**АНОТАЦІЯ**

У ході магістерського дослідження була розроблена розподілена система кластеризації даних, яка базована на алгоритмі k-means та використовує веб-браузер у якості обчислювального вузла.

Основною новизною даного проекту є новий підхід до розподілених обчислень, а саме: легке долучення нових клієнтів, прозорість процесу обчислень та зручний інтерфейс, який не потребує додаткових дій з боку користувача.

Було створено прототип системи, який реалізує наступний функціонал:

* Грід-система для розподіленої кластеризації даних алгоритмом k-means. Система є динамічною (не залежить від кількості веб-клієнтів) та гнучкою (можна змінити як вхідні дані, так і алгоритм обробки);
* Централізований ресурс у вигляді веб-додатку ASP.NET MVC, який надає основну інформацію про проект, статистику обчислень та результати кластеризації;
* Веб-клієнт у вигляді javascript-коду, який завантажується у браузер користувача та забезпечує процес розподілених обчислень.

Створений прототип після деякої адаптації може використовуватись для таких цілей:

* Добровільні грід-системи – для волонтерських обчислень певної ресурсоємкої задачі з використанням обчислювальних ресурсів користувача (за принципом систем Boinc);
* Комерційні грід-системи – для виділення обчислювальних ресурсів при необхідності із взаємною вигодою (користувачам можуть нараховуватися певні бонуси, відсутність реклами, тощо).

Обсяг роботи складає 110 сторінок.

**ABSTRACT**

The paper presents a distributed computing system for data clusterization, which is based on k-means algorithm and uses a web browser as a processing node.

The main novelty of this project is a new approach to distributed computing: easy attration for new customers, transparency of computing and user-friendly interface that doesn’t require additional actions from the user.

As the result of the master's work was created a prototype of system that implements next functionality:

• Grid system for distributed data clusterization with algorithm k-means. The system is dynamic (not depends on the number of web clients) and flexible (can be changed input data and processing algorithm);

• Web application ASP.NET MVC for information about the project, statistics and calculation results;

• Web client as javascript code, that is loaded to the client's browser and provides a distributed computing.

Created prototype after some modification may be used for the following purposes:

• Voluntary grid system - for volunteer computing of some hard problem using free computer resources (such as Boinc systems);

• Commercial grid systems - to provide computing resources when required (as profit users can obtain some bonuses, no ads, etc.).

The amount of work is 110 pages.

**ЗМІСТ**

АНОТАЦІЯ 3

ABSTRACT 4

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ І СПЕЦІАЛЬНИХ ТЕРМІНІВ 8

ВСТУП 10

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТОЇ ОБЛАСТІ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ 13

1.1. Визначення розподілених обчислень, їх характеристики та історія 13

1.2. Приклади використання розподілених обчислень 16

1.3. Алгоритми та моделі розподілених обчислень 17

1.4. Волонтерські розподілені обчислення 19

1.5. Веб-браузер у розподілених обчисленнях 22

РОЗДІЛ 2. ПІДХОДИ ДО СТВОРЕННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ 25

2.1. Архітектура системи 25

2.2. Алгоритми кластеризації 25

2.2.1. Визначення понять кластерного аналізу 25

2.2.2. Алгоритм k-means 28

2.3. Вхідні дані системи 33

2.4. Вихідні дані системи 37

2.5. Критерії оцінки результатів 38

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОЇ СИСТЕМИ 39

3.1. Специфікація вимог до розподіленої системи 39

3.1.1. Вступ 39

3.1.2. Загальний опис 40

3.1.3. Характеристики системи 41

3.1.4. Вимоги зовнішніх інтерфейсів 45

3.1.5. Інші нефункційні вимоги 47

3.2. Технології програмної реалізації розподіленої системи 48

3.3. Проектування розподіленої системи 51

3.3.1. UML діаграми 51

3.3.2. Проектування бази даних 51

3.3.3. Проектування бізнес-логіки 52

3.3.4. Логіка розподілених обчислень 54

3.4. План забезпечення якості розподіленої системи 58

3.4.1. Мета 58

3.4.2. Управління проектом розподіленої системи 58

3.4.2.1. Організація проведення робіт по розробці розподіленої системи 58

3.4.2.2. Завдання учасників проекту розробки розподіленої системи 59

3.4.2.3. Ролі та зобов’язання кожного з учасників проектування розподіленої системи 59

3.4.2.4. Оцінка ресурсів для забезпечення якості розподіленої системи 59

3.4.3. Документація 59

3.4.3.1. Мета 59

3.4.3.2. Мінімальні вимоги до документації 60

3.4.4. Стандарти, практики, конвенції і метрики 60

3.4.5. Тестування 60

РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ РОЗПОДІЛЕНОЇ СИСТЕМИ 61

4.1. Результати кластеризації даних 61

4.2. Статистика роботи розподіленої системи 64

4.3. Висновки до результатів роботи розподіленої системи 68

РОЗДІЛ 5. ЕКОНОМІЧНА ЧАСТИНА 70

5.1. Економічна характеристика проектного рішення 70

5.2. Розрахунок витрат на розроблення та впровадження проектного рішення 71

5.3. Визначення комплексного показника якості 77

5.4. Визначення експлуатаційних втрат 80

5.5. Розрахунок ціни споживання проектного рішення 83

5.6. Визначення показників економічної ефективності 84

5.7. Висновки економічної частини 86

ВИСНОВКИ 88

ПЕРЕЛІК ПОСИЛАНЬ 90

ДОДАТКИ 93

**ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ І СПЕЦІАЛЬНИХ ТЕРМІНІВ**

Браузер (веб-оглядач) – програмне забезпечення для комп’ютера або іншого електронного пристрою, що надає користувачеві доступ та взаємодію з ресурсами веб-мережі;

Розподілені обчислення (розподілена обробка даних) - спосіб розв'язання трудомістких обчислювальних задач з використанням двох і більше обчислювальних засобів, об'єднаних в мережу;

Алгоритм k-means (кластеризація методом k-середніх) – найбільш популярний метод неієрархічної кластеризації даних, в якому основна ідея – мінімізація середнього квадратичного відхилення точок кластерів від їх центрів;

Кластеризація (кластерний аналіз) – задача розбиття заданої вибірки об'єктів (ситуацій) на підмножини, що називаються кластерами, так, щоб кожен кластер складався зі схожих об'єктів, а об'єкти різних кластерів істотно відрізнялися;

WEB 2.0 – поняття, яке об’єднює ряд новітніх технологій та послуг мережі інтернет (зокрема HTML 5) та використання цих послуг не лише задля отримання інформації, а як динамічну інтерактивну взаємодію між клієнтом та сервером;

Javascript – клієнтська скриптова мова, яка повсюдно використовується у браузерах для придання веб-сторінкам інтерактивності та динамічності;

HTML 5 – версія гіпертекстової мови розмітки та набір технологій, які забезпечують нові можливості по семантичній розмітці сторінки, підтримку мультимедіа об’єктів, векторної графіки, збереження даних на стороні клієнта тощо;

Грід - узгоджене, відкрите і стандартизоване середовище, яке забезпечує гнучкий, безпечний та скоординований розподіл ресурсів у рамках віртуальної організації;

BOINC - некомерційне кросплатформове ПЗ для організації розподілених систем; використовується для організації добровільних обчислень.

**ВСТУП**

В наш час стрімкого розвитку обчислювальних алгоритмів та необхідності обробки все більших об’ємів даних, розмір яких зростає з кожним роком, виникає різка потреба доступу до потужних обчислювальних ресурсів. Великі комерційні організації та наукові інститути можуть вирішити цю проблему з використанням суперкомп’ютерів та обчислювальних кластерів. Для менших організацій та некомерційного використання такі обчислювальні ресурси є практично недоступними.

Розподілені мережі є альтернативою таких обчислень. Їхня потужність часом наближається до потужності найсучасніших суперкомп’ютерів (наприклад сумарна потужність усіх Boinc-проектів разом взятих становить 9,966 ПФлопс/сек [1], потужність №1 у списку ТОП-500, суперкомп’ютера Tianhe-2, становить 33,86 ПФлопс/сек, при цьому 2-й та 3-й у цьому списку мають відповідно 16,324 та 8,162 ПФлопс/сек [2], що є близьким показником до потужності розподілених мереж).

Розподілені волонтерські обчислення мають значні переваги перед зосередженими – насамперед це їх масштабованість, можливість практично кожному прийняти участь (або створити власний проект), значно менші витрати на обслуговування та підтримку (потрібні кошти лише на розробку ПЗ та підтримку централізованого сервера для синхронізації та збереження даних).

Але, незважаючи на їх переваги, є й чималі недоліки. Їх ефективність значно менша у порівнянні зі зосередженими або паралельними системами (затримки через комунікацію між клієнтом та сервером, неоднорідність клієнтських машин тощо). Основною характеристикою, яка дозволяє компенсувати меншу ефективність, є кількість обчислювальних вузлів.

Якщо при використанні кластерів, а, тим більше, суперкомп’ютерів, стоїть різке обмеження по масштабованості мережі (це як значні кошти, так і пристосованість лише до певного апаратного забезпечення або певної обчислювальної задачі), то для розподілених мереж ця проблема не стоїть. Кожен обчислювальний вузол може розглядатися як некритичний компонент, який при необхідності можна видалити або замінити і при цьому загальна потужність системи не зміниться.

При цьому проблема стоїть лише у кількості цих вузлів та процесі долучення нових користувачів до обчислень. Зазвичай процес долучення виглядає наступним чином [3]:

* Користувач дізнається про розподілені обчислення на певному веб-ресурсі, через рекламу тощо;
* Зацікавлений користувач заходить на сайт обчислень, завантажує клієнтське програмне забезпечення;
* Користувач встановлює додаток на своїй машині, налаштовує різні параметри і лише після цього запускає обчислення.

В процесі обчислень користувач бачить певну інформацію та статистику, для чого саме використовуються ресурси його комп’ютера, але не може бути впевнений, що саме це відбувається в даний момент (що його комп’ютер не використовується для зловмисних цілей, наприклад ddos-атак). Переважна більшість програмного коду клієнтів є закритою.

Для більшості потенційних користувачів не так гостро стоїть питання безпеки, як сам процес долучення до мережі. Користувачі не бачать доцільності в цьому, не мають вільного часу або можливості встановити клієнт.

В даній роботі пропонується наступний підхід до розподілених обчислень:

* Використання браузера в якості обчислювального вузла. Це позбавить необхідності завантажувати окремий клієнт для обчислень, налаштовувати його та запускати;
* Веб-браузер є, можливо, найбільш поширеним додатком, тому це надає доступ до обчислень більш ніж третині населення землі, яка має доступ до інтернету. Сюди також можна включити інші електронні пристрої, які мають підтримку сучасних браузерів, такі як планшети, телефони тощо. Користувач може отримати доступ до системи у будь-якій точці світу зі свого персонального пристрою, що так важливо у «пост-комп’ютерну» епоху переходу на персональні цифрові пристрої;
* Створити систему з відкритим вихідним кодом для забезпечення прозорих обчислень та безпеку обчислювальних ресурсів користувача;
* Веб-орієнтовані обчислення дозволять залучити значно більшу кількість користувачів, що компенсує втрати ефективності через затримки мережі, виконання у інтерпретаторі тощо. Це можна досягнути як введенням певних бонусів (наприклад додаткові бали, відсутність реклами на сайті тощо), так і широкою рекламною кампанією в інтернеті.

В області веб-орієнтованих розподілених обчислень було написано декілька наукових публікацій [4,5], але всі вони надають лише теоретичний матеріал, робочого прототипу системи не існує.

Отже метою даної роботи є створення робочого прототипу розподіленої системи для кластеризації даних з використанням ресурсів веб-клієнтів, отримання практичних результатів та статистики її роботи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТОЇ ОБЛАСТІ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ

* 1. Визначення розподілених обчислень, їх характеристики та історія

Розподілені обчислення – це область комп’ютерних наук, яка вивчає розподілені системи. Розподілена система – це програмна система, у якій компоненти, які знаходяться на мережевих ресурсах, комунікують та координують свої дії методом відправки повідомлень [6]. Компоненти взаємодіють між собою з метою досягнення загальної мети. Є багато альтернатив механізму відправки повідомлень, включаючи RPC-подібні коннектори та черги повідомлень.

Є 3 значущі характеристики розподілених систем:

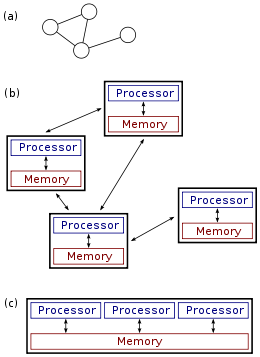
* Паралелізм компонентів
* Відсутність глобальних годинників
* Незалежний вихід з ладу компонентів

Розподілені системи можуть мати загальну мету, таку як вирішення великої обчислювальної задачі. Також кожний вузол може мати свого індивідуального користувача з власними потребами, і вигода з розподіленої мережі у цьому випадку може бути координація використання спільних ресурсів або забезпечення комунікаційних сервісів для користувачів.

Іншими типовими властивостями розподілених мереж є [7]:

* Система повинна забезпечувати відмовостійкість при помилках клієнтів;
* Структура системи (топологія мережі, затримки комунікації, кількість вузлів) є динамічною, система може складатися з різних видів комп’ютерів і мережевих зв’язків, а також система може змінюватись під час виконання розподілених обчислень;
* Кожний вузол має лише обмежене, неповне бачення системи в загальному. Кожний вузол повинен знати лише про свою частину вхідних даних.

Різницю між паралельними та розподіленими системами можна побачити на рис. 1.1.



*Рис.1.1. Порівняння архітектури розподілених (a-b) та паралельних (с) систем*

Частина (а) є схематичним зображенням типової розподіленої системи; зазвичай система представляється як мережева топологія, у якій кожний вузол є обчислювальним пристроєм та кожна лінія, яка поєднює вузли, є мережевим зв’язком. Частина (b) демонструє таку ж розподілену систему у більших деталях: кожний вузол має свою локальну пам’ять, ізольовану від інших вузлів, інформація може бути передана лише через повідомлення від одного вузла до іншого, використовуючи доступні мережеві з’єднання. Частина (с) представляє паралельну систему, у якій кожний процесор має прямий доступ до загальної пам’яті.

Історія розподілених процесів починається з архітектури операційних систем 1960-х років. Першими поширеними розподіленими системами були локальні мережі, такі як Ethernet та ARPANET. Пошта стала найбільш успішним додатком ARPANET і, можливо, найпершим прикладом масштабної розподіленої мережі.

Вивчення розподілених обчислень стало окремою гілкою комп’ютерних наук наприкінці 70-х та на початку 80-х років. Перша конференція в цій галузі, Symposium on Principles of Distributed Computing (PODC), датується 1982 р., і його європейський аналог International Symposium on Distributed Computing (DISC) був вперше проведений у 1985 р.

У 1973 році Джон Шох і Джон Хапп з каліфорнійського науково -дослідного центру Xerox PARC написали програму, яка ночами запускалася в локальну мережу PARC і примушувала робочі комп'ютери виконувати обчислення.

У 1978 році радянський математик Віктор Глушков працював над проблемою макроконвеєрних розподілених обчислень. Він запропонував ряд принципів розподілу роботи між процесорами.

У 1988 році Арьєн Ленстра і Марк Менес написали програму для факторизації довгих чисел. Для прискорення процесу програма могла запускатися на декількох машинах, кожна з яких обробляла свій невеликий фрагмент.

У 1994 році Девідом Джиді була запропонована ідея щодо організації масового проекту розподілених обчислень, який використовує комп'ютери добровольців (так звані Волонтерські обчислення) - SETI@Home.

У січні 1996 року стартував проект GIMPS з пошуку простих чисел Мерсенна, також використовуючи комп'ютери простих користувачів як добровільну обчислювальну мережу.

28 січня 1997 року стартував конкурс RSA Data Security на вирішення завдання злому 56-бітного ключа шифрування інформації RC5 методом простого перебору.

17 травня 1999 стартував SETI@home на базі Grid, а на початку 2002 року завершилася розробка Каліфорнійського Університету в Берклі відкритої платформи BOINC (Berkeley Open Infrastructure for Network Computing), розроблюваної з квітня 2000 року спочатку для SETI@home, але першим на платформі BOINC став проект Predictor@Home, запущений 9 червня 2004 року.

* 1. Приклади використання розподілених обчислень

Причини використання розподілених систем можуть бути наступними:

* Сам характер додатку може вимагати використання мережі зв'язку, яка об’єднює декілька комп'ютерів, наприклад, дані є отримані в одному фізичному місці та необхідні в іншому;
* Є багато випадків, коли використання одного комп'ютера було б в принципі можливим, але використання розподіленої системи виявляється вигіднішим з практичних міркувань. Наприклад, це може бути більш економічно вигідним для отримання бажаного рівня продуктивності за допомогою кластера з декількох недорогих комп'ютерів у порівнянні з одним суперкомп’ютером. Розподілена система може забезпечити більшу надійність, ніж зосереджена система, так як немає єдиної точки відмови. Крім того, розподілена система може бути простішою для розширення та управління, ніж монолітна однопроцесорна система. [7]

Приклади розподілених мереж та програм для розподілених обчислень можуть бути наступні:

* Телекомунікаційні мережі
  + Телефонні та стільникові мережі;
  + Комп’ютерні мережі, такі як інтернет;
  + Бездротові wi-fi мережі;
  + Маршутизатори.
* Мережеві програми
  + World wide web та peer-to-peer мережі;
  + Масові мережеві онлайн ігри;
  + Розподілені бази даних та розподілені системи управління даними;
  + Розподілені інформаційні системи обробки, такі як банківські системи та системи бронювання авіакомпаній.
* Контроль обчислень у реальному часі
  + Повітряні системи контролю;
  + Різноманітні промислові системи контролю.
* Паралельні обчислення
  + Наукові обчислення, включаючи кластерні та грід обчислення, а також численні волонтерські проекти обчислень;
  + Розподілені графічні системи.
  1. Алгоритми та моделі розподілених обчислень

Розподілені алгоритми є різновидом паралельних алгоритмів, але включають окремі частини, які виконуються одночасно на незалежних процесорах і мають обмежену інформацію про те, що виконують інші частини алгоритму. Прикладом може слугувати розподілена версія алгоритму k-means, де на клієнті обчислюються дві незалежні частини алгоритму – обчислення належності векторів певним кластерам (вхідні дані – набір центральних точок кластерів) та обчислення нових центрів кластерів (вхідні дані – набір посилань вектор-кластер).

Однією з найважливіших проблем розподілених алгоритмів є координування роботи частин алгоритму та обробка виключних ситуацій і помилок комунікації між вузлами.

Іншими проблемами розподілених обчислень є:

* Неподільна транзакція – операція, де набір певних змін застосовується як єдиний оператор. Якщо транзакція пройшла успішно, це означає, що всі зміни були проведені. Якщо сталася помилка під час транзакції, то вона відміняється і жодні зміни не будуть внесені. Для вирішення цієї проблеми використовуються алгоритми двохфазної та трьохфазної фіксації;
* Консенсус – алгоритми консенсусу забезпечують прийняття єдиного загального рішення усіма процесами;
* Вибір лідера – вибір певного процесу як організатора деякої задачі, розподіленої на декілька вузлів;
* Взаємне виключення – вимога, щоб жодні два процеси не були одночасно в своїх критичних секціях (гонка процесів);
* Розподілення ресурсів – виділення ресурсів процесу у найбільш ефективний спосіб;
* Реплікація – розподілення інформації між багатьма процесами для покращення надійності, відмовостійкості або доступності.

Велика кількість досліджень фокусується на розумінні асинхронної природи розподілених обчислень:

* Синхронізація – запуск синхронних алгоритмів у асинхронних системах;
* Логічні годинники – механізм визначення хронологічного та причинно-наслідкового зв’язку у розподілених системах;
* Синхронізація годинників – алгоритми забезпечення глобально-послідовного часу подій.

Для розподілених обчислень використовують різні види програмних та апаратних архітектур. На нижчому рівні є необхідним поєднати багато процесорів деяким видом мережі, незважаючи на те, чи мережа розміщена на платі чи складається зі слабо пов’язаних пристроїв та кабелів. На вищому рівні виникає необхідність взаємодії процесів, які запускаються на цих пристроях, за допомогою деякої комунікаційної системи.

Зазвичай розподілені системи використовують одну з наступних архітектур:

* клієнт-сервер – «розумна» клієнтська програма звертається до сервера за даними та задачею, обчислює її і повертає результат на сервер;
* 3-рівнева архітектура – переміщення клієнтської логіки в середній рівень, що дозволяє звільнити клієнта від додаткової логіки (є типовою для веб-додатків);
* n-рівнева архітектура – зазвичай використовується у веб-додатках, коли сервер перенаправляє запити клієнта на інші сторонні сервіси;
* кластерна – набір однотипних кластерів, які тісно співпрацюють, при цьому загальний процес виконується паралельно. Завдання розподіляється на частини, які є індивідуальними для кожного вузла, після цього вони об’єднюються для отримання загального результату;
* peer-to-peer мережі – децентралізована архітектура, де немає центрального сервера, який розподіляє задачі, всі вузли при цьому є рівноправними;
* просторобазована – архітектура, де створюється віртуалізація єдиного адресного простору між процесами.
  1. Волонтерські розподілені обчислення

Волонтерські обчислення – це розподілені обчислення з використанням наданих добровільно обчислювальних ресурсів користувачів. Сучасні обчислювальні системи для добровільних обчислень будуються на базі GRID-систем.

Такі проекти розподілених обчислень, як SETI@Home і Folding@Home, володіють не меншою сумарною потужністю, ніж найсучасніші суперкомп'ютери . Інтегральна продуктивність проектів на платформі BOINC за даними на 5 грудня 2013 року становить 9,966 петафлопс [1]. Продуктивність мережі Bitcoin на 6 жовтня 2013 досягла 17 000 петафлопс. Для порівняння, пікова продуктивність найпотужнішого суперкомп'ютера Tianhe-2, становить 33,86 пфлопс/сек [2].

На сьогоднішній день, для спрощення процесу організації та управління розподіленими обчисленнями, було створено декілька програмних комплексів, як комерційних, так і безкоштовних.

Загальна схема участі в проекті розподілених обчислень виглядає наступним чином:

* Потенційний учасник завантажує клієнтську частину програмного забезпечення під свою операційну систему, встановлює, налаштовує та запускає її;
* Клієнт періодично звертається до сервера проекту та запитує у нього дані для обробки, після обчислень результат. При цьому клієнт виконується з найменшим пріоритетом і не заважає основній роботі системи.

Існує декілька методів залучення учасників у проекти розподілених обчислень. Насамперед це реклама, в тому числі на веб-сайтах по суміжній тематиці проекту. Майже кожен проект намагається зацікавити потенційних учасників описом важливості розв'язуваної задачі і подальшого застосування результатів її вирішення у реальному житті. Тут, безумовно, велику вигоду можуть мати системи, у яких такі додатки є (наприклад, пошук ліків та інші біомедичні проекти).

Багато проектів створюють середовище для змагання учасників за обсягом проведених обчислень, як в особистому, так і в командному заліку. Ймовірно, стимулом для участі в таких проектах є можливість «похвалитися» перед іншими учасниками доступними обчислювальними ресурсами. Важливим фактором тут є наявність докладної і гарно оформленої статистики про виконану учасниками роботу, таблиці рейтингів, форуми для обговорення проекту і т.д. - все це утворює соціальну мережу для спілкування учасників у середовищі однодумців. При цьому власне мета обчислень проекту для багатьох з них йде на другий план і стає не так важливою, як, наприклад, деталі оформлення сайту та клієнтського програмного забезпечення. Успішність такого підходу доводить існування безлічі сайтів команд-учасниць того чи іншого проекту, які добровільно і самостійно рекламують обраний проект і залучають нових учасників.

Деякі проекти розподілених обчислень так чи інакше фінансуються і пропонують своїм учасникам грошову винагороду при досягненні певних результатів. Зокрема, проекти, що займаються пошуком рідкісних об'єктів (наприклад чисел спеціального виду), можуть нагороджувати конкретних учасників, які знайшли на своєму комп'ютері черговий об'єкт пошуку. Такі проекти можна розглядати як лотерею, в якій учасники платять своїми комп'ютерними ресурсами, обчислюючи щось потенційно корисне (або марне), і мають шанс виграти приз. При цьому шанс на успіх прямо пропорційний вкладеним потужностям - як і в лотереї: чим більше купуєш лотерейних квитків, тим більша імовірність виграшу.

Незважаючи на те, що більшість проектів розподілених обчислень грунтуються на добровільній участі, для рядового користувача існує потенційна загроза стати мимовільним учасником того чи іншого проекту, клієнтське ПЗ якого може бути встановлене комп'ютерним вірусом. При сучасному рівні розвитку інформаційних технологій і мереж передачі даних, така клієнтська програма на вірусної основі, що не завдає шкоди інформації, але займає обчислювальну потужність комп'ютера, може довгий час залишатися непоміченою.

Незважаючи на позитивні риси добровільних обчислень, часто лунає критика у їхню сторону. Організаторами розподілених обчислень спочатку заявляється безкорисливість участі всіх, хто приєднується до їх проекту, а також те, що їх результати будуть опубліковані. Однак проблема в тому, що вихідні тексти клієнтських програм більшості проектів не відкриті для загального доступу, а це означає, що кінцевий учасник проекту не має можливості проаналізувати роботу клієнтської програми - спроба дизасемблювати програму може бути протизаконна; таким чином точно невідомо, якими саме обчисленнями насправді займається клієнтське ПЗ.

Організації, які беруть участь в проектах розподілених обчислень:

* Стенфордський університет;
* Університет Берклі.

Проекти розподілених обчислень існують для різних нерозв’язаних проблем у галузі біології, медицини, математики, криптографії, природничих наук і тд. Більш повний список доступний на веб-ресурсі Distubuted Computing [8].

Для організації добровільних обчислень розроблене наступне програмне забезпечення:

* Apache Hadoop;
* BOINC - відкрита інфраструктура для розподілених обчислень Університету Берклі (Berkeley Open Infrastructure for Network Computing), розповсюджується під ліцензією LGPL;
* Condor;
* Globus Toolkit – набір програм, що значно спрощують створення і управління розподіленими обчисленнями.
  1. Веб-браузер у розподілених обчисленнях

В наш час мережа інтернет є найбільш поширеною технологією, яка об’єднює людей з усіх куточків планети. Сьогодні, за даними Miniwatts Marketing Group, 1’670’000’000 людей використовують інтернет. Google нещодавно оголосив, що є більш ніж 1 трлн унікальних URL-адрес, а також мільярди веб-сторінок в кеші. Google використовує розподілену архітектуру для сортування та збереження такого об’єму даних. Facebook збирає більше ніж 200 млрд щомісячних переглядів сторінок і обслуговує більше 600 тисяч запитів на секунду. Якщо використати ці незадіяні ресурси користувачів (процесорний час яких у переважній більшості використовується на перегляд текстової та мультимедійної інформації), то можна створити набагато потужніші, за існуючі, обчислювальні мережі, та використати їх для вирішення деяких глобальних проблем, як то пошук ліків або прогнозування стихійних лих.

Веб-браузер, як і інтернет, пройшов довгий шлях удосконалень. World Wide Web, розроблена в 1991 році, підтримувала лише простий текст, HTTP (Hypertext Transfer Protocol) та FTP ( File Transfer Protocol).

Сьогодні, через два десятиліття, браузери підтримують безліч протоколів і додатків, проте до сих пір основою передачі даних є HTTP. З моменту розробки клієнтської мови JavaScript у 1995 році, стала можливою розробка веб-додатків. JavaScript є потужною мовою з підтримкою функціональних і процедурних парадигм, що робить її гнучкою та придатною для розподілених клієнтських обчислень. Сьогодні 95% браузерів підтримують технології Web 2.0. Є й інші клієнтські технології, які доречно використовувати для розподіленої мережі, такі як Adobe Flash та Microsoft Silverlight. Але їх основним недоліком є необхідність встановлювати додаткові плагіни та розширення, що є несумісним з основними ідеями даної роботи.

Підтримка технологій HTML 5 відкриває кардинально нові можливості у використанні браузера в якості вузла розподіленої системи. Найбільш перспективні з них наведені нижче (деякі використані у даній роботі, інші є у перспективі реалізації на майбутнє):

* Web-workers – забезпечують паралельне виконання декількох потоків на сторінці (реалізовано);
* Web-sockets – безперервний двосторонній зв’язок між сервером та клієнтом (не потрібний постійний обмін запитами та контроль станк клієнта);
* WebGL – використання методів OpenGL для рендерингу графіки, у цьому випадку ресурси відеоадаптера можна використати для більш швидкого та оптимізованого обчислення певних математичних задач (наприклад за допомогою шейдерів);
* Localstorage – збереження до 50 мб даних на клієнті та зручний доступ до них у будь-який час (на відміну від Cookies, дані не відправляються з кожним запитом на сервер, а зберігаються лише в локальній пам’яті клієнта, реалізовано);
* Native Client - технологія, яка є доступною на даний лише у браузері Google Chrome та забезпечує низькорівневе виконання команд на процесорі користувача.

Зважаючи на наведені вище технології можна сказати, що браузер користувача є підходящим компонентом для реалізації вузла розподілених обчислень. Використання веб-додатків дозволяє значно швидше донести інформацію про проект до потенційних користувачів, а легке долучення нових клієнтів дозволить переосмислити процес розподілених обчислень.

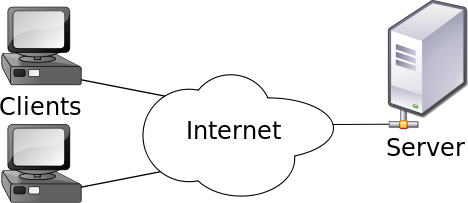
Отже основними завданнями магістерської роботи є:

* Створити розподілену мережу для кластеризації даних, обчислювальним вузлом у якій виступає веб-браузер;
* Створити веб-ресурс, який надає наступий функціонал:
  + Інформація про проект;
  + Реєстрація нових учасників;
  + Запуск обчислень;
  + Статистика (загальна та користувача);
  + Результати кластеризації;
* Реалізувати клієнтський скрипт, який забезпечує обчислення у веб-браузері.

РОЗДІЛ 2. ПІДХОДИ ДО СТВОРЕННЯ РОЗПОДІЛЕНОЇ СИСТЕМИ

2.1. Архітектура системи

У якості архітектури розподіленої мережі була обрана архітектура клієнт-сервер, як одна з найпростіших та найбільш підходящих для реалізації веб-орієнтованих обчислень (рис. 2.1)



*Рис.2.1. Клієнт-серверна архітектура*

Причини вибору такої архітектури наступні:

* Принцип побудови WWW та протоколу HTTP передбачає один сервер, до якого звертаються веб-клієнти (браузери) із запитом, та отримують результат – відповідь сервера;
* Архітектура легко реалізовується та під неї пристосовано багато програмних фреймворків.

2.2. Алгоритми кластеризації

2.2.1. Визначення понять кластерного аналізу

*Кластерний аналіз* (англ. Data clustering) — задача розбиття заданої вибірки об'єктів (ситуацій) на підмножини, що називаються кластерами, так, щоб кожен кластер складався зі схожих об'єктів, а об'єкти різних кластерів істотно відрізнялися. Завдання кластеризації відноситься до статистичної обробки, а також до широкого класу завдань навчання без вчителя. Кластерний аналіз — це багатовимірна статистична процедура, яка виконує збір даних, що містять інформацію про вибірку об'єктів і потім упорядковує об'єкти в порівняно однорідні групи — кластери (Q-кластеризация, або власне кластерний аналіз) [9].

Основна мета кластерного аналізу — знаходження груп схожих об'єктів у вибірці. Спектр застосувань кластерного аналізу дуже широкий: його використовують у археології, антропології, медицині, психології, хімії, біології, державному управлінні, філології, маркетингу, соціології та інших галузях науки. Однак універсальність застосування привела до появи великої кількості несумісних термінів, методів і підходів, що затрудняють однозначне використання і несуперечливу інтерпретацію кластерного аналізу.

*Формальне визначення кластеризації*

Нехай X~ — множина об'єктів, Y~ — множина номерів (імен, міток) кластерів. Задано функцію відстані між об'єктами \rho(x,x')~. Є кінцева вибірка об'єктів X^m = \{ x_1, \dots, x_m \} \subset X. Потрібно розбити вибірку на непересічні підмножини, що називаються кластерами, так, щоб кожен кластер складався з об'єктів, близьких по метриці \rho~, а об'єкти різних кластерів істотно відрізнялися. При цьому кожному об'єкту x_i\in X^m приписується номер кластера y_i~.

Алгоритм кластеризації — це функція a\colon X\to Y, яка будь-якому об'єкту x\in X ставить у відповідність номер кластера y\in Y. Множина Y~ в деяких випадках відома заздалегідь, проте частіше ставиться завдання визначити оптимальне число кластерів, з погляду деякого критерію якості кластеризації.

*Завдання*

Кластерний аналіз виконує наступні основні завдання:

* Розробка типології або класифікації;
* Дослідження корисних концептуальних схем групування об'єктів;
* Породження гіпотез на основі дослідження даних;
* Перевірка гіпотез або дослідження для визначення, чи дійсно групи, виділені тим чи іншим способом, присутні в наявних даних.

*Етапи*

Незалежно від конкретної сфери, застосування кластерного аналізу передбачає наступні етапи:

* Підбір вибірки для кластеризації;
* Визначення множини характеристик, по яких будуть оцінюватися об'єкти у вибірці;
* Обчислення значень тієї чи іншої міри схожості між об'єктами;
* Застосування одного з методів кластерного аналізу для створення груп схожих об'єктів;
* Перевірка достовірності результатів кластеризації.

Якщо кластерному аналізу передує факторний аналіз, то вибірка не потребує коректування — викладені вимоги виконуються автоматично самою процедурою факторного моделювання. В іншому випадку вибірку потрібно коректувати.

*Методи кластеризації*

Об'єднання схожих об'єктів у групи може бути здійснене різними способами. Для цього етапу існує цілий ряд методів:

* K-середніх (K-means);
* Нечітка кластеризація C-середніх (C-means);
* Графові алгоритми кластеризації;
* Статистичні алгоритми кластеризації;
* Алгоритми сімейства FOREL;
* Ієрархічна кластеризація або таксономія;
* Нейронна мережа Кохонена;
* Ансамбль кластеризаторів;
* Алгоритми сімейства KRAB;
* ЕМ-алгоритм;
* Метод просіювання;
* Вхідні дані;

Вхідними даними кластерного аналізу є набір об'єктів. В залежності від способу представлення цих об'єктів розрізняють такі типи вхідних даних:

* *Вектор характеристик.* Кожен об'єкт описується набором своїх характеристик; ці характеристики можуть бути числовими або нечисловими;
* *Матриця відстаней.* Кожен об'єкт описується відстанями до всіх інших об'єктів вибірки.

*Вимоги до вхідних даних*

Кластерний аналіз висуває наступні вимоги до даних:

1. Об'єкти не повинні корелювати між собою;
2. Об'єкти мають бути безрозмірними;
3. Розподіл об'єктів має бути близьким до нормального;
4. Об'єкти повинні відповідати вимозі стійкості, під якою розуміється відсутність впливу на їх значення випадкових чинників;
5. Вибірка повинна бути однорідною

**2.2.2. Алгоритм k-means**

В ході аналізу алгоритмів кластеризації було обрано алгоритм *k-means (метод k-середніх).*

*Кластериза́ція зобра́ження ме́тодом k-сере́дніх* полягає у наступному: будується деяка цільова функція *Ф(°)*,що виражає якість поточного розбиття зображення на *k* кластерів із центрами у точках *Сі, і=1,…,n; k* — задано.

Даний набір значень *(x1, x2, …, xn)*, кожне з яких представляє *d*-вимірний вектор. Метою алгоритму є розділення *n* значень на *k* множин *(k ≤ n) S = {S1, S2, …, Sk}* так, щоб мінімізувати відстань між значеннями в одному кластері.

\underset{\mathbf{S}} {\operatorname{arg\,min}} \sum_{i=1}^{k} \sum_{\mathbf x_j \in S_i} \left\| \mathbf x_j - \boldsymbol\mu_i \right\|^2  (2.1)

*де****μ****i є представленням точок у Si.*

Вибравши в початковий момент центри кластерів довільним чином, далі для кожного вектора ітеративно визначаємо його приналежність до одного із *k* кластерів і обчислюємо нові значення для центрів кластерів, намагаючись при цьому мінімізувати функцію *Ф(°)*.

Одним із недоліків цього методу є порушення умови зв'язності векторів одного кластера, ось чому розвиваються різні модифікації методу k-середніх, а також його нечіткі аналоги (*fuzzy k-means methods*), у яких на першій стадії алгоритму допускається приналежність одного пікселя до декількох кластерів (із різним ступенем принадлежності).

*Стандартний алгоритм k-means (рис.2.2):*

1. Дано проініціалізований набір *k* центроїдів  *m1,…,mk* (варіанти ініціалізації наведені нижче). Алгоритм виконується почерговим виконанням двох операцій:
2. **Крок прив’язки:** Зіставити кожний вектор з певним кластером, відстань до центру якого буде мінімальною:

S_i^{(t)} = \big \{ x_p : \big \| x_p - m^{(t)}_i \big \|^2 \le \big \| x_p - m^{(t)}_j \big \|^2 \ \forall\ 1 \le j \le k \big\}, (2.2)

*де кожне x_p є прив’язаним до лише одного S^{(t)}, навіть якщо воно може бути прив’язано до двох або більше з них;*

1. Переконатися, що в кожному кластері міститься хоча б одна точка. Для цього кожний порожній кластер потрібно доповнити довільною точкою, що розташована «далеко» від центра кластера;
2. **Крок зміни центроїдів**: Центр кожного кластера замінити середнім значенням від елементів кластера;

m^{(t+1)}_i = \frac{1}{|S^{(t)}_i|} \sum_{x_j \in S^{(t)}_i} x_j  (2.3)

1. Якщо кількість ітерацій більша за гранично допустиму або після вибору нових центроїдів їх значення не змінилися у порівнянні зі значеннями попередньої ітерації – завершити алгоритм, у іншому випадку перейти до кроку 2.

У якості функції відстані між векторами зазвичай використовується евклідова відстань. При використанні інших функцій алгоритм може перестати сходитись до єдиних центроїдів.

*Ініціалізація початкових центроїдів*

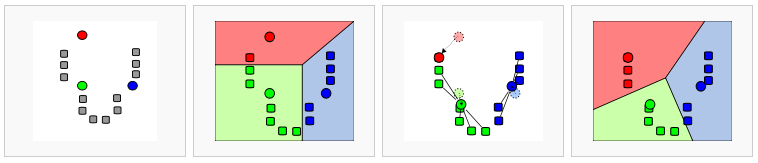
Зазвичай використовуються методи ініціалізації *Forgy* і випадкового розбиття.

*Метод Forgy* випадковим чином вибирає *K* векторів з набору даних і використовує їх як початкові центроїди.

*Метод випадкового розбиття* спочатку випадково вибирає центроїди кластерів з векторів, після цього переходить до кроку оновлення прив’язки.

*Метод Forgy* має тенденцію до розширення початкових кластерів, в той час як *метод випадкового розбиття* розміщує центроїди близько до центру набору даних.

*Метод випадкового розбиття*, як правило, кращий для алгоритмів, таких як k-harmonic means і fuzzy k-means. Для максимізації очікування і стандартних k-means алгоритмів, метод Forgy є пріоритетнішим.



2) Створено k кластерів, присвоюючи кожний вектор найближчому центроїду. Поділ на кластери представлений діаграмою Вороного

4) Кроки 2 і 3 повторюються, доки не буде досягнуте зближення

3) Оновлення центроїді відносно нових кластерів

1) Згенеровано k початкових центроїдів для набору даних (показані у кольорі)

*Рис.2.2. Схематичне представлення кроків алгоритму k-means*

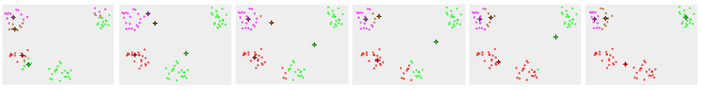
*Складність алгоритму k-means.*

За дослідженнями Девіда Артура і Сергія Васільвіцького, на деяких класах множин складність алгоритму за часом, становить  [10].

*Проблеми алгоритму k-means:*

* Не гарантується досягнення глобального мінімуму сумарного квадратичного відхилення V, а лише одного з локальних мінімумів;
* Результат залежить від вибору вихідних центрів кластерів, їх оптимальний вибір невідомий;
* Число кластерів повинне бути відомим заздалегідь.

На рис. 2.3 наведений типовий приклад збіжності методу k середніх до локального мінімуму. У цьому прикладі результат "кластеризації" методом k-середніх суперечить очевидній кластерній структурі даних. Маленькими кружками позначені точки даних, чотирипромінні зірки - центроїди. Належні їм точки даних пофарбовані в той самий колір.



*Рис.2.3. Приклад збіжності до локальних мінімумів*

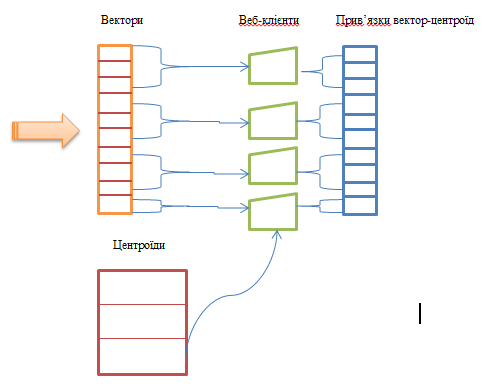
*Причини вибору алгоритму k-means як прикладу задачі для розподілених обчислень:*

* Алгоритм легко реалізовується та є відносно простим, що підходить для реалізації в інтерпретованій мові;
* Алгоритм добре піддається розпаралеленню та не потребує значних додаткових затрат на обробку результатів розподілених обчислень. На рис. 2.4, 2.5 наведена схема декомпозиції алгоритму для розподілених обчислень.

**Вхідні дані**

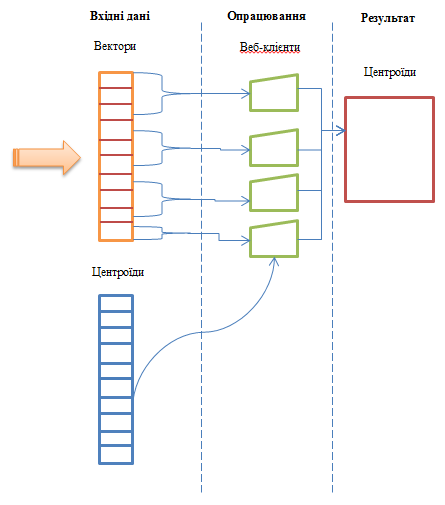
**Опрацювання**

**Результат**



**Отримання з сервера або кешу**

*Рис.2.4. Схема декомпозиції кроку прив’язки алгоритму k-means*



**Отримання з сервера або кешу**

*Рис.2.5. Схема декомпозиції кроку оновлення центроїдів алгоритму k-means*

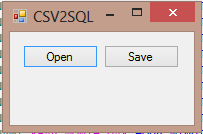
2.3. Вхідні дані системи

Даними для кластеризації можуть бути будь які тестові вибірки, статистичні дані. В даній роботі було вирішено приблизити завдання кластеризації до реальних задач та було обрано базу даних сейсмічних коливань землі в якості вхідних даних.

База даних The Significant Earthquake Database була отримана з ресурсу National Geophysical Data Center [11] . Вона включає інформацію про руйнівні землетруси з 2150 р. до н.е. до нашого часу і містить наукові дані про землетруси, які відповідають хоча б одному з наступних критеріїв:

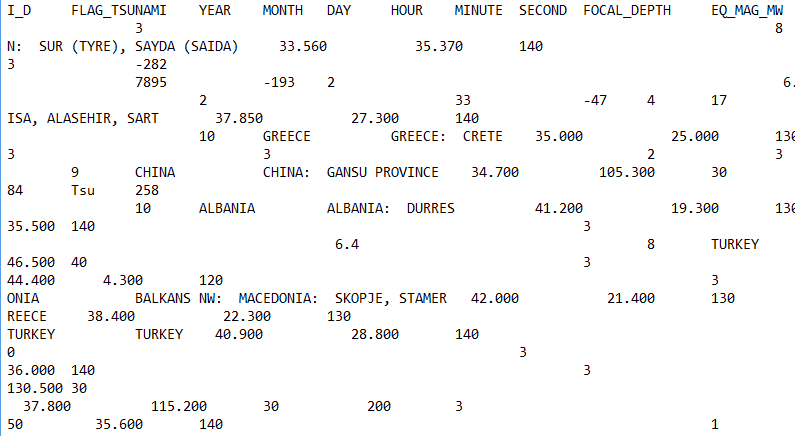
* Збитки від стихійного лиха > 1млн доларів
* Кількість людських жертв > 10
* Магнітуда землетрусу > 7,5
* Модифікована інтенсивність Меркаллі > X
* Землетрус викликав цунамі

База даних була завантажена у вигляді CSV-даних та, за допомогою написаної утиліти, конвертована у sql-запит для вставляння у базу даних (рис.2.6).



*Рис.2.6. Програма для конвертації csv-даних у sql-запити*

Фрагмент csv-даних наведений на рис.2.7:



*Рис. 2.7. Фрагмент csv-даних*

За допомогою програми-конвертера вони були занесені в базу даних наступної структури (до полів, які будуть використовуватись у системі, додані коментарі з поясненнями):

create table [dbo].Earthquake

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

CalculationId int default 0,

I\_D int not null, -- унікальний ідентифікатор

FLAG\_TSUNAMI nvarchar(3) null, -- чи викликав землетрус цунамі

YEAR int null, -- рік стихійного лиха

MONTH int null, -- місяць

DAY int null, -- день

HOUR int null, -- година

MINUTE int null, -- хвилина

SECOND decimal(3,1) null, -- секунда

FOCAL\_DEPTH int null, -- глибина залягання осередку землетрусу

EQ\_MAG\_MW decimal(3,1) null, -- магнітуда землетрусу за різними шкалами та способами вимірювань

EQ\_MAG\_MS decimal(3,1) null,

EQ\_MAG\_MB decimal(3,1) null,

EQ\_MAG\_ML decimal(3,1) null,

EQ\_MAG\_MFA decimal(3,1) null,

EQ\_MAG\_UNK decimal(3,1) null,

INTENSITY int null, -- інтенсивність землетрусу

COUNTRY nvarchar(100) null, -- країна, де відбулося стихійне лихо

STATE nvarchar(10) null, -- штат

LOCATION\_NAME nvarchar(100) null, -- назва території

LATITUDE decimal(6,3) null, -- широта

LONGITUDE decimal(6,3) null, -- довгота

REGION\_CODE int null, -- код регіону

DEATHS int null, -- к-ть смертей

DEATHS\_DESCRIPTION int null, -- опис смертельних випадків

MISSING int null, -- к-ть пропавших безвісті

MISSING\_DESCRIPTION int null,

INJURIES int null, -- к-ть поранених

INJURIES\_DESCRIPTION int null,

DAMAGE\_MILLIONS\_DOLLARS decimal(9,3) null, -- збиток, млн. доларів

DAMAGE\_DESCRIPTION int null, -- опис збитків

HOUSES\_DESTROYED int null, -- к-ть зруйнованих будинків

HOUSES\_DESTROYED\_DESCRIPTION int null,

HOUSES\_DAMAGED int null, -- к-ть пошкоджених будинків

HOUSES\_DAMAGED\_DESCRIPTION int null,

TOTAL\_DEATHS int null, -- загальна к-ть смертей

TOTAL\_DEATHS\_DESCRIPTION int null,

TOTAL\_MISSING int null, -- загальна к-ть пропавших безвісті

TOTAL\_MISSING\_DESCRIPTION int null,

TOTAL\_INJURIES int null, -- загальна к-ть поранених

TOTAL\_INJURIES\_DESCRIPTION int null,

TOTAL\_DAMAGE\_MILLIONS\_DOLLARS decimal(9,3) null, -- загальний об’єм збитків, млн. доларів

TOTAL\_DAMAGE\_DESCRIPTION int null,

TOTAL\_HOUSES\_DESTROYED int null, -- загальна к-ть зруйнованих будинків

TOTAL\_HOUSES\_DESTROYED\_DESCRIPTION int null,

TOTAL\_HOUSES\_DAMAGED int null, -- загальна к-ть пошкоджених будинків

TOTAL\_HOUSES\_DAMAGED\_DESCRIPTION int null,

);

Кластеризація даних проводиться по 3 змінних:

* Широта місця виникнення землетруса (Latitude);
* Довгота місця виникнення землетруса (Longitude);
* Інтенсивність землетрусу.

Інтенсивність землетрусу (англ. Intensity of earthquake) — міра впливу землетрусу на об'єкти, що виражається балами сейсмічної шкали в залежності від ступеню руйнування типових будівель і споруд, відчуттів очевидців, змін земної поверхні.

Інтенсивність вказана у 12-бальній шкалі MSK-64 (Медведєва-Шпонхойера-Карніка) [12].

Результат кластеризації можна інтерпретувати наступним чином: якщо кількість кластерів k=3, то можна визначити області земної поверхні, де переважно відбувалися і, можливо, відбудуться землетруси з різною інтенсивністю:

* Потужні
* Середньої потужності
* Слабкі

Такі дані не претендують на наукову достовірність, а будуть використані лише як приклад вхідних даних для задачі кластеризації.

Для визначення відстані між векторами даних, а саме координатами сейсмічних коливань, потрібна деяка модифікація функції евклідової відстані.

Координати у даних преставлені у вигляді значень градусів - дробових числах з точність 3 знаки після коми.

Можливі діапазони значень:

* Широта – від -90° до 90° (від 90 градусів південної широти до 90 градусів північної широти);
* Довгота – від -180° до 180° (від 180 градусів західної широти до 180 градусів східної довготи);
* Інтенсивність – від 0 до 12 балів (чим більше значення, тим більш потужний землетрус).

Формула визначення відстані між векторами прийме наступний вигляд:

function euclidianDistance(a, b) {

var dV1 = Math.abs(a.V1 - b.V1);

var dV2 = Math.abs(a.V2 - b.V2);

var dV3 = Math.abs(a.V3 - b.V3);

var maxdV1 = 180;

var maxdV2 = 360;

if (dV1 > maxdV1/2)

dV1 = maxdV1 - dV1;

if (dV2 > maxdV2/2)

dV2 = maxdV2 - dV2;

ds = Math.sqrt(dV1 \* dV1 + dV2 \* dV2 + dV3 \* dV3);

return ds;

};

*Де a, b – вектори для визначення відстані між ними,*

*.V1 – широта,*

*.V2 – довгота,*

*.V3 – інтенсивність,*

Для широти і довготи у функції вводиться поправка: якщо абсолютна відстань між точками (різниця у градусах) більша граничної величини (половини повної окружності), то вона віднімається від величини повної окружності. Тобто якщо довгота двох точок буде відповідно -180° і 180°, то відстань між ними повинна бути 0.

2.4. Вихідні дані системи

Система несе в собі дві мети:

* Зібрати статистику роботи розподіленої мережі, яка повинна включати наступні дані:
  1. Кількість користувачів (загальна, активних);
  2. Потужність системи (середня швидкість обробки даних, загальна, пікова);
  3. Об’єм оброблених даних (поточний та загальний об’єм даних, дані які обробляються, номер ітерації);
* Вивести результати кластеризації даних (отримані кластери, їх центри та вектори, які їм належать).

Вихідні дані повинні бути візуалізовані у інтерактивному вигляді (на веб-сторінці). Для даних статистики це можуть бути як текстові дані, так і графіки, які показують залежність параметрів від часу роботи системи.

Результати кластеризації (враховуючи специфіку вхідних даних, які характеризують потужність та місце виникнення сейсмічних коливань) повинні бути візуалізовані на карті землі. Для більш кращого візуального представлення, дані з різних кластерів повинні бути виведені як точки з різними кольорами.

2.5. Критерії оцінки результатів

Для доведення доцільності та ефективності представленої розподіленої веб-орієнтованої системи, повинні бути досягнуті наступні результати:

* Результати кластеризації повинні відповідати візуальному групуванню векторів та, в деякій мірі, статистиці по цих даних;
* При достатній к-ті користувачів (>10 осіб) система повинна продемонструвати перевагу та більшу потужність обчислень у порівнянні з обчисленнями, які проводяться на зосередженій системі. Дану оцінку можна провести, проаналізувавши статистику роботи системи.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОЇ СИСТЕМИ

* 1. Специфікація вимог до розподіленої системи
     1. **Вступ**

Призначення, мета

Розподілена веб-орієнтована система обчислень призначена для кластеризації даних з використанням обчислювальних ресурсів веб-клієнтів. У якості даних для кластеризації була обрана інформація про інтенсивність та місце виникнення сейсмічних коливань землі. Основна ідея програмного засобу – використання клієнтського інтернет-браузера як основного обчислювального вузла системи.

Продукти-аналоги

До аналогів системи можна віднести існуючі системи розподілених волонтерських обчислень (наприклад на основі BOINC) [1].

Перспективи продукту

Існуючі аналоги забезпечують велику сумарну потужність, але містять проблеми щодо залучення нових користувачів та необхідності встановлення додаткового програмного забезпечення. Дана система повинна забезпечити новий підхід до реєстрації користувачів та запуску обчислень, ефективність обробки даних та масштабованість. Також вона повинна бути пристосованою до модифікацій, щоб при потребі можна було замінити як задачу для обчислень, так і вхідні дані. Система може використовуватись як платформа для волонтерських обчислень певної ресурсоємкої задачі. А також як один з інструментів обробки великих об’ємів даних ресурсами клієнтів. Перспективами системи є залучення великої кількості користувачів, завдяки розміщенню на популярному ресурсі, та досягнення значної сумарної потужності.

* + 1. Загальний опис

**Характеристики продукту**

У ході виконання проекту, потрібно досягнути такі цілі:

* Створити веб-ресурс, який буде надавати наступну функціональність:
  + Інформація про проект;
  + Реєстрація нових користувачів;
  + Запуск обчислень;
  + Статистика роботи (по користувачу та загальна);
  + Результати кластеризації.
* Створити скрипт з алгоритмом обробки даних, для виконання у клієнтських браузерах;
* Розгорнути систему на зовнішньому ресурсі та зібрати статистику її роботи.

Класи користувачів та їх характеристики

Система передбачає 2 основні класи користувачів:

* Авторизований користувач (клієнт розподіленої мережі) – виконує обчислення у веб-браузері, може переглядати статистику своїх попередніх обчислень та виконаних задач;
* Анонімний користувач – виконує обчислення у веб-браузері, може переглядати інформацію про проект та статистику його роботи;

Середовище функціонування

*Сервер*

Веб-сервер IIS з веб-додатком ASP.NET MVC

*Клієнт*

Браузер з підтримкою HTML5 та необхідних технологій для функціонування клієнтського скрипта

* + 1. Характеристики системи

***Перегляд інформації про проект та статистики роботи системи***

***Опис і пріоритет:***

Система повинна надавати інформацію про проект (його ідею, мету), алгоритм обробки, дані та результат. А також відображати статистику роботи системи. Пріоритет – високий.

Послідовність дія/відгук:

*Таблиця 3.1.*

|  |  |
| --- | --- |
| **Дія користувача** | **Відповідь системи** |
| Перехід на сторінку  «Про проект» | Сторінка інформації про проект. Пропозиція приєднатися до системи розподілених волонтерських обчислень. Динамічно оновлювана статистика роботи системи. |

***Функціональні вимоги:***

REQ1: Система повинна надавати інформацію про проект розподілених обчислень (ідея, мета, принципи роботи, алгоритм, дані).

REQ2: Система повинна надавати динамічно оновлювану статистику обчислень в системі, а саме:

* Кількість користувачів протягом всього часу роботи системи;
* Кількість користувацьких сесій протягом всього часу роботи системи;
* Кількість користувачів онлайн;
* Об’єм обчислених даних;
* Пікова потужність системи;
* Поточна потужність системи.

***Запуск розподілених обчислень***

***Опис і пріоритет:***

Система повинна забезпечувати приєднання користувачів до розподіленої мережі та запуск обчислень. Пріоритет – високий.

Послідовність дія/відгук:

*Таблиця 3.2.*

|  |  |
| --- | --- |
| **Дія користувача** | **Відповідь системи** |
| Перехід на сайт ресурсу | Сторінка інформації про проект. Пропозиція приєднатися до системи розподілених волонтерських обчислень |
| Перехід на сторінку обчислень | Запит про реєстрацію або вхід користувача. Повинна бути можливість увійти в систему як анонімний користувач. |
| Авторизація користувача або реєстрація нового | Сторінка запуску розподілених обчислень з поточною статистикою роботи системи. Відправка листа на пошту новому користувачу з даними для входу. |
| Прямий перехід на сторінку обчислень за допомогою url | Автоматичний запуск сесії обчислень |
| Запуск обчислень (після кліку на кнопку) | Запуск сесії обчислень |
| Обчислення завершено | Повернення результату на сервер. Запит наступного блоку даних для обчислень. Запуск нової сесії. |
| Зупинка обчислень | Інформація про проведені обчислення. Початок наступної сесії обчислень. |

***Функціональні вимоги:***

REQ1: Система повинна надавати можливість приєднання до обчислень як авторизованим (з можливість перегляду попередньої статистики), так і анонімним користувачам; повинна бути можливість старту обчислень по посиланню (та автоматична авторизація через передані параметри);

REQ2: Вхід у систему повинен проводитися з використанням пошти та паролю користувача. Після реєстрації нового користувача система повинна відправляти лист з інформацією та даними для входу на пошту користувача;

REQ3: Обчислення повинні проводитись добровільно та прозоро для користувача, з виведенням та динамічним оновленням інформації про поточні обчислення, а саме:

* Номер сесії (перед запуском обчислень);
* Час запуску обчислень;
* Поточний інтервал даних, які обробляються;
* Об’єм оброблених даних;
* Швидкість обчислень (к-ть векторів за секунду);
* Статус сесії обчислень (запуск, обчислення, зупинено);
* Індикатор прогресу з інформацією про виконану роботу (у відсотках).

REQ4: Після зупинки обчислень повинна відображатися статистика по сесії обробки даних;

REQ5: При закритті сторінки повинна появлятися пропозиція залишитися для продовження обчислень.

***Перегляд результатів роботи системи***

***Опис і пріоритет:***

Система повинна відображати результати роботи системи (оброблені дані, їх аналіз та статистику). Пріоритет – середній.

Послідовність дія/відгук:

*Таблиця 3.3.*

|  |  |
| --- | --- |
| **Дія користувача** | **Відповідь системи** |
| Перехід на сторінку результатів роботи системи | Результат кластеризації у вигляді візуалізованих кластерів, накладених на карту землі. |

***Функціональні вимоги:***

REQ1: Обчислення системи повинні проводитись над набором даних про сейсмічну активність землі та включати наступні параметри:

* Широта;
* Довгота;
* Інтенсивність.

Результатом роботи системи повинний бути набір кластерів (груп), сформований алгоритмом k-means (який описаний вище).

REQ2: Візуалізація роботи системи повинна проводитись на інтерактивній карті (наданій стороннім ресурсом, наприклад Google) з накладеними областями кластерів;

REQ3: Система повинна надавати статистику роботи у вигляді інтерактивних графіків для наступних параметрів:

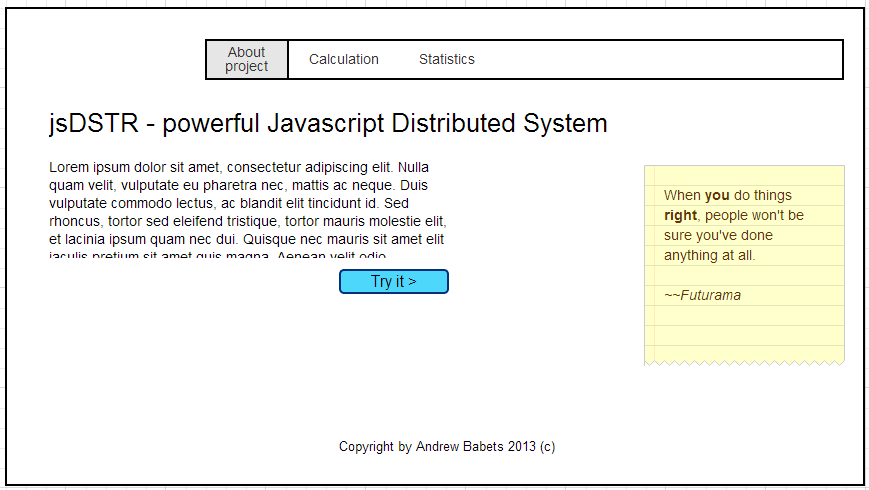
* Швидкість обчислень;
* Кількість користувачів / онлайн;
* Потужність системи;
  + 1. Вимоги зовнішніх інтерфейсів

Користувацькі інтерфейси

Система повинна бути кросбраузерною. Дизайн повинен бути адаптивний до розширення екрану. Макет сайту повинен включати меню з посиланнями вверху сторінки, нижню частину з інформацією про проект та автора і параграф з статистикою роботи системи. Центральна область призначена для наповнення даними сторінки (рис. 3.1). Система повинна надавати локалізовану версію інтерфейсу (3 мови – українська, англійська та російська). На сторінках, де використовуються конфіденційні дані користувача, повинен бути застосований протокол HTTPS для захисту даних, які передаються.

Сайт повинен надавати наступні сторінки:

* Стартова сторінка [/] – назва та коротка мета проекту (стартова сторінка);
* Про проект [/about] – інформація про проект (рис.3.1.);



Наповнення сторінки

Статистика

Меню

Футер

*Рис.3.1. Макет сторінки «Про проект»*

* Сторінка авторизації користувача та початку обчислень [/processing] (рис.3.2,3.3);

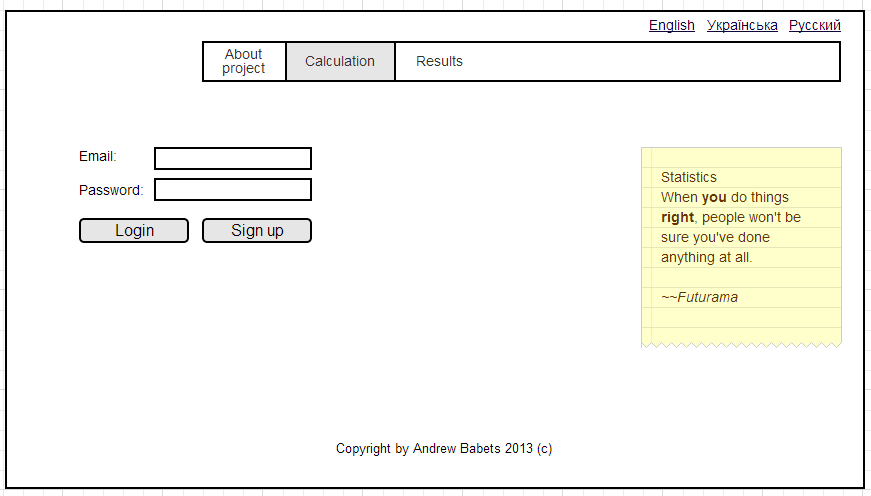


Рис.3.2. Макет сторінки авторизації користувача

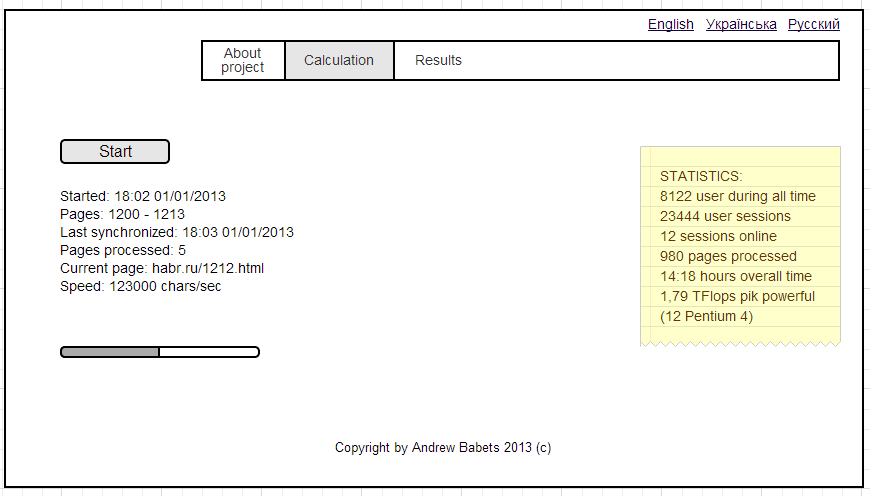


Рис.3.3. Макет сторінки обчислень

* Сторінка результатів роботи системи [/results] – результати кластеризації, статистика роботи системи у вигляді графіків.

Програмні інтерфейси

Для роботи *веб-сервера* необхідні наступні компоненти:

* бібліотеки .NET;
* MVC 3.0;
* встановлений та налаштований Microsoft SQL Server 2008 (зі встановленою базою даних);
* IIS з налаштованим сервісом WCF для API;

Для нормального функціонування обчислень на клієнті необхідно веб-браузер з включеним javascript, який підтримує наступні технології:

* Web-sockets;
* Localstorage.
  + 1. Інші нефункційні вимоги

Вимоги продуктивності

Враховуючи втрати ефективності при комунікації, об’єднанню результатів розподілених обчислень та збереженні даних, необхідно забезпечити більшу сумарну потужність при запуску системи на 10 клієнтах у порівнянні з потужність зосередженої системи (процесор Core i7, 4 гб RAM).

Для цього необхідно максимально оптимізувати роботу наступних компонент системи:

* Обчислювальні алгоритми клієнта (також мінімізація javascript файлів);
* Комунікація між клієнтом та сервером (оптимізована передача даних через ajax);
* Комунікація між веб-сервером та базою даних (оптимізація запитів sql, вибір підходящої технології взаємодії з базою даних);
* Алгоритм розподілення даних для обчислень.

Вимоги безпеки

Система повинна надавати гарантію безпеки конфіденційних даних користувачів. Також необхідно максимально передбачити спроби злому системи та виведення її з ладу.

Атрибути якості програмного продукту

Атрибутами якості даної системи можна вважати її ефективність та практичну цінність у порівнянні із зосередженими обчисленнями. Ці атрибути повинні забезпечуватися зручним інтерфейсом системи, її загальною потужністю, масштабованістю та легкістю залучення користувачів.

* 1. Технології програмної реалізації розподіленої системи

Для реалізації програмного засобу було використано наступні технології та ресурси (їх опис та порівняння з альтернативами):

* *СУБД MS SQL* – основними її перевагами є висока інтеграція з сервісами Microsoft .NET, широкий спектр налаштувань для оптимізації запитів, пристосованість до великих навантажень. Аналоги – MySql (безплатна, але надає менш потужні засоби для розробки баз даних), Oracle (платна), PostgreSQL;
* *Веб-сервер IIS* – його перевагами є зручність налаштувань та запуску, підтримка ASP.NET MVC веб-додатків;
* *Веб-додаток ASP.NET MVC* – використання паттерну MVC дозволяє зручно розділяти представлення від даних, гнучко налаштовувати «дружні» url-адреси, застосовувати стандартні засоби для динамічних запитів Ajax. Крім того ASP.NET MVC у більшість мірі відповідає ідеології протоколу HTTP у порівнянні з Web Forms (концепція запит-відповідь, відсутність механізму збереження стану веб-сторінки, що значно зменшує розмір переданих даних);
* *Microsoft .NET* – програмна платформа, яка є досить поширеною та входить в дистрибутив останніх версій Windows. Забезпечує такі ключові переваги, як загальномовне середовище виконання (CLR), що дозволяє використовувати будь-які мови програмування, які підтримують .NET, автоматичне управління пам’яттю, підтримку багатьох програмних фреймворків та бібліотек;
* *Мова програмування C#* - основними перевагами є зручний синтаксис, підтримка різноманітних мовних конструкцій (в тому числі LINQ), підтримка ООП (що є досить важливим для великого проекту), велика кількість готових бібліотек;
* *LINQ TO SQL* – механізм запитів до бази даних, основною перевагою є значно простіша взаємодія з кодом, розширюваність, підтримка анонімних типів, методів-розширень, дерев виразів. LINQ TO SQL є ефективнішою та більш економною, у порівнянні з Entity Framework при невеликій складності логіки бази даних;
* Windows Forms – використана для програми-конвертера даних з csv-формату у запити sql. Дозволяє швидко та зручно розробляти інтерфейс користувача для Windows програм.
* *Ajax* – підхід до побудови інтерактивних користувацьких інтерфейсів веб-додатків, які підтримують динамічний обмін інформацією між браузером та сервером. Є однією з ключових технологій розподілених веб-орієнтованих обчислень. Існує декілька підходів реалізації Ajax, у даному проекті використаний об’єкт XMLHttpRequest. Альтернативами є використання таких технологій як Flash та Silverlight, але їхнім недоліком є необхідність додаткових розширень для браузера.
* *Javascript* – основна клієнтська мова у браузері, дозволяє створювати інтерактивні сторінки, є одним з компонентів DHTML. У даному проекті використана як мова для клієнтських обчислень у веб-орієнтованій розподіленій системі.
* *HTML 5* – версія мови HTML та набір клієнтських технологій. У даному проекті використані наступні:
  + *WebWorkers* – технологія для запуску декількох програмних потоків з основного потоку javascript. Дозволяє реалізувати розподілені обчислення у фоні, не заважаючи роботі основного інтерфейсу та інших вкладок браузера;
  + *Localstorage* – технологія зберігання даних даних на клієнті, які, на відміну від Cookies, не передаються на сервер з кожним запитом (можна зберігати до 50 мб даних або більше при знятті обмежень у браузері). У даному проекті використовується для кешування даних кластеризації та подальшому отриманні їх з локального сховища.
* *JSON* – формат передачі даних, більш короткий у порівнянні з XML, є «рідним» форматом для javascript та дозволяє проводити серіалізацію та десереалізацію даних. У даному проекті використовується для передачі даних між клієнтом та сервером;
* *jQuery* – javascript фреймворк для полегшення розробки веб-сторінок, виконання стандартних операцій і тд;
* *Bootstrap* – безплатний набір інструментів для створення сайтів та веб-додатків. Містить в собі HTML та CSS шаблони оформлення для веб-форм, кнопок, міток, блоків навігації і інших компонентів веб-інтерфейсу, включаючи javascript розширення. У даному проекті використовується для кросбраузерності, верстки та шаблонів оформлення;
* *Parallel.js* – бібліотека javascript, «обгортка» над функціональністю webworkers;
* *GIT* – система контролю версій, дозволяє гнучко управляти історією коду, змінами. Альтернативою є SVN, але GIT забезпечує більш широкі можливості по злиттю змін та дозволяє зберігати локальні репозиторії (без необхідності зв’язку із сервером);

Адрес публічного репозиторію проекту: [13];

* *Github [github.com]* – ресурс для моніторингу завдань проекту, який надає безплатний репозиторій, інструменти для роботи з ним та можливість обміну інформацією між користувачами;
* *Somee.com* – хостинг проектів asp.net mvc, включає безплатні тарифні плани.

Проект розміщений за адресом:

http://jsd.somee.com.

* 1. Проектування розподіленої системи

3.3.1. UML діаграми

Для полегшення проектування системи було розроблено наступні UML-діаграми:

* Діаграма класів (дод. Б)
* Діаграма послідовностей (дод. В)
* Діаграма компонентів (дод. Д)
* Діаграма варіантів використання (дод. Е)

**3.3.2. Проектування бази даних**

У дод. А наведена структура бази даних проекту.

Авторизація користувачів та збереження їх даних проводиться через стандартні механізми ASP.NET Forms Authentification із автоматично згенерованими таблицями.

У якості інтерфейсу між базою даних обраний LINQ TO SQL, який забезпечує зручну інтеграцію запитів в код через LINQ та генерування класів-моделей з таблиць. При складних запитах (коли потрібно провести оптимізацію або використати конструкцію, яку LINQ не підтримує) використовується ADO.NET.

**3.3.3. Проектування бізнес-логіки**

У якості веб-платформи було обрано ASP.NET MVC. На рис. 3.4 представлена схема взаємодії компонентів MVC.

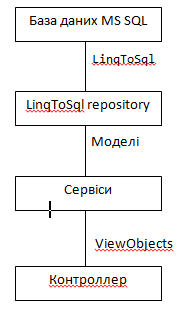


*Рис.3.4. Схема взаємодії компонентів MVC*

Як можна побачити з цієї схеми, браузер посилає запит на веб-сервер, у відповідності до Routes сервер перенаправляє запит на один із контроллерів (саме тут відбувається конвертування з «дружніх» url у відповідні url контроллера). Після отримання управління контролер може звернутися до моделі (щоб змінити або отримати значення в базі даних) та формує view, який відправляється у вигляді html сторінки користувачу.

При проектуванні архітектури було вирішено зробити додатковий рівень сервісів між контролером та моделями, який надає доступ контроллеру лише до певних дій, які визначає сервіс. Це дозволяє полегшити розуміння коду та забезпечує відокремлення бізнес-логіки від логіки представлення.

Усі методи сервісів повертають контроллеру не саму модель, а viewobject, який є представленням моделі. Це надає контроль над значеннями, які отримує клієнт (наприклад можна серіалізувати лише потрібні поля для клієнтського об’єкта) та забезпечує певний рівень безпеки доступу до об’єктів бази даних. На рис. 3.5 можна побачити модель архітектури системи.



*Рис.3.5. Модель архітектури системи*

У системі присутні наступні класи:

* Моделі (Centroid, CentroidAssignment, Earthquake, KmeansCalculation, Log, Session, Settings, Task);
* Сервіси (KmeansService, LogService, SettingsService, SessionService);
* ViewObjects (VectorViewObject, AssignmentViewObject, SessionViewObject, StatisticsViewObject);
* Helpers (ConfigurationHelper, ResourcesHelper, Singleton);
* Контроллери (AboutController, ProcessingController, ResultsController, StatisticsController);
* Views (AboutView, ProcessingView, ResultsView, StatisticsView – partial view);

Для забезпечення єдиного шаблону сторінок, була створена master-сторінка, яка містить спільні елементи для усіх сторінок сайту. Для полегшення створення макету сторінок та стилів стандартних елементів інтерейсу, було вирішено використовувати бібліотеку Bootstrap, яка надає стандартні компоненти для веб-розробки. Вона базується на jQuery, цей фреймворк також включений у проект.

Основий акцент було надано динамічному оновленню інформації на сторінках за допомогою технології ajax

Усі javacript скрипти є мінімізованими (за допомогою утиліти) для пришвидшення загрузки та кращої оптимізації коду.

**3.3.4. Логіка розподілених обчислень**

Усю логіку розподілених обчислень можна умовно поділити на дві наступні групи:

* Логіка розподілених обчислень клієнт-сервер (незалежна від вхідних даних та методу обробки). Включає наступні поняття:
  + *Користувацька сесія (клас Session)* – об’єкт, який представляє користувацьку сесію обчислення. Він має 3 стани (State):
    - *Started* – сесія запущена та обчислюється на клієнті;
    - *Stopped* – сесія зупинена (через клієнтську, серверну помилку або зупинку сесії користувачем);
    - *Completed* – обчислення завершені та результат збережений на сервері.

*StateMessage* – повідомлення про останню подію, пов’язану із сесією;

*Guid* – унікальний ідентифікатор сесії, який передається між клієнтом та сервером;

*UserName* – користувач, якому належить сесія;

*CalculationId* – ідентифікатор поточних обчислень (глобальних);

*CreatedDate* – час створення сесії;

*ChangedDate* – час останнього оновлення сесії;

* + *SessionService* – сервіс який відповідає за управління сесіями. Включає наступні методи:
    - *CreateSession* – викликається при запиті клієнта на створення нової сесії;
    - *PingSession* – метод для перевірки стану сесії. Клієнт після запуску обчислень циклічно викликає ping з деяким інтервалом часу (в даному випадку 1 секунда). Якщо ping є неуспішним (виникла деяка помилка), він повторюється до максимальної кількості спроб (5). Якщо всі вони є неуспішними, сесія зупиняється та запускається нова. На сервері ping оновлює час останньої зміни сесії, який використовується у методі TerminateInvactiveSessions;
    - *CompleteSession* – клієнт повертає результат обчислень на сервер та починає нову сесію;
    - *CancelSession* – сесія завершується після деякої помилки (неправильний стан або виникла виключна ситуація);
    - *TerminateInvactiveSessions* – завершення всіх неактивних сесій, останній час оновлення яких раніше, ніж 1 хвилина тому. Таким чином забезпечується контроль активності клієнта.
* Логіка розподіленого алгоритму k-means

Вона включає наступні сутності:

* + *Earthquake* – об’єкт даних сейсмічної активності, він описаний вище;
  + *KmeansCalculation* – об’єкт, який представляє поточні глобальні обчислення. Містить наступні властивості:
    - *K* – кількість кластерів для алгоритму k-means;
    - *Iteration* – поточна ітерація обчислень;
    - *MaxIterations* – максимальна к-ть ітерацій, після чого обчислення вважаються завершеними;
    - *State* – стан обчислень, можливі наступні значення:
      * *Started* – обчислення k-means створене та очікує ініціалізації (визначення випадкових початкових центроїдів тощо);
      * *AssignmentLoop* – крок обчислення прив’язки векторів до кластерів;
      * *UpdateCentroidsLoop* – крок оновлення центроїді;
      * *Completed* – обчислення успішно завершене;
      * *Failed* – обчислення зупинене через помилку;
      * *Successful* – успішне виконання операції (стан для клієнта);
      * *Error* – помилка виконання операції (стан для клієнта);
      * *Busy* – усі завдання поточного кроку у стані виконання (очікування на завершення користувацьких сесій).
    - *StateMessage* – інформація про останню зміну обчислень.
  + *Centroid* – інформація про обчислений центроїд. Містить дані про центр кластера, обчислення та ітерацію, до якої він відноситься;
  + *CentroidAssignment* – інформація про прив’язку вектора до певного кластера;
  + *Task* – обчислювальна задача для клієнта. Містить наступні властивості:
    - *State* – поточний стан задачі, може приймати наступні значення:
      * *Idle* – невирішене;
      * *Started* – обчислюється;
      * *Completed* – завершене;
      * *Cancelled* – відмінене (наприклад сталася деяка помилка).
    - *Type* – поточний крок обчислень;
    - *SessionGuid* – до якої користувацької сесії відноситься;
    - *Iteration* – номер ітерації обчислень;
    - *SlotStart* – початок даних для обчислень;
    - *SlotCapacity* – розмір даних для обчислень.
  + *KmeansService* – сервіс для отримання клієнтських задач для кластеризації та збереження обчислених результатів. Включає наступні методи:
    - *GetTask* – отримання наступного завдання або запуск нових обчислень;
    - *CompleteTask* – збереження результатів обчислень;
    - *CancelTask* – завершення задачі обчислень (наприклад у випадку помилки або з метода TerminateInvactiveSessions).

У дод. А наведено схематичне представлення структури KmeansService та його вихідний код.

На рис. 3.6 наведено схематичну модель комунікації клієнта і сервера:

**Client (js)**

**CreateSession**

**Calculation…………...**

**PingSession**

**CompleteSession**

**CancelSession**

**SessionService**

**CreateSession**

**PingSession**

**CompleteSession**

**CancelSession**

*Рис.3.6. Модель комунікації клієнта та сервера*

**3.4. План забезпечення якості розподіленої системи**

Для забезпечення якості розподіленої системи було розроблено план забезпечення якості ПЗ згідно з вимог стандарту IEEE 730 SQAP. В цьому документі проводиться аналіз системи згідно вимог зазначених у стандарті для забезпечення якості програмного забезпечення.

**3.4.1.Мета**

Даний документ визначає методику проведення процедур для забезпечення якості програмного забезпечення та визначення методів та стандартів, які будуть використовуватися для забезпечення якості проекту веб-орієнтованої розподіленої системи кластеризації даних.

Метою є забезпечення належного рівня якості розроблюваного програмного продукту шляхом виявлення та виправлення помилок, що виникають в процесі розробки системи підтримки прийняття рішень. Також проводиться перевірка стабільності та продуктивності роботи системи.

**3.4.2. Управління проектом розподіленої системи**

Проект повинен розроблятися за ітеративною методикою. Кожна ітерація передбачає удосконалення існуючого продукту та деяку проміжну версію функціоналу. Перед виходом нової версії проводиться тестування та верифікація реалізованих змін та загальне тестування програмної системи. Після цього оформляється документ, в якому фіксується кількість та види проведених тестів, а також їх результати.

**3.4.2.1. Організація проведення робіт по розробці розподіленої системи**

Контроль якості спроектованого програмного забезпечення проводиться керівником проекту. Він визначає, чи розроблений продукт відповідає функціональним вимогам та поставленому завданню. Забезпечення якості виконується особою, яка працює над проектом.

**3.4.2.2. Завдання учасників проекту розробки розподіленої системи**

Теоретична підготовка та програмна реалізація проекту забезпечується студентом, який проводить магістерське наукове дослідження. Повинний бути проведений аналіз інформації по заданій темі та реалізований готовий програмний продукт.

Керівник проекту здійснює контроль над виконанням проекту та його відповідністю поставленій меті.

**3.4.2.3. Ролі та зобов’язання кожного з учасників проектування розподіленої системи**

Виконавець магістерського дослідження відповідає за виконання поставлених завдань у проектній роботі. Керівник магістерського проекту здійснює контроль виконання робіт у магістерському дослідженні.

**3.4.2.4. Оцінка ресурсів для забезпечення якості розподіленої системи**

У виконанні магістерської роботи основним ресурсом є час, оскільки дослідження проводиться у рамках навчального процесу. Затрати на забезпечення якості будуть становити 15% від загального часу реалізації системи.

**3.4.3. Документація**

**3.4.3.1. Мета**

Результатом виконання магістерського дослідження повинен бути оформлений звіт про виконання проведених робіт, у якому описуються основні положення проведеного дослідження та отримані результати. Звіт повинен містити аналіз предметної області, методик та підходів до реалізації проекту. Також він повинен містити розділ програмної реалізації,

.

**3.4.3.2. Мінімальні вимоги до документації**

Вимоги до звіту магістерського дослідження описані у методичних вказівках, виданих університетом НУ ЛП.

**3.4.4. Стандарти, практики, конвенції і метрики**

Даний розділ не відповідає плану забезпечення якості, тому що проект виконаний у формі дослідження та не передбачає метрик контролю.

**3.4.5. Тестування**

Тестування програмного продукту проводилось ручною методикою, усі знайдені помилки були зазвітовані у системі контролю версій github [13].

# РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ РОЗПОДІЛЕНОЇ СИСТЕМИ

# 4.1. Результати кластеризації даних

У ході дослідження експериментальним шляхом було встановлено наступні оптимальні параметри алгоритму k-means та розподілення:

* *Максимальна кількість ітерацій* – 10, після чого обчислення вважаються завершеними (центроїди досягнули потрібної точності);
* *Кількість кластерів (K)* – 4. У контексті даних, що обробляються, вони можуть інтерпретуватися як:
  + Зони з повишеною сейсмічною активністю (1);
  + Зони з вище середньої сейсмічною активністю (2);
  + Зони з середньою сейсмічною активністю (3);
  + Зони з низькою сейсмічною активністю (4).
* *Розмір блоку даних для клієнтської задачі обчислення кроку прив’язки* – 200 векторів;
* *Розмір блоку даних для клієнтської задачі обчислення кроку оновлення центроїдів* – 200 векторів;

Обчислення кластеризації даних k-means закінчилися на 10 ітерації, оновлені значення центроїдів повністю збіглися з попередніми значеннями, отже початкові значення центроїдів були вибрані оптимально (методом випадкового розбиття).

Отримані наступні значення центроїдів подані у табл. 4.1:

*Таблиця 4.1. Кінцеві значення центроїдів*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Широта | Довгота | Інтенсивність | Інтерпретація даних кластера (рівень сейсмічної активності) |
| 1 | 27.849 | 32.462 | 8.468 | Середня |
| 2 | 20.600 | 122.000 | 7.000 | Низька |
| 3 | -9.200 | 113.100 | 9.000 | Висока |
| 4 | 40.301 | 69.823 | 9.000 | Вища середньої |

Якщо накласти координати центроїдів на карту землі, то отримаємо наступні результати (рис.4.1):



*Рис.4.1. Накладення центроїдів на карту*

Точки центроїді попадають у наступні місця планети:

1. Середня сейсмічна активність – Ель-Менія, Єгипет.

Єгипет знаходиться недалеко біля сейсмічно-активних областей (Середземномор’я, Кавказ). Останнім часом поступають у ЗМІ новини про незначні поштовхи силою 4-5 балів у Єгипті, які виникають переважним чином у пустелі та не приносять значної шкоди. Останній випадок землетрусу відбувся 7 квітня 2013 року з магнітудою 4,8 у східних провінціях Єгипта. Отже результат обчислення кластера можна вважати достовірним.

1. Низька сейсмічна активність – Філіпінське море.

Філіпіни лежать біля так званого тихоокеанського сейсмічного поясу, тому коливання земної кори там поширене явище. Останній сильний землетрус відбувся 31 серпня 2012 року у Філіпінському морі магнітудою 7,6, у результаті одна людина загинула та одна була поранена. Результати можна вважати достовірними.

1. Висока сейсмічна активність – Індійський океан біля берегів о.Ява, Індонезія.

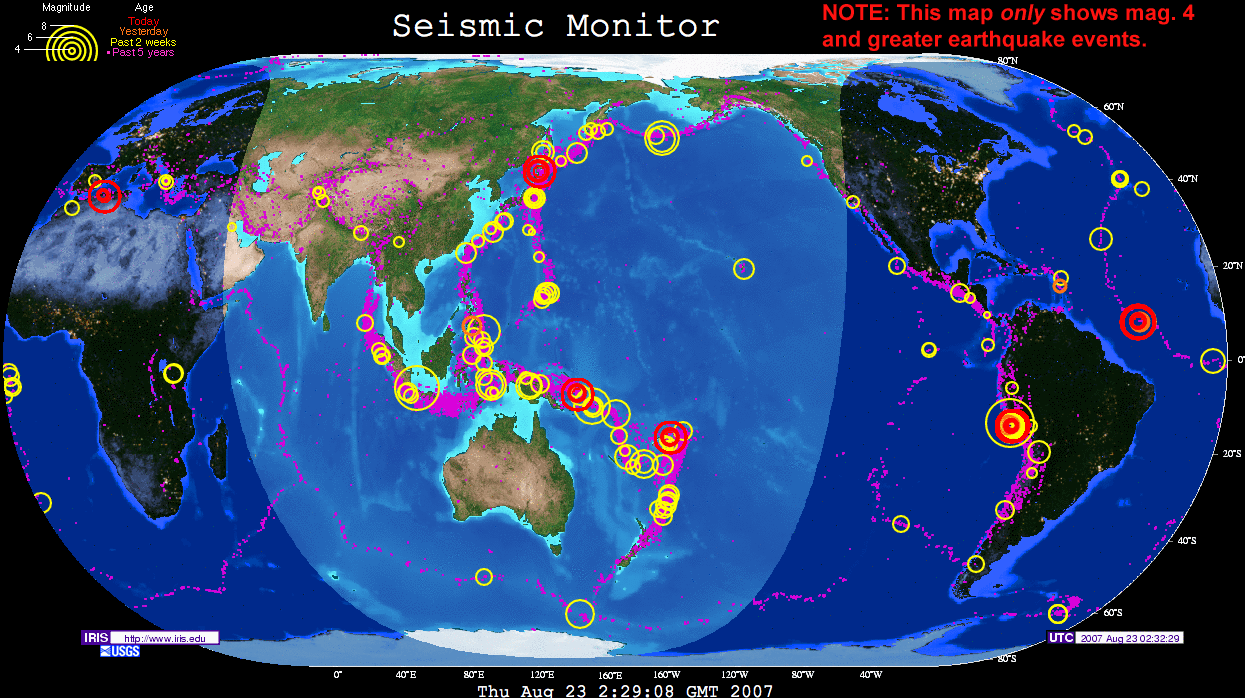
Індонезійський архіпелаг був здавна осередком найсильніших землетрусів, зареєстрованих на планеті. 11 квітня 2012 року біля західного побережжя Північної Суматри відбувся землетрус силою 8,6 бала, який став 11-м по магнітуді з 1900 року, маючи силу, нетипову для внутрішньоплитових землетрусів. Крім цього з 2009 року в Індонезії відбулося ще 5 потужних землетрусів магнітудою 7-8 балів. Результати можна вважати достовірними.

1. Вища середнього сейсмічна активність – Худжанд, Таджикистан.

Велика концентрація гір (93% країни) та близькість інших сейсмічних зон обумовлює виникнення руйнівних землетрусів. Найбільш потужним можна назвати землетрус в Ашхабаді (Туркменістан), який вважається одним з найбільш руйнівних, сила в епіцентрі склала 9-10 балів, магнітуда землетрусу 7,3, у результаті землетрусу в Ашхабаді було зруйновано 90-98% всіх будівель та загинуло 89% населення. Враховуючи ці та інші дані, можна вважати результати достовірними.

Отже кожний з центроїдів кластерів з деякими похибками відповідає реальним даним по сейсмічній активності.

Для порівняння отриманих даних з науковими, було використано ресурс IRIS Seismic Monitor [14], де міститься сейсмічна карта землі (рис. 4.2).



*Рис.4.2. Сейсмічна карта землі*

Порівнявши дані, можна зробити висновок, що проведені обчислення були успішними. Враховуючи ненаукове призначення результатів та використання вхідних даних сейсмічної активності лише у якості тестових даних, центроїди кластерів досить точно відображають точки сейсмічної активності.

**4.2. Статистика роботи розподіленої системи**

Система була розгорнута на зовнішньому ресурсі та залучено користувачів для тестування її продуктивності, відмовостійкості та зручності у користуванні.

Отримано наступні дані роботи системи:

* Загальна кількість користувачів (рис.4.3);



*Рис.4.3. Загальна кількість користувачів*

* Кількість нових користувачів (рис. 4.4);



*Рис.4.4. Кількість нових користувачів*

* Кількість користувачів онлайн (рис. 4.5);



*Рис.4.5. Кількість користувачів онлайн*

* Кількість користувацьких сесій обчислень (рис.4.6);



*Рис. 4.6. Кількість користувацьких сесій обчислень*

* Середня тривалість користувацької сесії – 1,24 секунди;
* Середня потужність системи (рис.4.7);



*Рис.4.7. Середня потужність системи*

* Пікова потужність системи (рис.4.8)

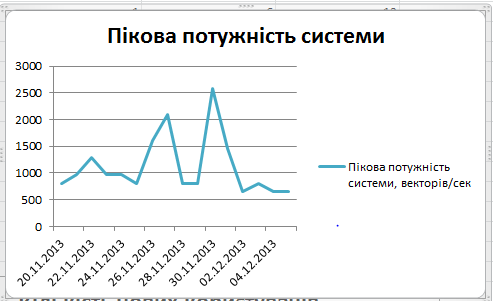
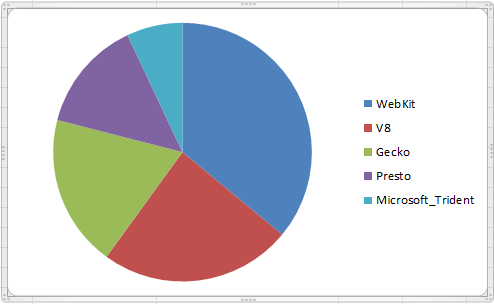


Рис.4.8. Пікова потужність системи

* Використання браузерів (браузерних двигунів) (рис.4.9)



*Рис.4.9. Використання браузерних двигунів*

**4.3. Висновки до результатів роботи розподіленої системи**

Проаналізувавши графіки потужності (середньої та пікової), можна зробити наступні висновки:

* Середня швидкість обчислень на зосередженій системі середньої конфігурації (Core i5 2,4 Ггц, 4 Гб RAM, C++) 600 векторів на секунду;
* Навіть при відносносно невеликій кількості користувачів розподіленої мережі, реалізована система є значно ефективнішою у обчисленнях, ніж зосереджена система. При збільшенні кількості користувачів ефективність буде також зростати;
* Система є доцільною при великому наборі даних, який важко (або неможливо) обчислити за короткий час на зосередженій системі;
* Втрати ефективности через комунікації між сервером та клієнтом, об’єднання результатів розподілених обчислень та виконання обчислень у інтерпретаторі повністю компенсовуються загальною кількістю користувачів системи;
* Середнє відношення кількості користувачів розподіленої мережі та однієї зосередженої системи, при якій потужність систем (швидкість обчислень) є однаковою, рівна ~2,5 розподілених користувачі на одну зосереджену систему (її конфігурація описана вище). Таке відношення обумовлене втратами ефективності через розподілений доступ до обчислень (обмін даними через мережу на відміну від обміну даними між потоками в оперативній пам’яті зосередженої системи, повільніше виконання інтерпретатора у порівнянні з компілятором, затримки при збереженні та вичитуванні з бази даних);
* Враховуючи вище сказане, можна зробити висновок, що використаний новий підхід веб-орієнтованих обчислень є ефективним засобом вирішення деякої ресурсоємкої задачі та може бути використаний у інших практичних цілях, наприклад:
  + NP-повні задачі (наприклад задачі комівояджера);
  + Побудова та навчання нейронних мереж;
  + Пошук великих послідовностей та чисел (наприклад простих чисел Мерсена);
  + Статистична обробка даних.

**РОЗДІЛ 5. ЕКОНОМІЧНА ЧАСТИНА**

**5.1. Економічна характеристика проектного рішення**

Спроектована система призначена для розподілення обчислень ресурсоємких задач між веб-клієнтами. Результатом проектування є веб-сайт, який надає наступні ресурси:

* Інформацію про проект, дані та алгоритм обчислень;
* Саме процес розподілених обчислень на веб-клієнтах;
* Статистику роботи системи як для конкретного користувача, так і загальну;
* Результати обчислень.

Система є гнучкою до змін та дозволяє легко заміняти як дані, які обробляються, так і алгоритми їх обробки.

Аналогів системи як таких не існує, більшість існуючих розподілених мереж діють за принципом сервер - десктоп-клієнт (програма), яку користувач повинен завантажити на комп’ютер, встановити та запустити процес обчислень. Ключовою перевагою спроектованої системи є відсутність будь-яких зовнішніх клієнтів, які завантажуються на машину користувача. Усі обчислення відбуваються через браузер (оглядач) при перегляді сайту.

Деяким аналогом системи можна назвати систему Bitcoin – електронну платіжну систему, у якій аналогом валюти є потужності клієнтів та об’єм задач, які вони виконали. Тому ще однією перспективою даного проекту можна назвати використання його у ролі деякого мотиватора (наприклад замість показу реклами запропонувати виконати певні корисні обчислення тощо).

Іншими його перевагами є:

* Гнучкість та масштабованість системи (для збільшення потужності мережі потрібно лише запросити більше користувачів на сайт, без додаткового збільшення серверних потужностей);
* Можливість легкого доступу у будь-якій точці світу з різних пристроїв, які мають сучасну версію оглядача (це може бути як комп’ютер, так і телефон, планшет тощо);
* Значно легше залучення нових користувачів (через веб-рекламу, надання певних бонусів тощо);

Ринковою та технічною новизною проекту є вже зазначена вище архітектура сервер - веб-клієнт.

Сфери застосування продукту можуть бути наступні:

* Наукові дослідження, обробка великих об’ємів даних (коли потрібно значні обчислювальні ресурси, а суперкомп’ютери та кластери не підходять у фінансовому плані);
* Розподілені сховища даних;
* Переведення частини серверних обов’язків на клієнтську частину.

У даному вигляді система є концептом, який при необхідності можна модифікувати під різні обчислювальні задачі та дані для обробки, де потрібна значна потужність.

**5.2. Розрахунок витрат на розроблення та впровадження проектного рішення**

**Розрахунок витрат на оплату праці**

Витрати на розроблення і впровадження програмного засобу (К) визначаються як:

(5.1)

де витрати на розроблення програмного засобу, грн..;

витрати на налагодження і дослідну експлуатацію програмного засобу на ЕОМ, грн.

Витрати на розроблення програмного засобу передбачають:

1. витрати на оплату праці розробників ();
2. відрахування в єдиний пенсійний фонд ();
3. вартість додаткових виробів що закуповуються ();
4. транспортно-заготівельні витрати ();
5. витрати на придбання спец обладнання ();
6. накладні витрати ();
7. інші витрати ();

Для проведення розрахунків витрат на оплату праці необхідно визначити категорії працівників, котрі беруть участь у проектуванні, їх чисельність, середньоденну заробітню плату спеціаліста відповідної категорії та трудомісткість робіт у людино-днях (людино-годинах).

У проекті беруть участь 3 особи: 2 розробники та тестер. Їх заробітна плата становить:

* розробник – 12000 грн. (2 ос.)
* тестер – 7992 (1 ос.).

Трудомісткість робіт над проектом становить 72 людино-днів для розробника та 72 людино-днів для тестера.

Середньоденна заробітна плата *і*-го розробника () обчислюється за формулою:

(5.2)

де основна місячна заробітна плата розробника *і*-ї спеціальності, грн.;

місячний фонд робочого часу, днів (24 дні).

Середня заробітна плата розробника становить 500 грн.

Середня заробітна плата тестера становить 333 грн

Розрахунок витрат на оплату праці всіх розробників проекту обчислюється за формулою:

(5.3)

де чисельність розробників проекту і-ї спеціальності, ос.;

час, витрачений на розроблення проекту працівником і-ї спеціальності, дні;

денна заробітна плата розробника і-ї спеціальності, грн.

Розрахунок витрат на оплату праці розробників зведено у табл. 5.1

*Таблиця 5.1**Розрахунок витрат на оплату праці*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №  зп/п | Спеціальність  Розробника | Кількість працівників, осіб | Час роботи, дні | Денна заробітна плата працівників, грн. | Витрати на оплату праці, грн. |
| 1 | Розробник | 2 | 72 | 500 | 72000 |
| 2 | Тестер | 1 | 72 | 333 | 23976 |
|  | **Разом:** | 3 | 216 | 833 | 95976 |

**Бюджет витрат на куповані товари**

Витрати на оплату праці працівникам тягнуть за собою додаткові зобов’язання підприємства перед державними фондами . Підприємство зобов’язане здійснити відрахування до Пенсійного фонду – Єдиний соціальний внесок (Згідно 2-го класу професійного ризику – 36,77 % )

3. Витрати на додаткові вироби, що закуповуються () (папір, диски тощо) визначаються за їхніми фактичними цінами з врахуванням найменування, номенклатури та необхідної їх кількості в проекті. Вихідні дані та результати розрахунків занесені в табл. 5.2. Транспортно-заготівельні витрати () становлять 10% від суми витрат на додаткові вироби, що закуповуються.

– вартість упаковки паперу формату А4.

– вартість принтера

– вартість флеш пам’яті

Розраховуємо суму витрат з урахуванням транспортно-заготівельних витрат:

*Таблиця 5.2. Розрахунок витрат на куповані товари*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| №  зп/п | Назва купованих виробів | Марка, тип | Кількість на розроблення,грн. | | Ціна за одиницю, грн.. | Сума витрат,грн. | Сума витрат з урахуванням транспортно-заготівельних витрат, грн. |
| 2 | Набір А4 | Maestro | 1 | | 31 | 31 | 34,1 |
| 2 | Принтер | Canon IP2400 | | 1 | 450 | 450 | 495 |
| **3** | Флеш-пам'ять | Transcend 8GB | | 1 | 80 | 80 | 88 |
| Разом: | | | | | | | 617,1 |

Витрати на додаткові вироби становлять 617,1 грн

**Розрахунок вартості спецобладнання**

4. Витрати на придбання спецобладнання () для проведення експериментальних робіт розраховуються в тому випадку, коли для розроблення та впровадження проектного рішення необхідне придбання додаткових технічних засобів. Вартість спецобладнання для виконання конкретного проекту визначається на основі специфікації в їх потребі та фактичних цін з урахуванням транспортно–заготівельних витрат. Транспортно-заготівельні витрати (Втр) становлять 14% від суми витрат на придбання спецобладнання.

Розраховуємо суму витрат з урахуванням транспортно-заготівельних витрат, грн. та заносимо дані в табл. 5.3.

*Таблиця 5.3. Розрахунок витрат на придбання спецобладнання*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №  зп/п | Назва обладнання | Марка, тип | Кількість на розроблення,грн. | Ціна за одиницю, грн.. | Сума витрат з урахуванням транспортно-заготівельних витрат, грн. |
| 1 | Сервер | Impression Web Server NetNavigator E 0213 | 1 | 9488 | 10436,8 |
| Разом: | | | | | 10436,8 |

Витрати на придбання спецобладнання становлять 10436, 8 грн.

**Бюджет загальновиробничих витрат**

5. Накладні витрати () проектних організацій передбачають витрати на управління, загальногосподарські, невиробничі витрати. Вони становлять 20-30% витрат на оплату праці.

Розраховуємо накладні витрати:

6. Інші витрати () – це витрати, які не враховані в попередніх статтях витрат. Їх розраховують за встановленими відсотками до витрат на оплату праці (8%).

7. Витрати на розроблення проектного рішення обчислюємо за формулою:

(5.4)

Витрати на налагодження і дослідну експлуатацію системи визначаємо згідно з формулою:

(5.5)

де – вартість однієї години роботи ПК, грн./год.;

- кількість годин роботи ПК на налагодження програми, год.

При роботі ПК споживає 0.036 КВт/год., тариф на електроенергію становить 0.28 грн., таким чином вартість однієї години роботи ПК () становить 0,01008 грн. (0.036 КВт/год.\* 0.28 грн.).

На написання та відлагодження системи було витрачено () 34 дні (34\*8 = 272год.), тому:

Результати розрахунків зведено в табл. 5.4 кошторис витрат на розробку проектного рішення.

*Таблиця 5.4. Кошторис витрат на розробку проектного рішення*

|  |  |
| --- | --- |
| Назва елементів витрат | Сума витрат, грн. |
| Витрати на розроблення проектного рішення,зокрема: | 169193,56 |
| витрати на оплату праці | 95976 |
| відрахування до єдиного пенсійного фонду | 35290,38 |
| витрати на додаткові вироби, які закуповують | 617,1 |

|  |  |
| --- | --- |
| витрати на придбання спец обладнання | 10436,8 |
| накладні витрати | 19195,2 |
| інші витрати | 7678,08 |
| Витрати на налагодження і дослідну експлуатацію системи | 2,75 |
| Разом | 169196,31 |

**5.3. Визначення комплексного показника якості**

Комплексний показник якості () визначається порівнянням показників якості проектованої системи і вибраного аналога.

За аналог обирається продукт, що відповідає проектному рішенню (проектованій системі) за сферою застосування та функціональним призначенням і є широко представленими на обраному ринку.

Для визначення використовується система показників технічного рівня і якості, яка містить такі групи:

1. Показники призначення:

* Актуальність програмного забезпечення;
* Універсальність;
* Ступінь новизни;
* Відкритість системи;
* Продуктивність роботи системи.

1. Показники надійності та безпеки:

* Тривалість функціонування системи;
* Ймовірність помилки.

1. Патентно-правові показники:

* Патентно-правовий статус.

1. Ергономічні показники:

* Зручність налаштування;
* Легкість експлуатації.

Комплексний показник якості проектованої системи визначаємо методом арифметичного середньозваженого з формули

(5.6)

де – кількість одиничних показників (параметрів), прийнятих для оцінювання якості проектованої системи;

- коефіцієнт вагомості кожного з параметрів щодо їхнього впливу на технічний рівень та якість проектованої системи (встановлюється експертним шляхом), причому:

(5.7)

- часткові показники якості, визначені порівнянням числових значень одиничних показників проектованої системи і аналога за формулами

або (5.8)

де , – кількість значень і-го одиничного показника якості відповідно проектної системи і аналога. . У якості деякого аналога даної системи була обрана розподілена система SETI@Home. Комплексний показник якості 1,365. Числове значення показників від одного (система не відповідає показнику якості) до п’яти (повністю відповідає показнику якості).

Результати розрахунків наведені в табл. 5.5.

*Таблиця 5.5. Визначення комплексного показника якості*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показники | Числове значення показників | | Відносний показник якості,  Сi | Коефіцієнт вагомості, qi | Ci x qi |
|  | Аналог | Проект. прогр. Продукт |
| Обрахунок ресурсоємких задач розподілено | 5 | 4 | 0,8 | 0,2 | 0,16 |
| Доступ до результів обчислень | 3 | 5 | 1,7 | 0,05 | 0,085 |
| Доступність вибору даних та алгоритму обчислень | 2 | 5 | 2,5 | 0,2 | 0,5 |
| Захищеність від збоїв системи | 5 | 4 | 0,8 | 0,1 | 0,08 |

*Продовження табл. 5.5*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Надійність процесу обчислень (даних і результатів) | 5 | 4 | 0,8 | 0,15 | 0,12 |
| Безпека даних клієнта | 5 | 4 | 0,8 | 0,05 | 0,04 |
| Безпека даних обчислень | 5 | 4 | 0,8 | 0,05 | 0,04 |
| Обмежена ліцензія по використанню | 3 | 5 | 1,7 | 0,05 | 0,085 |
| Зручне долучення нових клієнтів до мережі | 3 | 5 | 1,7 | 0,15 | 0,255 |
| *Разом* | 36 | 40 | 11,6 | 1,0 | 1,365 |

**5.4. Визначення експлуатаційних втрат**

Для порівняння програмних засобів в експлуатаційні витрати включають вартість підготовки даних () і вартість години роботи ПК (). Одноразові експлуатаційні витрати визначаються за формулою:

(5.9)

де - одноразові експлуатаційні витрати на проектне рішення (аналоги), грн..;

- вартість підготовки даних для експлуатації проектного рішення (аналога), грн.;

- вартість машино-години роботи ПК для проектного рішення (аналога), грн.

Річні експлуатаційні витрати визначаються за формулою

(5.10)

де – експлуатаційні річні витрати проектного рішення, грн.;

- періодичність експлуатації проектного рішення (аналога), разів/рік.

Вартість підготовки даних для експлуатації проектного рішення (аналога) () визначають за формулою

(5.11)

де *і* – номер категорій персоналу, які беруть участь у підготовці даних;

– чисельність співробітників *і* – ї категорії, ос.;

- трудомісткість роботи працівників *і*–ї категорії, ос.;

- середня денна ставка робітника *і*–ї категорії з врахуванням відрахувань до єдиного державного фонду, грн./год.

Середньогодинна ставка оператора визначається за формулою

(5.12)

де – основна місячна зарплата працівника *і* – ї категорії, грн.;

- коефіцієнт, який враховує збори до єдиного державного фонду;

- місячний фонд робочого часу, год.

Розраховуємо вартість підготовки даних для експлуатації проектного рішення:

**Розрахунок витрат на підготовку даних для роботи на ЕОМ**

Вартість машино-годин роботи ПК для проектного рішення рівна 2,75 грн. Розраховуємо одноразові експлуатаційні витрати, вони становлять

Періодичність експлуатації проектного рішення становить 5 разів/рік. Враховуючи цей параметр розраховуємо річні експлуатаційні витрати:

Вхідні дані та отримані результати заносимо в табл. 5.6.

*Таблиця 5.6. Розрахунок витрат на підготовку даних для роботи на ЕОМ*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категорія персоналу | Чисельність співробітників *і-ї категорії, ос.* | Час роботи співробітників *і*-ї категорії, год | Середньогодинна ЗП співробітника *і-*ї категорії, грн | Витрати на підготовку даних, грн. |
| Проектне рішення | | | | |
| Розробник | 2 | 40 | 85,48 | 6843,2 |
| Тестер | 1 | 32 | 56,93 | 1824 |
| Разом | 3 | 72 | 142,41 | 8667,2 |
| Аналог | | | | |
| Розробник | 3 | 16 | 100 | 4800 |
| Інженер | 3 | 40 | 150 | 18000 |
| Разом | 6 | 56 | 250 | 22800 |

**5.5. Розрахунок ціни споживання проектного рішення**

Ціна споживання () – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

(5.13)

де - ціна придбання проектного рішення, грн.;

- теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.

(5.14)

де – норматив рентабельності (13%);

- витрати на прив’язку та освоєння проектного рішення на конкретному об’єкті,

- витрати на доукомплектування технічних засобів на об’єкті,

- ставка податку на додану вартість (20%).

Цп = 169196,31 \* (1 + 0,13) \* (1 + 0,2) + 0 + 0 = 265976,6 грн (ціна придбання проектного рішення)

Теперішню вартість витрат на експлуатацію проектного рішення розраховують за формулою:

(5.15)

де - річні експлуатаційні витрати в – му році, грн.;

- строк служби проектного рішення, років;

- річна ставка проценту банків (8%).

(5.16)

ВеNPV = (43336 / (1 + 0,08)) \* 5 = 24075,5 \* 5 = 200629,63 грн (теперішня вартість витрат на експлуатацію проектного рішення на термін 5 років з процентною ставкою 8%)

Розраховуємо ціну споживання проектного рішення:

Цс (п) = 265976,6 + 200629,63 = 466606,23 грн (ціна споживання проектного рішення)

Визначаємо ціну споживання для аналогу. Визначаємо теперішню вартість витрат на експлуатацію аналогу. Термін експлуатації аналогу становить 5 років, тоді . Оскільки , то:

ВеNPV(а) = (114000 / (1 + 0,08)) \* 5 = 615600 грн (теперішня вартість витрат на експлуатацію аналога)

Тепер розраховуємо ціну споживання проекту аналогу:

Цс(а) = 300000 + 615600 = 915600 грн (ціна спроживання аналога)

B(e)П = 8667,2 \* 5 = 43336 грн (річні експлуатаційні витрати для періодичності експлуатації проектного рішення 5 разів на рік)

B(e)П (А) =22800 \* 5 = 114000 грн (річні експлуатаційні витрати для аналога)

Цп(а)= 300000 грн (ціна придбання проектного рішення – аналога)

**5.6. Визначення показників економічної ефективності**

Зважаючи на те, що базою для порівняння було обрано відповідний програмний засіб, то в даному розділі розраховуються такі показники:

1.Показник конкурентоспроможності

(5.17)

2. Економічний ефект у сфері експлуатації (грн.)

(5.18)

Еекс = 114000 – 43336 = 70664 грн.

3. Економічний ефект у сфері проектування (грн.)

(5.19)

Розраховується якщо та :

4. Додатковий економічний ефект у сфері експлуатації (грн.)

(5.20)

Еексд = 96137,59 + 89016,288768 + 82422,4896 + 76317,12 + 70664 = =414557,49грн

5. Додатковий економічний ефект у сфері проектування (грн.)

(5.6.5)

Епрд = 110349,98 грн (додатковий економічний ефект у сфері проектування)

6.Термін окупності витрат на проектування рішення (років)

(5.21)

Пя = 1,365 (комплексний показник якості)

*Таблиця 5.7. Показники економічної ефективності*

|  |  |  |  |
| --- | --- | --- | --- |
| Назва показників | Одиниці вимірювання | Значення показників | |
| Аналог | Проектне рішення |
| 1. Капітальні вкладення | грн. |  | 265976,6 |
| 1. Ціна придбання | грн. | 300000 | 265976,6 |
| 1. Річні експлуатаційні витрати | грн. | 114000 | 43336 |

*Продовження табл.5.7*

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Ціна споживання | грн. | 915600 | 466606,23 |
| 1. Економічний ефект у сфері експлуатації | грн. |  | 34023,4 |
| 1. Додатковий економічний ефект у сфері експлуатації | грн. |  | 414557,49 |
| 1. Економічний ефект у сфері проектування | грн. |  | 75102,34 |
| 1. Додатковий економічний ефект у сфері проектування | грн. |  | 110349,98 |
| 1. Термін окупності витрат за проектування рішення | грн. |  | 0,61 |
| 1. Коефіцієнт конкурентоспроможності | грн. |  | 0,69 |

**5.7. Висновки економічної частини**

У даній роботі було проведено аналіз економічних показників проектованої системи. У якості деякого аналога даної системи була обрана розподілена система SETI@home, яка працює за подібним принципом.

З отриманих результатів можна зробити висновок, що спроектована система є конкурентно спроможною відносно аналогів (коефіцієнт конкурентоспроможності = 0,69, що є хорошим показником у області IT).

Проектування системи себе окупить через 0,61 року (~7 місяців) після запуску в експлуатацію.

Ціна придбання та ціна споживання також відрізняється в меншу сторону у порівнянні з аналогом:

* ціна придбання проектного рішення 265976,6 грн проти 300000 грн у аналога;
* ціна споживання проектного рішення 466606,23 грн проти 915600 грн у аналога).

Отже економічний ефект у сфері експлуатації становить 34023,4 грн, у сфері проектування 75102,34, що говорить на користь даного проектного рішення.

Додатковий економічний ефект у сфері експлуатації становить 414557,49 грн, у сфері проектування - 110349,98 грн.

Область розподілених обчислень, в тому числі комерційних, є недостатньо розвинутою на ринку, тому додатковою перевагою буде новизна проекту, його ідеї.­­

З наведених вище показників можна зробити висновок, що проектне рішення є вигідним та конкурентоспроможним у порівнянні з аналогами.

**ВИСНОВКИ**

# У результаті виконання магістерського дослідження було створено розподілену систему кластеризації даних, яка базована на ресурсах веб-клієнтів та було досягнуто наступні цілі:

* Розроблено веб-ресурс, який надає такий функціонал:
  + Інформацію про проект розподілених обчислень;
  + Реєстрацію користувачів;
  + Обчислення в браузері користувача;
  + Статистика обчислень (для конкретного користувача та в загальному системи);
  + Візуалізація результатів кластеризації.
* Реалізована основна мета даного проекту – новий підхід до розподілених обчислень, використання веб-браузера у якості основного обчислювального вузла;
* Створена система є легко масштабована та гнучка до змін, що дозволяє пристосовувати її до різних типів задач;
* Обчислення можуть проводитися з будь-якої точки світу через інтернет, використовуючи як комп’ютер, так і інші персональні цифрові пристрої, які мають влаштований сучасний браузер (планшет, телефон тощо);
* Доведена практична користь системи, проведено порівняння швидкості обчислень із обчисленнями на зосередженій системі, яке показало ефективність роботи системи (більшу потужність) при кількості обчислювальних клієнтів більше трьох;
* Проведений аналіз результатів кластеризації даних, який показав відповідність результатів очікуваним. Порівняння відбувалося як візуальним (за допомогою порівняння центрів отриманих кластерів з картою сейсмічних коливань, отриманої з тематичного ресурсу), так і аналітичним способом (за допомогою перегляду статистики сейсмічних коливань по конкретних територіях);
* Зібрано статистику роботи системи (кількість користувачів, користувацьких сесій, відсоток використання різних браузерів).

Перспективи системи можуть бути наступними:

* Використання модифікації системи для реалізації проектів добровільних розподілених обчислень з метою розв’язання певної ресурсоємкої задачі;
* Комерційне використання системи, коли користувачі зможуть надавати свої обчислювальні ресурси за потребою, а взамін отримувати певні бонуси.

Новизна магістерського дослідження була досягнути новим підходом до реалізації розподілених обчислень, завдяки використанню незадіяних обчислювальних ресурсів користувачів мережі інтернет.

Враховуючи вищесказані аспекти системи, можна вважати, що мета магістерського дослідження була досягнута; проведена робота відповідає критеріям завдання.

ПЕРЕЛІК ПОСИЛАНЬ

* + - 1. BOINC: Open-source software for volunteer computing and grid computing [Electronic resource]. – Mode of access: http://boinc.berkeley.edu/. Last access: 2013. – Title from the screen.
      2. Top 500 supercomputer sites [Electronic resource]. – Mode of access: http://www.top500.org/. Last access: 2013. – Title from the screen.
      3. Розподілені обчислення в Україні | Распределенные вычисления в Украине. [Електронний ресурс]. – Режим доступу: http://distributed.org.ua. – Назва з екрану.
      4. Jerzy Duda, Wojciech Dłubacz: Distributed Evolutionary Computing System Based on Web Browsers with JavaScript [Text] // Applied Parallel and Scientific Computing // 11th International Conference, PARA 2012, Helsinki, Finland, June 10-13, 2012, Revised Selected Papers. – 2013. – Part III. – pp. 183-191.
      5. Shinjo, Y. ; Dept. of Comput. Sci., Univ. of Tsukuba, Tsukuba, Japan ; Fei Guo ; Kaneko, N. ; Matsuyama, T.: A distributed web browser as a platform for running collaborative applications [Text] // Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on. – 2011/ - pp. 278 – 286.
      6. Эндрю Таненбаум, Мартин ван Стеен: Распределенные системы. Принципы и парадигмы = Andrew S. Tanenbaum, Maarten van Steen. "Destributed systems. Principles and paradigms". – Санкт-Петербург: Питер, 2003. – 877 с. – ISBN 5-272-00053-6.
      7. Ghosh, Sukumar (2007), Distributed Systems: An Algorithmic Approach, Chapman & Hall/CRC, ISBN 978-1-58488-564-1.
      8. Distributed Computing: Active Projects [Electronic resource]. – Mode of access: http://distributedcomputing.info/. Last access: 2013. – Title from the screen.
      9. Олдендерфер М. С., Блэшфилд Р. К.: Кластерный анализ / Факторный, дискриминантный и кластерный анализ: пер. с англ.; Под. ред. И. С. Енюкова. — М.: Финансы и статистика, 1989. — 215 с.
      10. Arthur, D. and Vassilvitskii, S. (2007). "k-means++: the advantages of careful seeding". Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035.
      11. National Geophysical Data Center / World Data Service (NGDC/WDS) Significant Earthquake Database, Boulder, CO, USA. [Electronic resource]. – Mode of access: http://www.ngdc.noaa.gov/nndc/struts/form?t=101650&s=1&d=1. Last access: 2013. – Title from the screen.
      12. Шкала Медведева-Шпонхойера-Карника (MSK-64). [Електронний ресурс]. – Режим доступу: http://pts.mi-perm.ru/region/MSK-64.pdf. – Назва з екрану.
      13. Andriy Babets: jsDSTR - Javascript Distributed Clusterization System. [Електронний ресурс]. – Режим доступу: https://github.com/tesss/jsdstr. – Назва з екрану.
      14. IRIS Seismic Monitor. [Electronic resource]. – Mode of access: http://www.iris.edu/seismon/. Last access: 2013. – Title from the screen.
      15. Топорков В.В. [Текст] : Модели распределенных вычислений. – ФИЗМАТЛИТ, 2004. – 320с.
      16. Дюран Б. Оделл П. : Кластерный анализ. – Статистика, 1977. – 277с.
      17. [Владимир Дронов](http://www.ozon.ru/person/352088/) [Текст] : HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов // Профессиональное программирование. - [БХВ-Петербург](http://www.ozon.ru/brand/1098685/), 2011. – 416с.
      18. Адам Фримен [Текст] : ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов ; перевод Ю. Артеменко // Expert's Voice. – Вильямс, 2013. – 688c.
      19. Alexei White [Text] : Major JavaScript Engines // JavaScript Programmer's Reference. — Indianapolis, IN 46256: Wiley Publishing, Inc., 2009. — P. 12 — 13. — (Programmer's Reference). — ISBN 978-0-470-34472-9.
      20. Кристиан Нейгел и др. [Текст] : C# 5.0 и платформа .NET 4.5 для профессионалов = Professional C# 5.0 and .NET 4.5. — М.: «Диалектика», 2013. — 1440 с. — ISBN 978-5-8459-1850-5.
      21. Герберт Шилдт. [Текст] : C# 4.0: полное руководство = C# 4.0 The Complete Reference. — М.: «Вильямс», 2010. — С. 1056. — ISBN 978-5-8459-1684-6.
      22. Э. Фримен, Э. Фримен [Текст] : Изучаем HTML, XHTML и CSS = Head First HTML with CSS & XHTML. — П.: «Питер», 2010. — 656 с. — ISBN 978-5-49807-113-8.

ДОДАТКИ

Додаток А

Вихідні коди

*Структура бази даних проекту*

use [jsDSTR];

go;

create table [dbo].[Settings] -- налаштування

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

[Key] nvarchar(max) not null,

[Value] nvarchar(max) not null

)

create table dbo.Session -- користувацькі сесії обчислень

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

Guid uniqueidentifier not null,

UserName nvarchar(max) not null,

State int not null,

StateMessage nvarchar(max) null,

CalculationId int not null

)

create table [dbo].[Log] -- логування помилок та інформації

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

Message nvarchar(max) not null,

UserName nvarchar(max) null,

Context nvarchar(max) null,

Type int not null

);

create table [dbo].Earthquake -- дані для кластеризації

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

CalculationId int default 0,

I\_D int not null,

FLAG\_TSUNAMI nvarchar(3) null,

YEAR int null,

MONTH int null,

DAY int null,

HOUR int null,

MINUTE int null,

SECOND decimal(3,1) null,

FOCAL\_DEPTH int null,

EQ\_MAG\_MW decimal(3,1) null,

EQ\_MAG\_MS decimal(3,1) null,

EQ\_MAG\_MB decimal(3,1) null,

EQ\_MAG\_ML decimal(3,1) null,

EQ\_MAG\_MFA decimal(3,1) null,

EQ\_MAG\_UNK decimal(3,1) null,

INTENSITY int null,

COUNTRY nvarchar(100) null,

STATE nvarchar(10) null,

LOCATION\_NAME nvarchar(100) null,

LATITUDE decimal(6,3) null,

LONGITUDE decimal(6,3) null,

REGION\_CODE int null,

DEATHS int null,

DEATHS\_DESCRIPTION int null,

MISSING int null,

MISSING\_DESCRIPTION int null,

INJURIES int null,

INJURIES\_DESCRIPTION int null,

DAMAGE\_MILLIONS\_DOLLARS decimal(9,3) null,

DAMAGE\_DESCRIPTION int null,

HOUSES\_DESTROYED int null,

HOUSES\_DESTROYED\_DESCRIPTION int null,

HOUSES\_DAMAGED int null,

HOUSES\_DAMAGED\_DESCRIPTION int null,

TOTAL\_DEATHS int null,

TOTAL\_DEATHS\_DESCRIPTION int null,

TOTAL\_MISSING int null,

TOTAL\_MISSING\_DESCRIPTION int null,

TOTAL\_INJURIES int null,

TOTAL\_INJURIES\_DESCRIPTION int null,

TOTAL\_DAMAGE\_MILLIONS\_DOLLARS decimal(9,3) null,

TOTAL\_DAMAGE\_DESCRIPTION int null,

TOTAL\_HOUSES\_DESTROYED int null,

TOTAL\_HOUSES\_DESTROYED\_DESCRIPTION int null,

TOTAL\_HOUSES\_DAMAGED int null,

TOTAL\_HOUSES\_DAMAGED\_DESCRIPTION int null,

);

create table KmeansCalculation -- обчислення

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

K int not null,

Iteration int not null,

MaxIterations int not null,

State int not null,

StateMessage nvarchar(max) null

);

create table Centroid -- збережені центроїди

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

V1 decimal(6,3) not null,

V2 decimal(6,3) not null,

V3 decimal(6,3) not null,

CalculationId int not null,

Iteration int not null,

Committed bit not null

);

create table CentroidAssignment -- збережені прив’язки

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

CentroidId int null,

VectorId int null,

CalculationId int not null,

Iteration int not null,

);

create table Task -- збережені користувацькі завдання

(

Id int identity(1,1) primary key,

CreatedDate datetime default getdate(),

ChangedDate datetime default getdate(),

State int not null,

Type int not null,

SessionGuid uniqueidentifier null,

CalculationId int not null,

Iteration int not null,

SlotStart int not null,

SlotCapacity int not null

);

*Вихідний код алгоритму кластеризації k-means для розподіленої системи (KmeansCalculationService.cs):*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.UI;

using JSDstr.Helpers;

using JSDstr.Interfaces;

using JSDstr.Models;

using JSDstr.Repositories;

using JSDstr.ViewObjects;

/\*

\* Enter points:

\* -> GetTask

\* - Start

\* |

\* StartMainLoop

\* |

\* StartMainStep

\* |

\* StartAssignmentLoop

\* |

\* - StartAssignmentStep -> AssignmentTask

\* - StartUpdateCentroidsStep -> UpdateCentroidsTask

\* - Completed -> CompletedTask

\* - Failed -> FailedTask

\* - Error -> ErrorTask

\* -> CompleteTask

\* - CompleteAssignmentStep -> SuccessfulTask

\* |

\* CompleteAssignmentLoop

\* |

\* StartUpdateCentroidsLoop -> SuccessfulTask

\* - CompleteUpdateCentroidsStep -> SuccessfulTask

\* |

\* CompleteUpdateCentroidsLoop

\* |

\* CompleteMainStep\_\_\_\_\_\_\_\_\_

\* | |

\* CompleteMainLoop StartMainStep

\* | |

\* - Complete -> StartAssignmentLoop -> SuccessfulTask

\* - Failed -> FailedTask

\* - Error -> ErrorTask

\* -> CancelTask

\* - CancelTask -> SuccessfulTask

\* -Error

\*/

namespace JSDstr.Services

{

public class KmeansService : Singleton<KmeansService>, ICalculationService

{

private KmeansService(){}

#region fields

private readonly IRepository<KmeansCalculation> \_calculationRepository = new SqlRepository<KmeansCalculation>();

private readonly IRepository<Earthquake> \_earthquakeRepository = new SqlRepository<Earthquake>();

private readonly IRepository<Centroid> \_centroidRepository = new SqlRepository<Centroid>();

private readonly IRepository<CentroidAssignment> \_assignmentRepository = new SqlRepository<CentroidAssignment>();

private readonly IRepository<Task> \_taskRepository = new SqlRepository<Task>();

private readonly IRepository<Session> \_sessionRepository = new SqlRepository<Session>();

private readonly ISettingsService \_settingsService = new SettingsService();

private KmeansCalculation \_calculation;

private VectorViewObject[] \_vectors;

private VectorViewObject[] \_centroids;

private VectorViewObject[] \_newCentroids;

private AssignmentViewObject[] \_assignments;

private Task[] \_tasks;

private CalculationTaskViewObject \_calculationTask;

private int N { get { return \_vectors.Length; } }

private int K { get { return \_calculation.K; } }

private int MaxIterations { get { return \_calculation.MaxIterations; } }

private readonly object \_locker = new object();

private int \_assignmentsSlotCapacity;

private int \_updateCentroidsSlotCapacity;

private CalculationTaskViewObject SuccessfulCalculationTask

{

get

{

return new CalculationTaskViewObject(\_calculation)

{

State = CalculationState.Successful,

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

};

}

}

#endregion

#region interface

public CalculationTaskViewObject GetTask(CalculationTaskViewObject calculationTask)

{

lock (\_locker)

{

try

{

Init(calculationTask, true);

switch (\_calculation.State)

{

case CalculationState.Started:

return Start();

case CalculationState.AssignmentLoop:

return StartAssignmentStep();

case CalculationState.UpdateCentroidsLoop:

return StartUpdateCentroidsStep();

case CalculationState.Completed:

return Completed();

case CalculationState.Failed:

return Failed("Calculation failed");

default:

return Error("Invalid calculation state");

}

}

catch (CalculationFailedException ex)

{

return HandleCalculationFailed(ex);

}

catch (Exception ex)

{

return Error(ex.Message);

}

}

}

public CalculationTaskViewObject CompleteTask(CalculationTaskViewObject calculationTask)

{

lock (\_locker)

{

try

{

Init(calculationTask);

switch (\_calculation.State)

{

case CalculationState.AssignmentLoop:

return CompleteAssignmentStep();

case CalculationState.UpdateCentroidsLoop:

return CompleteUpdateCentroidsStep();

case CalculationState.Failed:

return Failed("Calculation failed");

default:

return Error("Invalid calculation state");

}

}

catch (CalculationFailedException ex)

{

return HandleCalculationFailed(ex);

}

catch (Exception ex)

{

return Error(ex.Message);

}

}

}

public CalculationTaskViewObject CancelTask(CalculationTaskViewObject calculationTask)

{

lock(\_locker)

{

try

{

Init(calculationTask);

if (\_calculation.State == CalculationState.AssignmentLoop ||

\_calculation.State == CalculationState.UpdateCentroidsLoop)

{

GetTasks();

var sessionTasks = \_tasks.Where(x =>

x.SessionGuid == \_calculationTask.SessionGuid && x.State == TaskState.Started).ToArray();

if (sessionTasks.Any())

{

foreach (var sessionTask in sessionTasks)

{

sessionTask.State = TaskState.Cancelled;

sessionTask.SessionGuid = null;

}

\_taskRepository.Save(sessionTasks);

}

return SuccessfulCalculationTask;

}

return Error("Invalid calculation state for CancelTask");

}

catch (CalculationFailedException ex)

{

return HandleCalculationFailed(ex);

}

catch (Exception ex)

{

return Error(ex.Message);

}

}

}

#endregion

#region steps

private CalculationTaskViewObject Start()

{

\_centroids = GetRandomVectors();

SaveCentroids(true, true);

\_assignments = GetEmptyAssignments();

return StartMainLoop();

}

private CalculationTaskViewObject StartMainLoop()

{

return StartMainStep(true);

}

private CalculationTaskViewObject StartMainStep(bool getTask)

{

\_calculation.Iteration++;

\_calculationRepository.Save(\_calculation);

SaveCentroids(true, true);

//SaveAssignments(true);

return StartAssignmentLoop(getTask);

}

private CalculationTaskViewObject StartAssignmentLoop(bool getTask)

{

\_calculation.State = CalculationState.AssignmentLoop;

\_calculationRepository.Save(\_calculation);

return getTask ? StartAssignmentStep() : SuccessfulCalculationTask;

}

private CalculationTaskViewObject StartAssignmentStep()

{

return GetNextTask();

}

private CalculationTaskViewObject CompleteAssignmentStep()

{

MergeAssignments(\_calculationTask.Assignments);

CompleteSessionTasks();

return GetNextStep();

}

private CalculationTaskViewObject CompleteAssignmentLoop()

{

return StartUpdateCentroidsLoop();

}

private CalculationTaskViewObject StartUpdateCentroidsLoop()

{

\_calculation.State = CalculationState.UpdateCentroidsLoop;

\_calculationRepository.Save(\_calculation);

SaveCentroids(false, true);

return SuccessfulCalculationTask;

}

private CalculationTaskViewObject StartUpdateCentroidsStep()

{

return GetNextTask();

}

private CalculationTaskViewObject CompleteUpdateCentroidsStep()

{

if(\_calculationTask.Centroids.Length != K)

throw new InvalidOperationException("UpdateCentroids: results length != K");

for (int i = 0; i < \_calculationTask.Centroids.Length; i++)

{

var centroid = \_calculationTask.Centroids[i];

if (centroid == null)

continue;

\_newCentroids[i] += centroid;

}

SaveCentroids(false, false);

CompleteSessionTasks();

return GetNextStep();

}

private CalculationTaskViewObject CompleteUpdateCentroidsLoop()

{

var nextIteration = false;

for (var i = 0; i < \_newCentroids.Length; i++)

{

var clusterSize = \_assignments.Count(x => x.C == i);

if (clusterSize == 0)

clusterSize = 1;

\_newCentroids[i] \*= (decimal)(1.0 / clusterSize);

if (!Equals(\_newCentroids[i], \_centroids[i]))

nextIteration = true;

\_centroids[i] = \_centroids[i].CopyFrom(\_newCentroids[i]);

}

SaveCentroids(false, false);

SaveCentroids(true, false);

return CompleteMainStep(nextIteration);

}

private CalculationTaskViewObject CompleteMainStep(bool nextIteration)

{

if (!nextIteration)

{

return CompleteMainLoop(false);

}

if (\_calculation.Iteration == MaxIterations)

{

return CompleteMainLoop(true);

}

return StartMainStep(false);

}

private CalculationTaskViewObject CompleteMainLoop(bool isMaxIterations)

{

return Complete(isMaxIterations);

}

private CalculationTaskViewObject Complete(bool isMaxIterations)

{

\_calculation.StateMessage = isMaxIterations ?

string.Format("Max iteration: {0}", MaxIterations) :

"Not difference between iterations";

\_calculation.State = CalculationState.Completed;

\_calculationRepository.Save(\_calculation);

return new CalculationTaskViewObject(\_calculation)

{

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

};

}

private CalculationTaskViewObject Completed()

{

return new CalculationTaskViewObject(\_calculation)

{

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

};

}

private CalculationTaskViewObject Failed(string message)

{

if (\_calculation != null)

{

\_calculation.State = CalculationState.Failed;

\_calculation.StateMessage = message;

\_calculationRepository.Save(\_calculation);

\_calculationTask = new CalculationTaskViewObject(\_calculation)

{

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

};

}

else

{

\_calculationTask = Error(message);

}

ClearCalculation();

return \_calculationTask;

}

private CalculationTaskViewObject Error(string error)

{

return \_calculation != null

? new CalculationTaskViewObject(\_calculation)

{

State = CalculationState.Error,

StateMessage = error,

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

}

: new CalculationTaskViewObject

{

State = CalculationState.Error,

StateMessage = error,

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

};

}

#endregion

#region init

private void Init(CalculationTaskViewObject calculationTask, bool fromGet = false)

{

\_calculationTask = calculationTask;

\_assignmentsSlotCapacity = \_settingsService.AssignmentsSlotCapacity;

\_updateCentroidsSlotCapacity = \_settingsService.UpdateCentroidsSlotCapacity;

InitCalculation(fromGet);

if (\_calculation.State != CalculationState.Completed && \_calculation.State != CalculationState.Failed)

{

InitVectors();

InitCentroids();

InitAssignments();

InitTasks();

}

}

private void InitCalculation(bool fromGet)

{

var currentCalculationId = \_settingsService.CurrentCalculationId;

if (\_calculation == null || \_calculation.Id != currentCalculationId)

{

if (currentCalculationId == 0)

{

if(!fromGet)

throw new Exception("Init create wasn't performed from GetTask");

\_calculation = new KmeansCalculation

{

Iteration = 0,

K = \_settingsService.KmeansK,

MaxIterations = \_settingsService.MaxIterations,

State = CalculationState.Started

};

\_calculation = \_calculationRepository.Insert(\_calculation);

\_settingsService.CurrentCalculationId = \_calculation.Id;

}

else

{

\_calculation = \_calculationRepository.Entities.SingleOrDefault(x => x.Id == currentCalculationId);

if (\_calculation == null)

{

throw new CalculationFailedException("Calculation is null");

}

}

}

}

private void InitVectors()

{

if (\_vectors == null)

{

var source = \_earthquakeRepository.Entities.Where(

x => x.Latitude.HasValue && x.Longitude.HasValue && x.Intensity.HasValue &&

x.CalculationId == \_calculation.Id);

if (!source.Any())

source = \_earthquakeRepository.Entities.Where(

x => x.Latitude.HasValue && x.Longitude.HasValue && x.Intensity.HasValue &&

x.CalculationId == 0);

if (!source.Any())

throw new CalculationFailedException("Vectors are empty");

\_vectors = source.Select(x => new VectorViewObject(x)).ToArray();

}

}

private void InitCentroids()

{

if (\_centroids == null)

{

if (\_calculation.State == CalculationState.AssignmentLoop)

{

GetCentroids(true);

}

else if (\_calculation.State == CalculationState.UpdateCentroidsLoop)

{

GetCentroids(true);

GetCentroids(false);

}

}

}

private void InitAssignments()

{

if (\_assignments == null)

{

if(\_calculation.State == CalculationState.UpdateCentroidsLoop)

{

GetAssignments();

}

}

}

private void InitTasks()

{

if (\_tasks == null)

{

if (\_calculation.State == CalculationState.AssignmentLoop || \_calculation.State == CalculationState.UpdateCentroidsLoop)

{

TerminateUnassignedTasks();

}

}

}

#endregion init

#region get-save-handle

// db => source, viewobjects

private void GetCentroids(bool committed)

{

var source = \_centroidRepository.Entities.Where(x =>

x.CalculationId == \_calculation.Id &&

x.Iteration == \_calculation.Iteration &&

x.Committed == committed);

var c = source.Count();

if (c != K)

throw new CalculationFailedException("Invalid number of centroids for current calculation");

if (committed)

{

\_centroids = source.Select(x => new VectorViewObject(x)).ToArray();

}

else

{

\_newCentroids = source.Select(x => new VectorViewObject(x)).ToArray();

}

}

private Centroid GetCentroidFromViewObject(VectorViewObject viewObject, bool committed)

{

return new Centroid

{

Id = viewObject.Id,

V1 = viewObject.V1,

V2 = viewObject.V2,

V3 = viewObject.V3,

Committed = committed,

CalculationId = \_calculation.Id,

Iteration = \_calculation.Iteration

};

}

// viewobjects => db, source

private void SaveCentroids(bool committed, bool createNew)

{

if (!createNew)

{

\_centroidRepository.Save(\_centroids.Select(x => GetCentroidFromViewObject(x, committed)));

return;

}

var source = \_centroids.Select(x => GetCentroidFromViewObject(x, committed)).AsQueryable();

source = \_centroidRepository.Insert(source);

if (committed)

{

\_centroids = source.Select(x => new VectorViewObject(x)).ToArray();

}

else

{

\_newCentroids = source.Select(x => new VectorViewObject(x)).ToArray();

}

}

private VectorViewObject[] GetRandomVectors()

{

var n = N;

var k = K;

var selectedVectors = new VectorViewObject[k];

var selectedIndexes = new int[k];

var testedIndexes = new HashSet<int>();

var tested = 0;

var selected = 0;

var rand = new Random();

while (selected < k)

{

if (tested == n)

throw new InvalidOperationException("tested == n");

var randomIndex = rand.Next(0, n - 1);

if (testedIndexes.Contains(randomIndex))

continue;

testedIndexes.Add(randomIndex);

tested++;

var vector = \_vectors[randomIndex];

var select = true;

int i;

for (i = 0; i < selected; i++)

{

if (vector.Equals(selectedVectors[i]))

{

select = false;

break;

}

}

if (select)

{

selectedVectors[selected] = vector;

selectedIndexes[selected] = randomIndex;

selected++;

}

}

return selectedVectors;

}

// db => source, viewobjects

private void GetAssignments()

{

var source = \_assignmentRepository.Entities.Where(x =>

x.CalculationId == \_calculation.Id &&

x.Iteration == \_calculation.Iteration);

var c = source.Count();

if (c > N)

throw new CalculationFailedException("Invalid number of assignments for current calculation");

\_assignments = source.Select(x => new AssignmentViewObject(x)).ToArray();

}

private CentroidAssignment GetAssignmentFromViewObject(AssignmentViewObject viewObject)

{

var calculationId = \_calculation.Id;

var iteration = \_calculation.Iteration;

return new CentroidAssignment

{

Id = viewObject.Id,

CentroidId = viewObject.C,

Vectorid = viewObject.V,

CalculationId = calculationId,

Iteration = iteration

};

}

private void MergeAssignments(AssignmentViewObject[] mergeWith)

{

if (mergeWith.Length > \_assignmentsSlotCapacity)

throw new InvalidOperationException("MergeAssignments: results length != SlotCapacity");

\_assignmentRepository.Insert(mergeWith.Select(GetAssignmentFromViewObject));

}

private AssignmentViewObject[] GetEmptyAssignments()

{

var assignments = new AssignmentViewObject[N];

for (var i = 0; i < N; i++)

assignments[i] = new AssignmentViewObject()

{

V = i

};

return assignments;

}

// db => source, viewobjects

private void GetTasks()

{

var tasksSource = \_taskRepository.Entities.Where(x =>

x.Type == \_calculation.State &&

x.CalculationId == \_calculation.Id &&

x.Iteration == \_calculation.Iteration);

\_tasks = tasksSource.ToArray();

}

private void TerminateUnassignedTasks()

{

var tasks = \_taskRepository.Entities.Where(x =>

x.CalculationId == \_calculation.Id &&

x.Iteration == \_calculation.Iteration &&

x.Type == \_calculation.State &&

x.State == TaskState.Started).ToArray();

var sessions = \_sessionRepository.Entities.Where(x =>

x.CalculationId == \_calculation.Id &&

x.State != SessionState.Started).ToArray();

var unassignedTasks = tasks.Join(sessions,

t => t.SessionGuid,s => s.Guid, (t, s) => t).ToArray();

foreach (var unassignedTask in unassignedTasks)

{

unassignedTask.SessionGuid = null;

unassignedTask.State = TaskState.Cancelled;

}

\_taskRepository.Save(unassignedTasks);

}

private TaskState[] GetTaskPlan()

{

GetTasks();

var taskPlan = new KeyValuePair<TaskState, DateTime>[N];

foreach (var task in \_tasks)

{

var slotStart = task.SlotStart;

var slotCapacity = task.SlotCapacity;

var state = task.State;

var changedDate = task.ChangedDate;

for (var j = 0; j < slotCapacity && slotStart + j < N; j++)

{

var k = slotStart + j;

if (taskPlan[k].Value == default(DateTime) || taskPlan[k].Value < changedDate)

taskPlan[k] = new KeyValuePair<TaskState, DateTime>(state, changedDate);

}

}

return taskPlan.Select(x => x.Key).ToArray();

}

private CalculationTaskViewObject GetNextTask()

{

var taskPlan = GetTaskPlan();

if (taskPlan.All(x => x == TaskState.Completed))

{

//return Error("All tasks are completed, but next iteration isn't started");

if (\_calculation.State == CalculationState.AssignmentLoop)

return CompleteAssignmentLoop();

if (\_calculation.State == CalculationState.UpdateCentroidsLoop)

return CompleteUpdateCentroidsLoop();

return Error("Invalid calculation state");

}

var slotCapacity = \_calculation.State == CalculationState.AssignmentLoop

? \_assignmentsSlotCapacity

: \_updateCentroidsSlotCapacity;

var taskIndex = -1;

for (var i = 0; i < taskPlan.Length; i++)

{

if (taskPlan[i] == TaskState.Idle || taskPlan[i] == TaskState.Cancelled)

{

taskIndex = i;

break;

}

}

if (taskIndex >= 0)

{

var task = new Task

{

CalculationId = \_calculation.Id,

Iteration = \_calculation.Iteration,

SessionGuid = \_calculationTask.SessionGuid,

SlotStart = taskIndex,

SlotCapacity = slotCapacity,

State = TaskState.Started,

Type = \_calculation.State

};

\_taskRepository.Insert(task);

AssignmentViewObject[] taskAssignments = null;

if (\_calculation.State == CalculationState.UpdateCentroidsLoop)

taskAssignments = \_assignments.Skip(taskIndex).Take(slotCapacity).ToArray();

\_calculationTask = new CalculationTaskViewObject(\_calculation)

{

Vectors = !\_calculationTask.VectorsCached ? \_vectors : null,

Centroids = \_calculation.State == CalculationState.AssignmentLoop ? \_centroids : \_newCentroids,

SlotStart = taskIndex,

SlotCapacity = slotCapacity,

Assignments = taskAssignments,

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

};

return \_calculationTask;

}

\_calculationTask = new CalculationTaskViewObject(\_calculation)

{

State = CalculationState.Busy,

SessionGuid = \_calculationTask.SessionGuid,

VectorsCached = \_calculationTask.VectorsCached

};

return \_calculationTask;

}

private void CompleteSessionTasks()

{

GetTasks();

var completedTask = \_tasks.FirstOrDefault(x => x.SessionGuid == \_calculationTask.SessionGuid);

if(completedTask != null)

{

completedTask.State = TaskState.Completed;

}

\_taskRepository.Save(completedTask);

}

private CalculationTaskViewObject GetNextStep()

{

var taskPlan = GetTaskPlan();

if (taskPlan.All(x => x == TaskState.Completed))

{

if (\_calculation.State == CalculationState.AssignmentLoop)

return CompleteAssignmentLoop();

if (\_calculation.State == CalculationState.UpdateCentroidsLoop)

return CompleteUpdateCentroidsLoop();

return Error("Invalid calculation state");

}

return SuccessfulCalculationTask;

}

private void ClearCalculation()

{

\_settingsService.CurrentCalculationId = 0;

\_calculation = null;

\_vectors = null;

\_centroids = null;

\_assignments = null;

}

private CalculationTaskViewObject HandleCalculationFailed(CalculationFailedException ex)

{

LogService.Log(ex);

return Failed(ex.Message);

}

#endregion get-save

}

public class CalculationFailedException : Exception

{

public CalculationFailedException(string message) : base(message){}

}

}

*Вихідний код клієнтських обчислень алгоритму k-means (частина файлу script.js):*

var Kmeans = function (task) {

if (!$.isPlainObject(task))

throw "Kmeans: task is null";

if (task.State != Processing.calculationState.AssignmentLoop && task.State != Processing.calculationState.UpdateCentroidsLoop)

throw "Kmeans: invalid task state";

if (task.VectorsCached) {

var vectorsJson = helpers.getFromStorage(vectorsStoreKey);

if (vectorsJson != null) {

task.Vectors = JSON.parse(vectorsJson);

} else {

task.VectorsCached = false;

throw "Kmeans: no vectors in cache";

}

} else if (task.Vectors == null) {

throw "Kmeans: vectors is null";

} else {

if (helpers.saveToStorage(vectorsStoreKey, JSON.stringify(task.Vectors)) != null)

task.VectorsCached = true;

}

if (Number(task.SlotStart) == NaN || task.SlotStart < 0)

throw "Kmeans: slotstart is invalid";

if (Number(task.SlotCapacity) == NaN || task.SlotCapacity <= 0)

throw "Kmeans: slotcapacity is invalid";

if (Number(task.K) == NaN || task.K <= 0 || task.K > task.Vectors.length)

throw "Kmeans: k is invalid";

if (task.State == Processing.calculationState.AssignmentLoop) {

if (task.Centroids == null || task.Centroids.length != task.K)

throw "Kmeans: centroids is invalid";

this.execute = function (d) {

function euclidianDistance(a, b) {

var dV1 = Math.abs(a.V1 - b.V1);

var dV2 = Math.abs(a.V2 - b.V2);

var dV3 = Math.abs(a.V3 - b.V3);

var maxdV1 = 180;

var maxdV2 = 360;

if (dV1 > maxdV1/2)

dV1 = maxdV1 - dV1;

if (dV2 > maxdV2/2)

dV2 = maxdV2 - dV2;

ds = Math.sqrt(dV1 \* dV1 + dV2 \* dV2 + dV3 \* dV3);

return ds;

};

var n = d.vectors.length;

var assignments = new Array(n);

for (var i = 0; i < n; i++) {

var vector = d.vectors[i];

var mindist = Number.MAX\_VALUE;

var best = 0;

for (var j = 0; j < d.k; j++) {

var dist = euclidianDistance(d.centroids[j], vector);

if (dist < mindist) {

mindist = dist;

best = j;

}

}

assignments[i] = {

Id: 0,

C: best,

V: i

};

}

return assignments;

};

this.handleResults = function (d) {

task.Vectors = null;

task.Centroids = null;

task.Assignments = d;

};

}

else if (task.State == Processing.calculationState.UpdateCentroidsLoop) {

if (task.Centroids == null || task.Centroids.length != task.K)

throw "Kmeans: centroids is invalid";

if (task.Assignments == null)

throw "Kmeans: assignments is invalid";

this.execute = function (d) {

function addVectors(a, b) {

return {

Id: a.Id,

V1: a.V1 + b.V1,

V2: a.V2 + b.V2,

V3: a.V3 + b.V3

};

};

var n = d.vectors.length;

var centroids = new Array(d.k);

for (var i = 0; i < n; i++) {

var cluster = d.assignments[i].C;

if (centroids[cluster] == null)

centroids[cluster] = d.vectors[i];

else

centroids[cluster] = addVectors(centroids[cluster], d.vectors[i]);

}

return centroids;

};

this.handleResults = function (d) {

task.Vectors = null;

task.Centroids = d;

task.Assignments = null;

};

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | **Додаток Б. Діаграма класів** |
| Виконав | Бабець А.В. |  |  | Діаграма класів | Сторінка |
| Перевірив | Мельник Р.А |  |  | 1 з 1 |
| ІКНІ кафедра ПЗ ПЗСм-21 | | | |
|  | | | | | | **Додаток В. Діаграма послідовностей** |
| Виконав | Бабець А.В. |  |  | Діаграма послідовностей | Сторінка |
| Перевірив | Мельник Р.А |  |  | 1 з 1 |
| ІКНІ кафедра ПЗ ПЗСм-21 | | | |
|  | | | | | | **Додаток Д. Діаграма компонентів** |
| Виконав | Бабець А.В. |  |  | Діаграма компонентів | Сторінка |
| Перевірив | Мельник Р.А |  |  | 1 з 1 |
| ІКНІ кафедра ПЗ ПЗСм-21 | | | |
|  | | | | | | **Додаток Е. Діаграма варіантів використання** |
| Виконав | Бабець А.В. |  |  | Діаграма варіантів використання | Сторінка |
| Перевірив | Мельник Р.А |  |  | 1 з 1 |
| ІКНІ кафедра ПЗ ПЗСм-21 | | | |

Додаток Ж

Інструкція користувача

1.Компоненти ПЗ

Система кластеризації даних за допомогою обчислювальних ресурсів веб-клієнтів (проект jsDSTR) поставляється у вигляді репозиторію GIT, який можна завантажити за наступною адресою:

*https://github.com/tesss/jsdstr.git*

Для завантаження необхідно встановити клієнт GIT та виконати команду «Clone repository» з адресою, вказаною вище (див. документацію по клієнту GIT).

Репозиторій містить наступні папки:

* */JSDstr/Csv2Sql/* – програма-конвертер з формату даних csv у запити sql (Windows Forms Application);
* */JSDstrDB/* - деякі резервні копії бази даних, допоміжні скрипти тощо;
* /JSDstrModelling/ – деякі UML-діаграми проекту;
* */JSDstr/* - основний проект розподіленої системи. Представлений у вигляді рішення Visual Studio 2010. Містить наступні складові:
  + *App\_Data*;
  + *App\_GlobalResources* – ресурси (включаючи ключі локалізації);
  + *bin* – вихідна папка скомпільованого проекту;
  + *Content* – ресурси веб-сторінок (css, малюнки тощо);
  + *Controllers* – контролери mvc додатку;
  + *Helpers* – допоміжні класи;
  + *Interfaces* – інтерфейси;
  + *Libs* – додаткові dll-бібліотеки для системи;
  + *Models* – моделі (відображення таблиць баз даних);
    - *Repositories* – клас репозиторію.
  + obj – папка вихідних об’єктніх файлів, містить вихідні файли при розгортанні системи;
  + *Properties*;
  + *Scripts* – додаткові скрипти від інших розробників та script.js – набір скриптів для даної системи;
  + *Services* – класи сервісів;
  + *Sql* – sql-запити для створення структури таблиць, заповнення даних тощо;
  + *ViewObjects* – представлення об’єктів для клієнта;
  + *Views* – представлення веб-сторінок.

1. Встановлення та налаштування ПЗ

Для розгортання веб-додатку на сервері необхідно зробити наступні кроки:

* Створити новий веб-сайт у системі управління IIS;
* У відповідну папку скопіювати скомпільований проект (/JSDstr/);
* Створити нову базу даних MS SQL;
* Згенерувати стандартні таблиці для Forms Authentication за допомогою утиліти aspnet\_regsql.exe (\Windows\Microsoft.NET\Framework\v4.0.30319\ aspnet\_regsql.exe), у якій вказати створену базу даних;
* Створити всі необхідні таблиці проекту в базі даних ( /JSDstr/SQL/tables.sql);
* Заповнити таблицю даних сейсмічної активності (/JSDstr/SQL/Earthquake\_FillData.sql);
* У конфігураційному файлі web.config прописати рядок з’єднання під назвою «ApplicationServices» для створеної бази даних, наприклад:

<connectionStrings>

<add name="ApplicationServices" connectionString="Data Source=basque\sqlexpress;Initial Catalog=jsDSTR;Integrated Security=True;Pooling=False" providerName="System.Data.SqlClient"/>

</connectionStrings>

* Прописати потрібні параметри k-means алгоритму у файлі web.config:

1. Кількість кластерів

<add key="KmeansK" value="4"/>

1. Максимальна кількість ітерацій алгоритму

<add key="MaxIterations" value="10"/>

1. Кількість векторів для відправки клієнту на кроці прив’язки

<add key="AssignmentsSlotCapacity" value="200"/>

1. Кількість векторів для відправки клієнту на кроці зміни центроїдів

<add key="UpdateCentroidsSlotCapacity" value="200"/>

* Запустити сайт

Для запуску локального веб-сервера Visual Studio необхідно попередньо створити базу даних (процес описаний вище), відкрити рішення /JSDstr/ JSDstr.sln та запустити проект.

1. Вирішення проблем та помилок:

Всі не оброблені виняткові ситуації та основні події записуються в логах системи (таблиця Log). Для перегляду інформації про необроблені виключні ситуації потрібно запустити, наприклад, такий запит, який поверне останні 100 записів:

*select top 100 \* from Log*

*where Type = 1*

*order by CreatedDate desc*

Для перезапуску обчислень (глобальних, наприклад при помилці), необхідно у таблиці Settings встановити значення для CurrentCalculationId = 0:

*update Settings*

*set value = 0*

*where key = ‘CurrentCalculationId’*

1. Базові функції ПЗ

Програмний продукт надає наступний функціонал:

* Головна сторінка проекту [*/*]

Навігація відбувається за допомогою посилань у головному меню (вверху сторінки). Для вибору мови інтерфейсу необхідно відкрити випадаючий список у верхньому правому куті сторінки.

* Інформація про проект [*/about*]

У правій частині сторінки знаходиться загальна статистика обчислень системи.

* Сторінка реєстрації користувачів та входу в систему [*/login*]

Також надає можливість увійти в систему як анонімний користувач.

* Сторінка запуску розподілених обчислень [*/processing*]

Для запуску обчислень необхідно клікнути кнопку «Старт», для зупинки відповідно «Стоп». Оновлення статистики обчислень, завершення обчислень та запуск нових сесій відбувається автоматично. Для перегляду, що саме передається між клієнтом та сервером і які події виникли, можна відкрити веб-консоль браузера та переглянути інформацію.

* Сторінка перегляду результатів кластеризації [*/results*].