**Національний технічний університет України**

**«Київський політехнічний інститут**

**ім. 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-го 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»

(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 ,

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

затверджені наказом по університету від «07» квітня 2017 р. №1290-с

2. Термін подання студентом роботи 14 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

3. Зміст роботи

1. Провести аналіз вже створених систем розподілених обрахунків для персональних комп’ютерів.
2. Запропонувати модель управління завданнями для розподіленого обрахування.
3. Розглянути особливості застосування запропонованої моделі для мобільних пристроїв.

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

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

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 | Аналіз підходів до побудови гетерогенних систем | 17.01.17 – 01.02.17 |  |
| 6 | Підготовка тезисів (ПТ-17) | 01.02.17 – 01.03.17 |  |
| 7 | Створення прототипу розподіленої системи, опис компонентів | 01.03.17 – 01.04.17 |  |
| 8 | Розробка алгоритму балансування задач між мобільними пристроями | 01.11.16 – 01.03.17 |  |
| 9 | Розробка алгоритму визначення закінчення роботи над завданням | 01.03.17 – 26.03.17 |  |
| 10 | Розробка розподіленої системи | 01.11.16 – 01.05.17 |  |
| 11 | Підготовка тексту диплому | 15.04.17 – 16.06.17 |  |

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

(підпис) (ініціали, прізвище)

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

(підпис) (ініціали, прізвище)

**РЕФЕРАТ**

Робота містить 53 сторінки, 27 рисунків та 2 таблиць. 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 53 pages, 27 figures and 2 tables, 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](#_Toc454016370)

[РОЗДІЛ 1 Аналіз методів оцінки стану людини за психологічними показниками 13](#_Toc454016371)

[1.1 Методи оцінки психофізіологічного стану людини 13](#_Toc454016372)

[1.1.1 Метод кольорової преференції 15](#_Toc454016373)

[1.1.2 Модифікований метод Дембо - Рубінштейн 16](#_Toc454016374)

[1.1.3 Метод комплексного дослідження психофізіологічних характеристик 17](#_Toc454016375)

[1.2 Проблематика сучасних досліджень стану людини 18](#_Toc454016376)

[1.3 Автоматизовані методи оцінки психофізіологічного стану людини 18](#_Toc454016377)

[1.3.1 Використання штучних нейронних мереж 18](#_Toc454016378)

[1.3.2 Використання експертних систем 20](#_Toc454016379)

[Висновки 22](#_Toc454016380)

[РОЗДІЛ 2 Інтелектуальна технологія оцінки психофізіологічного стану людини 24](#_Toc454016381)

[2.1 Вимоги до технології 24](#_Toc454016382)

[2.2 Структура інтелектуальної технології 25](#_Toc454016383)

[2.2.1 Середовище БЗ 26](#_Toc454016384)

[2.2.3 Підсистема прийняття рішень 27](#_Toc454016385)

[2.2.4 Підсистема вводу/виведення 28](#_Toc454016386)

[2.2.5 Підсистема обґрунтування 28](#_Toc454016387)

[Висновки 29](#_Toc454016388)

[РОЗДІЛ 3 Алгоритм визначення кольорової преференції 30](#_Toc454016389)

[Висновки 32](#_Toc454016390)

[РОЗДІЛ 4 Програмна реалізація інформаційної системи оцінки людини 33](#_Toc454016391)

[4.1 Постановка завдання та вимоги до системи 33](#_Toc454016392)

[4.2 Діаграма класів системи та їх опис 33](#_Toc454016393)

[4.3 Діаграма сутностей бази даних та їх опис 38](#_Toc454016394)

[4.4 Структура розробленої інформаційної системи 39](#_Toc454016395)

[4.5 Практичне застосування розробленої системи 41](#_Toc454016396)

[Висновки 48](#_Toc454016397)

[ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 49](#_Toc454016398)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 50](#_Toc454016399)

[Додаток А Блок-схема методу визначення кольорової преференції 52](#_Toc454016400)

**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. Дослідити алгоритми роботи та запропонувати технологію реалізації системи.
3. Розглянути особливості застосування запропонованої реалізації для мобільної платформи.

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

5. Створити розподілену систему обрахунків з використанням розроблених додатків та серверу налаштованих в одній мережі.

**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.

# 1.1 Розподілені обчислення. Мета та цілі їх застосування

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

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

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

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



Рисунок 1

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:

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. *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.

Існують такі види розподілених архітектур:

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

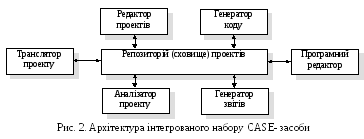
# Модель репозиторія

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

* 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.

На рис. 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:

* 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», яка ініціює підсистеми тоді, коли конкретні дані стають доступ­ними. Такий підхід можливо використовувати до систем, в яких форма даних добре структурована.

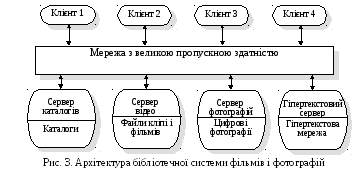
# Модель клієнт/сервер

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

Модель містить три основних компоненти:

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.

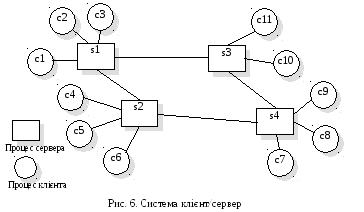


Приклад системи, організованої за типом моделі клієнт/сервер, зображе­но на рис. 3. Це багатокористувацька гіпертекстова система, призначена для підтримки бібліотеки відеофільмів і фотографій. В ній міститься декілька серверів, які розміщують різні типи медіа файлів і керують ними. Відео файли необхідно 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. Тут клієнтська програма є просто інтегрованим інтерфейсом користувача.

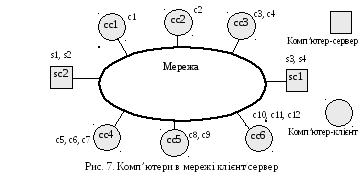
Підхід клієнт/сервер можна використовувати при реалізації систем, що базуються на репозиторії, який підтримується як сервер системи. Підсистеми, які мають доступ до репозиторію, є клієнтами. Але звичайно кожна підсистема керує власними даними. Під час роботи сервери і клієнти обмінюються даними, але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

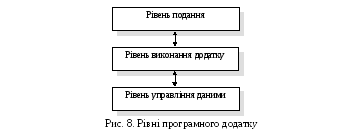
В архітектурі клієнт/сервер програмні додатки моделюються як набір се­рвісів, що надаються серверами, і множина клієнтів, які використовують дані сервіси. Клієнти повинні знати про доступні сервери, хоча і не можуть мати уя­в­лення про існування інших клієнтів. Як видно з рис. 6., на якій зображено мо­дель 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. 7. зображено фізичну архітек­туру системи, що складається із шести клієнтських машин і двох серверів. На них запускаються клієнтські і серверні процеси, зображені на рис. 6. В загаль­ному випадку, говорячи про клієнтів і сервери , маємо на увазі скоріше логічні процеси, ніж фізичні машини, на яких виконуються процеси.

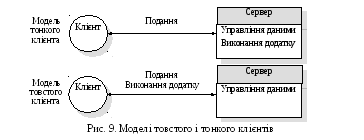


Архітектура клієнт/сервер повинна відображати логічну структуру прог­рамного додатку, який проектується .



На рис. 8 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. 9.



**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.

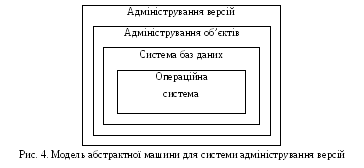
Модель товстого клієнта використовує обчислювальну потужність ло­кальних машин: і рівень виконання додатку, і рівень подання розміщуються на клієнтський сервер. Сервер в даному випадку, за сутністю, є сервером транзак­цій, який керує всіма трансакціями баз даних. Прикладом архітектури такого типу бути 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. На рис. 4. зображена подібна модель і показано, як за допомогою моделі абстрактної ма­шини можемо подати систему адміністрування версій.

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.



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.

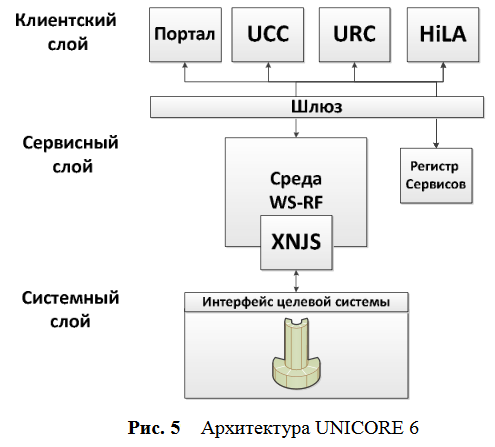
# Приклади реалізацій розподілених систем

# 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. 5). 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) 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.

[https://www.unicore.eu/documentation/architecture/] 14/05 Карла Каппа The architecture of UNICORE

# 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.

**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 виглядає стабільніша, ніж CDH.

Рисунок 7 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 8 Cинтаксис Hadoop під управлінням MapR

Класична конфігурація кластера Hadoop 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.

[http://it-ua.info/news/2014/10/16/hadoop-scho-de-navscho.html ] 10/05

# 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.8. 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 8 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. Приведено визначення гейміфікації.
2. Окреслено особливості гейміфікації, її підходи, властивості .
3. Здійснено огляд декількох за стосунків освітнього та дослідницького характеру.

# РОЗДІЛ 2

# Дослідити можливість їх застосування в навчальному процесі. Запропонувати модель навчального процесу із застосуванням принципів гейміфікації.

# 2.1 Нова освітня модель в контексті навчального процесу

Сучасний освітній процес в навчальних закладах включає наявність декількох видів занять таких як лекції, практичні та лабораторні роботи. На самостійну роботу роботу студента відводиться велика кількість часу. У тому числі і час для підготовки до занять. Зазвичай підготовка до лекцій являє собою повторення вже пройденого матеріалу або ознайомлення з новими знаннями і літературою такою як підручники, методичні посібники, конспекти лекцій. Підготовка до практичних робіт являє собою рішення прикладних практичних задач та виконання інших завдань. Підготовка до лабораторних робіт зазвичай зводиться до повторення теоретичного матеріалу. Під час лабораторних занять студентам пропонується отримати навички роботи з апаратурою і різними пристроями, установками або програмами-симуляторами на ПК. Цей досвід розширює світогляд студентів і дає можливість одержати навички роботи та побачити варіанти реального застосування знань, одержуваних на лекційних і практичних заняттях.

????. В данній роботі пропонується застосування в навчальному процесі навчальної моделі з застосуванням гейміфікації. Існує ряд переваг моделі з використанням гейміфікації перед подібною моделлю без використання гейміфікації (Рис 2.1.1 )

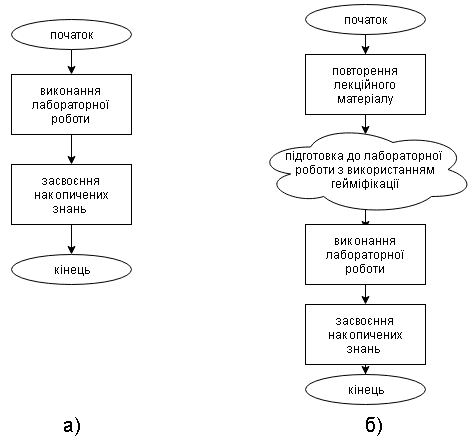


Рис. 2.1.1 Моделі підготовки до лабораторних занять а) з використанням гейміфікації б) без

. Також наявні переваги використання в навчальній моделі беспосередньо застосунка, ці переваги будуть висвітлені дещо пізніше. Стосовно переваг освітньої моделі на рисунку 2.1.2 представлений блок що включає ознайомлення з метою та змістом лабораторної роботи спираючись на методичний матеріал. Після нього слідує блок виконання завдань в програмному застосунку, що ознайомлять студента з виглядом лабораторної установки та її складовими частинами. Це дає змогу студентові напередодні виконання лабораторної роботи отримати короткі відомості про роботу та на момент безпосереднього виконання почувати себе впевнено в більш «знайомих» умовах при відпрацюванні роботи «на техніці». При відпрацюванні лабораторних робіт курсу технічна електродинаміка та поширення радіохвиль важливим чинником є знання студентом лабораторної установки та призначення окремих її «вузлів», оскільки для отримання вимірів при проведенні лабораторної роботи студентові необхідно заміняти

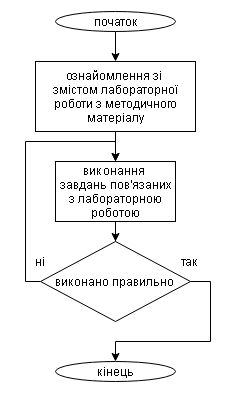


Рис. 2.1.2 Процес підготовки до лабораторної роботи.

або від’єднувати окремі частини установки. Ці маніпуляції дають змогу отримати різні результати вимірювань. В подальшому, користуючись цими результатами вимірювань студенти можуть порівняти їх, підтвердити певні відомі закономірності, або оцінити правильність отриманих вимірів.

Також необхідно звернути увагу, що при використанні данної освітньої моделі студент декілька разів звертається і до методичного матеріалу, як при підготовці до роботи з додатком так і під час виконання лабораторної роботи. Так само двічі студент проходить виконання самої роботи при роботі з додатком та в ході лабораторної роботи. Тобто збільшується кількість повторень котра позитивно впливає на якість запам’ятування і засвоєння знань.

Автор відомої кривої забування професор Еббінгауз завдяки своїм експериментам отримав наступний досвід. Він давав для запам'ятовування своїм студентам довгий список безглуздих звукосполучень типу «дейюкс», «кволі» і подібні до них. В процесі експерименту він встановив, що студенти запам'ятовували ту ж кількість звукосполучень за 38 повторень, вивчаючи їх з інтервалами протягом трьох днів, що і за 68 безперервних повторень протягом одного дня.[ <http://www.brainhack.me/2015/08/3-memory-laws/> 240517] Еббінгауз зрозумів, що в певний момент люди досягають плато - точки, в якій додаткові зусилля майже не призводять до додаткових результатів. Він почав експериментувати з часовими інтервалами між изучениями інформації і згодом довів, що ефективність може бути підвищена на 500 відсотків за допомогою впровадження найпростіших технологій. Його відкриття плато запам'ятовування, а також дослідження про те, як цього ефекту уникнути, привели до розвитку методів навчання, які допомогли прискорити процес запам'ятовування і лягли в основу однієї з найефективніших технік запам'ятовування в світі.

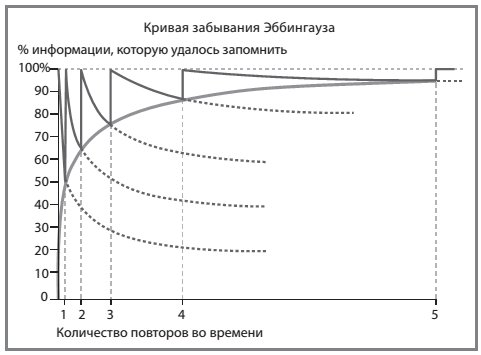


Рис. 2.1.3 Крива забування Ебінгауса

Згідно до найефективнішого методу навчання Ебінгаус визначає наступний алгоритм повторень.

* Перше повторення — відразу після закінчення читання.
* Друге повторення — через 20 хвилин після першого повторення.
* Третє повторення — через 24 години після другого.

Можна помітити, що запропонована модель навчання за графіком повторень дуже наближується до встановлених Ебінгаусом висновків, що свідчить про її високу ефективність.

# Відмінності між роботою(освіта) та грою

[presentation ENTERPRISE GAMIFICATION Engaging people by letting them have fun Mario Herger]

У своїй доповіді: «Зацікавлення людей, дозволяючи їм отримувати задоволення» - Маріо Хергер, доповідаючи про гейміфікацію, приводить порівняльну характеристику гри та роботи(Табл. 2.1). У таблиці в стовпці «робота» представлено сукупні характеристики, що є справедливими для більшості професій та багатьох видів періодичної монотонної діяльності. Оскільки освіта або точніше навчання — важка розумова діяльність, праця, в такому випадку справедливо говорити про тотожність роботи та навчання (провести аналогію з роботою). В буквальному сенсі слово «гейміфікація» можна зрозуміти як процес перетворення чогось в гру. В данній роботі мова йде про перетворення певного навчального матеріалу в гру. З таблиці можна побачити різку перевагу гри перед навчанням у чистому вигляді. В дещо меншій мірі гейміфікація дозволяє досягти цієї різниці, але відмінність все одно відчутна.

Табл. 2.1 Порівняльна характеристика між грою та роботою.

|  |  |  |
| --- | --- | --- |
|  | Гра | Робота |
| Завдання | Повторюється, веселе | Повторюється, одноманітне |
| Зворотній звязок | Постійно | Рідко |
| Мета | Зрозуміла | суперечливий, розпливчастий |
| Шлях до досконалості | Зрозумілий | Не зрозумілий |
| Правила | Зрозумілі | Не зрозумілі |
| Інформація | Потрібна кількість в потрібний час | Забагато і недостатньо |
| Помилки | Очікується, підбадьорений, ефектний, хвалитися про це | заборонено, карали, не говорити про це |
| Статус користувачів | прозорий, своєчасний | Прихований |
| Співробітництво | Присутнє | Присутнє |
| Швидкість, ризик | Вісокі | Низькі |
| Автономність | Висока | Середня-низька |
| Інформативність | Висока | Іноді |
| Перешкоди | Залежно від мети | Випадкові |

В те дни, когда я даже думать не хотел о рисовании, я исполь- зовал трюк, который узнал от Рика Рубина, который состоит в том, чтобы согласиться на минимальный объем работы. В моем случае я говорил себе, что могу прекратить рисовать, как только нарисую одну линию

[Mihaly Csikszentmihalyi: *Flow: The Psychology of Optimal Experience*, 1991 *Beyond Boredom and Anxiety: Experiencing Flow in Work and Play*, 1975]

Зворотній зв’язок, між користувачем і застосуванням є важливим засобом гейміфікації, він дозволяє найефективнішим шляхом тримати користувача в стані найбільш підходящому для ефективного освітнього процесу. Застосунок з використанням гейміфікації певним чином динамічно пристосовується до поведінки користувача. Наприклад за стосунок може змінювати складність завдань. Понижувати або збільшувати рівень складності завдань відповідно до поведінки користувача. Такий підхід може бути реалізований наприклад наступним чином. Якщо користувач починає робити помилки частіше, застосування динамічно знижує рівень складності, щоб користувач отримував адекватні за складністю завдання відповідно до свого рівня підготовки. Такий підхід є дуже ефективним, адже дозволяє підтримувати інтерес користувача не перевантажуючи його надто складними завданнями. Таким чином будь-який користувач знаходиться у стані сприятливому для навчання. Подібний процес описав психолог Михай Чиксентмихайи. Свою теорію він назвав теорією оптимального переживання. Теорія оптимального переживання ґрунтується на понятті потоку-стану. Потік-стану означає повне поглинання діяльністю настільки, що інші фактори вже не чинять істотного впливу. Описуючи величину отримуваного задоволення від процесу діяльності вчений зазначає, що задоволення настільки велике, що люди будуть готові навіть платити гроші щоб займатися цією діяльністю.

Також Михай вводить поняття потокової діяльності, зміст якої полягає в набутті радості. Прикладами таких занять є ігри, мистецтво, спорт, а також різні активності. Потокової діяльності змінюють особистість, роблячи її більш складною, завдяки її розвитку. Розвиток тісно пов’язаний зі змістом потокової діяльності (рис. 1).

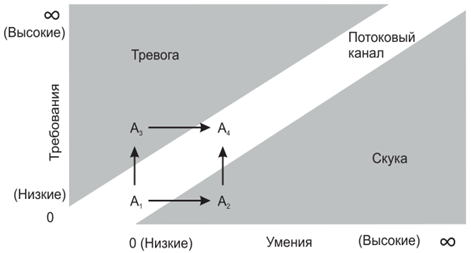
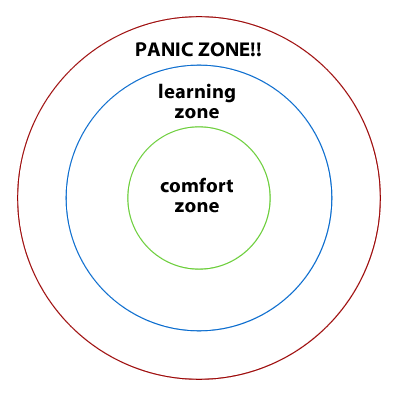


Рис. 2.4 Потоковий канал

Припустимо, що наведений нижче графік описує якусь діяльність, наприклад, гру в теніс. На його осях відкладені рівні розвитку умінь і складності завдань. буквою А позначений Алекс - хлопчик, який вчиться грати в теніс. На діаграмі він зображений в чотирьох тимчасових точках. На момент початку тренувань у нього практично відсутні навички гри в теніс ( точка А1 ). Від нього вимагається просто влучити по м'ячу. Це не складно, проте Алекс тренується з величезним задоволенням. Адже рівень складності завдання як раз відповідає його елементарних навичок. Тому на цьому рівні він, ймовірно, буде відчувати відчуття потоку. ( Рис. 2.4 ). Згодом, якщо він продовжить тренуватися, його спортивна форма стане краще. йому буде нудно просто відбивати м'яч через сітку ( точка А2 ). Можливо, він знайде собі більш досвідченого партнера і усвідомить, що існують більш складні завдання, ніж просто бити по м'ячу. У цій точці A3 він відчує занепокоєння через те, що погано справляється з ситуацією. Оскільки ні нудьга, ні тривога не відносяться до приємних станів, Алекс знову хоче випробувати стан потоку. З малюнка видно, що, якщо він нудьгує ( точка А2 ) і прагне знову потрапити в потік, у нього є тільки один вихід - підвищити складність виконуваних завдань. Поставивши перед собою нову, більш складну мету, що відповідає рівню його навичок, — наприклад, перемогти сильнішого супротивника, — Алекс знову опиниться в потоці ( точка А4 ). Якщо він відчуває занепокоєння ( точка A3 ), для повернення до стану потоку йому необхідно далі вдосконалювати свої навички. Стан в А4 можна вважати більш «просунутим» у порівнянні з А1, тому що він передбачає більш складні завдання і вимагає від гравця більшої майстерності. Однак ситуація А4, незважаючи на складність і принесену нею радість, також не є стабільною. якщо Алекс продовжить грати, він зіткнеться або з нудьгою через які втратили новизну завдань, або з тривогою і фрустрацією через усвідомлення своїх відносно слабких можливостей. прагнення отримувати задоволення знову штовхне його в потік, але вже на рівні складності, що перевищує А4. Така динаміка пояснює, чому потік сприяє особистісному розвитку. Людина не може довго отримувати насолоду, роблячи одне і те ж. Щоб уникнути нудьги, він повинен ставити собі нові цілі і, досягаючи їх, удосконалювати свої навички.



Comfort Zone:

Зона комфорту дійсно відмінне місце, щоб бути в. Це зона, де не приймають ніяких проблем. Це місце, де багато хто з нас живуть просто тому, що це зручно, безпечно, не займають думка або розум, і ніяких турбот не беруть участь. Отже, не інтелектуальні зусилля не виплачений, так як це зона нормального життя, яка не включає в себе ніяких ризиків взагалі. Звичайно ж, це не обов'язково означає, що ми щасливі в той час як ми знаходимося в Comfort Zone, тому що ми могли б бути в болю поки ми не вживати ніяких дій, щоб вийти з цієї ситуації, а ми віддатися самовдоволення, і наше життя перетворити в рутину, людина, яку ми сьогодні та сама людина, ми були вчора. У зв'язку з цим, життя не захоплюючий, і ми ні рости, ні розвиватися; навіть якби ми зробили, це нереально зростання або сприяє відсталості - так як недолік розвитку самої відсталість.

Паніка зона:

Зона паніки характеризуються сильною тривогою і напругою. Людина в цій зоні постійно перебуває в стані як безмисленной активності і стресу. Навіть коли нічого не відбувається, він знаходиться в постійній тривозі і стресі, тому що він знайде щось панікувати і бути стурбовані. Він може навіть створювати речі, щоб турбуватися про.

В основному, всі питання, великі і важливі, і вимагають уваги, яке супроводжується певною кількістю страху або тривоги, що перешкоджає нашій здатності впоратися з ситуацією належним чином. Життя сповнене проблем і труднощів. Ми часто бажаємо на краще, але ми насправді не робити жодних кроків, щоб зробити його краще. Ми знаходимося в стані постійної реакції на речі навколо нас. Як результат - такий же, як вона знаходиться в зоні комфорту - ми не будемо розвивати.

Навчання зони:

Зона навчання характеризується різним способом обробки все. Для кожної ситуації є чому повчитися, чи було це добре чи погано, правильно чи неправильно, легко чи важко, він завжди буде виглядати як досвід. Для того, щоб бути в навчальній зоні, щоб мати можливість сказати: «незалежно від того, що відбувається, я не буду користуватися і вчитися». У цій зоні, проблеми розглядаються як спосіб рости, помилки розглядаються як спосіб отримати досвід, і невдача розглядається як можливість створення персонажа. Людина, ми сьогодні були мудрішими, ніж людина, ми були вчора. Життя повне можливостей і сюрпризів, і наше хвилювання на-віч день слід збільшити день за днем.

Перейти до Вивченню Зоні:

Люди часто виявляються або в комфортній зоні або панічної зоні великою частиною свого життя. Там немає ніякого зростання, і завжди є тривога і стрес. Життя перетворюється в фатальну рутину, без зору або цілі, щоб дивитися вперед. Причина, чому людина повинна двічі подумати, як перейти в стан або зони навчання. Для цього потрібно лише воля, рішучість і дії.

Це зовсім не означає, що ми не знаходимо себе, перебуваючи в будь-якому панічної зоні або зоні комфорту; це просто означає, що, коли ми опиняємося в них, ми повинні бути в змозі прийняти рішення вийти з них і крок в навчальній зоні, і що ми будемо реагувати і реагувати на ситуацію по-іншому. Перебуваючи в зоні навчання змінить спосіб, яким ми живемо на краще. Саме тому ми повинні вирішити, щоб бути в навчальній зоні тепер.

# РОЗДІЛ 3

# Розглянути особливості застосування запропонованої навчальної моделі для курсу технічної електродинаміки.

# РОЗДІЛ 4 Розробити макети лабораторних робіт з курсу технічної електродинаміки, які базуються на принципах гейміфікації.

## Созданная компьютерная программа позволяет ознакомиться с устройством и порядком сборки лабораторной установки. В этом приложении создано макеты лабораторных установок предмета техническая электродинамика и распространение радиоволн. Игровая логика позволяет получить интерактивное взаимодействие с установками. При проведении лабораторных работ важной частью является порядок сборки установки. При снятии показаний с измерительных устройств установки в ходе проведения лабораторной работы производится замена некоторых её составляющих. Поочередная установка различных сменных деталей позволяет получить разные выходные данные. С помощью отличающихся данных студенты могут пронаблюдать физические зависимости, проверить практически известные теоретические знания. В приложении предоставляется возможность сборки установки.

## Для создания приложения была выбрана игровая платформа разработки Unity 3D так как она предоставляет возможности для создания интерактивной программы с использованием трехмерных моделей. Их использование позволяет сделать процесс работы в приложении более наглядным. Трехмерные модели составных частей установки были смоделированы в пакете создания трехмерной графики Blender 3D. Оба средства разработки являются бесплатными программными продуктами.

## Висновки

1. Сформовано необхідні завдання та вимоги до розробки інформаційної системи.
2. Запропоновано та впроваджено програмну реалізацію інформаційної системи оцінки психофізіологічного стану людини.
3. Розроблено додаткові модулі та підсистеми для поглибленого аналізу та представлення інформації в наглядному вигляді.
4. Впроваджено для використання в НАНЦ України.

# ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1. Проведено детальний аналіз методів оцінки стану людини за психологічними показниками.
2. Визначена необхідність розробки інтелектуальної технології оцінки психофізіологічного стану людини, який може використовувати існуючі методи за відсутності експерта.
3. Поставлене завдання розробки інтелектуальної технології. Визначено критерії та правила, за якими отримана оцінка вважається достовірною.
4. Запропоновано інтелектуальну технології оцінки психофізіологічного стану людини.
5. В рамках системи оцінки психофізіологічного стану людини, запропоновано алгоритм визначення кольорової преференції.
6. Розроблено апаратно-програмну інформаційну технологію оцінки психофізіологічного стану людини з можливостями налаштування та перегляду параметрів, введення інформації, її зберігання та використання, представлення інформації в декількох можливих варіантах.

# 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

# Додаток А Блок-схема методу визначення кольорової преференції



