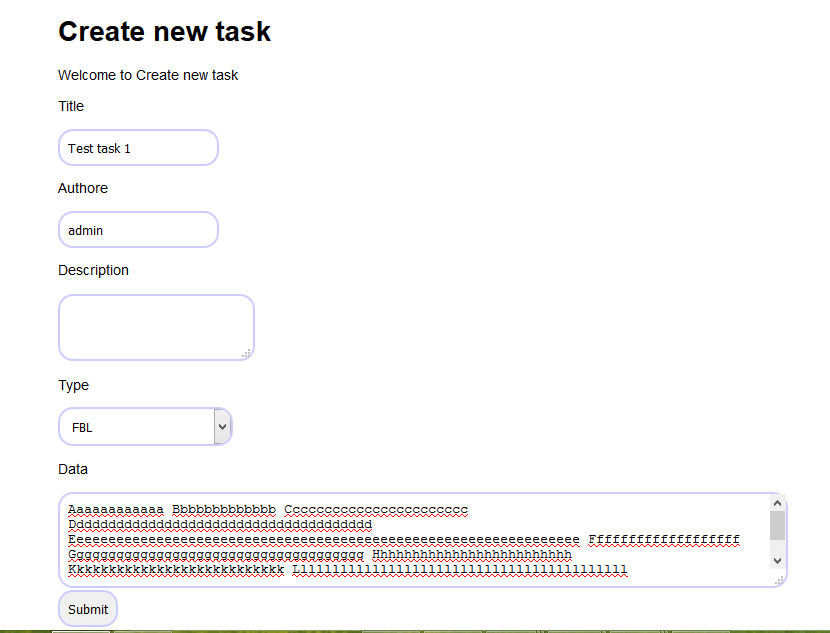


Рисунок 3.3 Сторінка створення завдання

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.3.4 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.



Рисунок 3.4 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.3.5 приведено приклади такого додатку на мобільний пристрій де є список завдань та вкладка із переглядом статусу виконання теперішнього завдання.

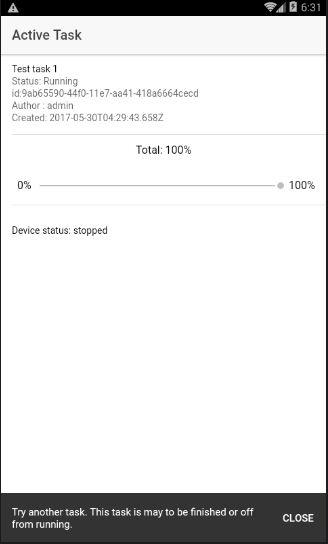


Рисунок 3.5 Приклад додатку на мобільному пристрої

## Висновки

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

Проведено детальний аналіз принципів побудови систем розподілених обрахунків та проведений огляд існуючих прикладів реалізацій даних систем у світі.

Також 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.

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.

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 / I. Zhukov // Aviation – 2010. – vol.16, Готовский Ю. В. Цветовая светотерапия. / Ю. В. Готовский, А. П. Вышеславцев, Л. Б. Косарева, Ю. Ф. Перов, М. М. Шрайбман – М. : ИМЕДИС, 2001. – 432 с
2. Іванкевич О. В., Лукашенко В. В. Засоби керування потоками даних у розподілених обчислювальних системах / О. В. Іванкевич, В. В. Лукашенко// Проблеми інформатизації та управління: Зб. наук. пр.–К.: Вид-во нац. авіац. ун-ту «НАУ-друк», 2010.–Вип. 3(31).–С. 65-69.
3. Мадяр А. Й. Инновационные методы исследований в психофизиологии цветового восприятия : методическое пособие / А. Й. Мадяр, Е. В. Моисеенко, Е. Э. Ковалевская – М. : 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 программирование / Дж. Джарратано — М.: 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., Воеводин Вл.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);

})

}

})

}

}

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)});

} })}