МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

„ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ”

«Паралельні обчислення»

Звіт з курсової роботи

На тему

Тема: ««Побудова інвертованого індексу»

Виконав:

студент групи ДА-82

Коновал В.О.

Варіант №13

Київ 2021

Інвертований індекс - це механізм швидкого пошуку за ключовими словами. Хоча основна реалізація інвертованого індексу може бути дуже складною задля оптимізації швидкості пошуку та максимальної компресії даних, щоб збільшити можливу кількість даних у пам'яті, основна ідея є дуже простою. Для створення інвертованого індексу може бути корисним (хоча і не обов'язково) спочатку створити прямий індекс, який співставляє документ та список термінів, що є у ньому. Це корисно для пошуку по документу, щоб побачити, які слова він містить, але це менш корисно, якщо ви намагаєтеся знайти, які документи відповідають певному набору ключових слів, тому що доведеться проходитися в циклі по списку слів для кожного документа для визначення чи міститься будь-яке з запитуваних ключових слів в документі. Замість цього пошукові системи покладаються на інвертовану версію цього індексу, де кожне ключове слово відповідає набору документів, які містять його, що робить складність пошуку O(log n) для будь-якого ключового слова.

**Принцип роботи програми**

Програма складається з двох проектів: Server та Client. Проект Server буде називатися серверною частиною, Client – клієнтська частина.

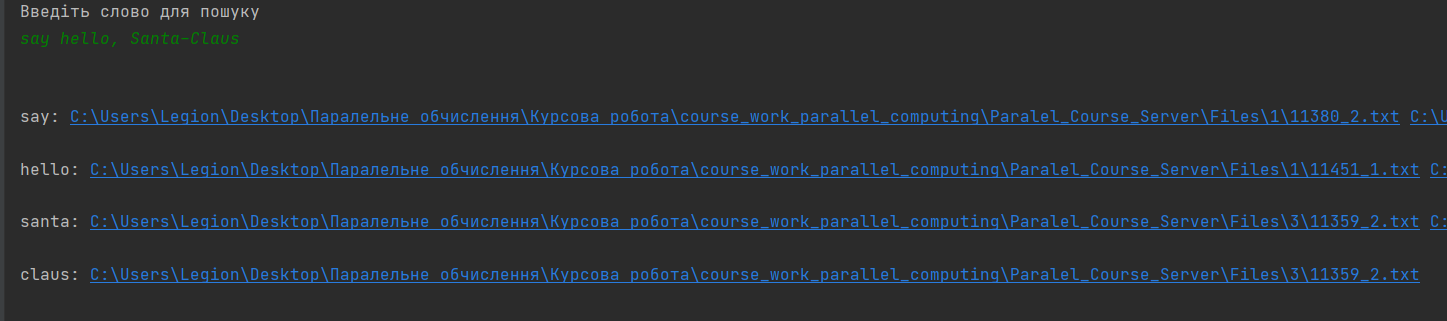
На серверній частині створюється об’єкт server класу ServerSocket, даному об’єкту надається порт, з яким він буде працювати. Далі відбувається створення об’єкту класу InvertedIndex і виклик функції main цього класу, дана функція проводить операцію побудови індексу з наявних у папках файлах, результат виконання повертається у раніше створений об’єкт. Далі відбувається створення об’єкту класу Socket і викликається метод accept, який використовується для очікування на підключення клієнтів. Метод accept буде виконуватись нескінченно, поки якийсь з клієнтів не спробує звернутися до нього. Якщо це відбудеться, тоді accept прийме з’єднання, створить об’єкт сокет для комунікації і після цього поверне цей об’єкт. Конструктор Connect створює сокет для конкретного клієнта, об’єкт класу InvertedIndex, у конструктор якого передаються результати роботи побудови індексу, яка було зроблена раніше та потоки(стріми) введення/виведення для цього сокета.

В залежності від введеного клієнтом запиту – сервер надсилає у відповідь результат пошуку файлів, у яких присутні слова з введеної користувачем у термінал строки.

Процес побудови індексу відбувається за допомогою методу класу InvertedIndex indexFile, який приймає текстовий шлях до певного файлу. Сам метод створює об’єкт класу File і перевіряє, чи є такий файл у списку з файлами, що були індексовані, якщо ні, то він додає його. Далі він відкриває даний фалй на читає дані з файлу використовуючи BufferedReader. Далі він проходиться по кожному слову з файлу, приводить слово до нижнього регістру функцією toLowerCase(), якщо дане слово присутнє в списку, додає до даного слова шлях до файлу. Якщо слово ще не було створене, то створює його.

Паралельність доступу різними потоками до одного списку можливо завдяки використанню конструкції ConcurrentHashMap, яка надає можливість одночасної роботи з нею багатьом потокам.

**Результати роботи програми:**



Зеленим шрифтом показано введену користувачем строку, далі показано відповідь сервера на запит.

**Порівняння швидкостей процесу побудови індексу в залежності від кількості потоків**

Як ми можемо бачити, для заданої, згідно завдання, кількості файлів, оптимальною кількістю потоків буде 4, подальше збільшення кількості потоків призводить або до незначного виграшу або до незначного програшу у часі(що відбувається частіше).

**Висновок**

В результаті виконання даної курсової роботи було розглянути метод побудови інвертованого індексу у багатопоточному середовищі з паралельним доступом до памяті. Було порівняно швидкість виконання процесу індексації в залежності від кількості потоків. Було встановлено, що для 2000 файлів оптимальним є використання 4 потоків. Також процес пошуку слів було розроблено з використання мережевих сокетів, де серверна частина виконує побудову індексу а також пошук файлів де зустрічаються слова зі строки, яку сервер отримує від клієнтської частини.

Додаток 1. Лістинг коду програми.

Серверна частина:

Клас Main

import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.\*;  
  
public class Main {  
 public static final int *PORT* = 7002;  
 public static LinkedList<Connect> *serverList* = new LinkedList<>();  
  
 public static void main(String[] args) throws IOException {  
 ServerSocket server = new ServerSocket(*PORT*);  
 InvertedIndex Index;  
 Index=InvertedIndex.*main*();  
  
  
 try {  
 System.*out*.println("Сервер запущено.");  
 while (true) {  
 Socket socket = server.accept();  
 System.*out*.println("\nНовий користувач.");  
 try {  
 *serverList*.add(new Connect(socket,Index));  
 } catch (IOException e) {  
 socket.close();  
 System.*out*.print(e);  
 }  
 }  
 } finally {  
 server.close();  
 }  
 }  
  
 static class Connect extends Thread {  
  
 private Socket socket;  
 private BufferedReader input;  
 private BufferedWriter output;  
 InvertedIndex Index;  
  
  
 public Connect(Socket socket, InvertedIndex Ind) throws IOException {  
 this.socket = socket;  
 this.Index= Ind;  
 input = new BufferedReader(new InputStreamReader(socket.getInputStream()));//Получение данных  
 output = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));//Отправка данных  
 start();  
 }  
  
 @Override  
 public void run() {  
 String word;  
 String Answer;  
  
 try {  
 for (Connect vr : *serverList*) {  
 vr.SendToClient("Введіть слово для пошуку");  
 }  
 while (true) {  
 word = input.readLine();//Отримання даних  
  
 if(word.equals("Стоп")) {  
 break;  
 }  
  
 else{  
 Answer =InvertedIndex.*search*(Arrays.*asList*(word.split("\\W+")),Index.Index,Index.Files);  
 }  
  
 for (Connect vr : *serverList*) {  
 vr.SendToClient(Answer+"\n");  
 }  
 }  
  
 } catch (IOException e) {  
 System.*out*.print(e);  
 }  
 }  
  
 private void SendToClient(String msg) {  
 try {  
 output.write(msg + "\n");  
 output.flush();  
 } catch (IOException ignored) {}  
 }  
 }  
}

Клас InverteIndex

import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.lang.System;  
import java.util.\*;  
import java.util.concurrent.ConcurrentHashMap;  
  
class InvertedIndex {  
 Map<String, List<Tuple>> Index;  
 List<String> Files;  
  
 InvertedIndex(Map<String, List<Tuple>> In, List<String> Fi){  
 this.Index = In;  
 this.Files = Fi;  
 }  
  
 public void indexFile(String FilePath) throws IOException {  
 File File = new File(FilePath);  
 int FileNum = Files.indexOf(File.getPath());  
 if (FileNum == -1) {//если в листе нет такого файла то мы добавляем его  
 Files.add(File.getPath());  
 FileNum = Files.size() - 1;  
 }  
  
 int Position = 0;  
 BufferedReader Reader = new BufferedReader(new FileReader(File)); //чтение текста из файла  
 for (String Line = Reader.readLine(); Line != null; Line = Reader.readLine()) {  
 for (String word : Line.split("\\W+")) {// \\W - любой символ кроме буквенного или цифрового знака  
 String lowWord = word.toLowerCase();  
 Position++;  
 List<Tuple> Idx = Index.get(lowWord);  
 if (Idx == null) {  
 Idx = new LinkedList<>();  
 Index.put(lowWord, Idx);  
 }  
 Idx.add(new Tuple(FileNum, Position));  
 }  
 }  
 System.*out*.println("indexed " + File.getPath() + " " + Position + " words");  
 }  
  
  
  
 public static String search(List<String> words,Map<String, List<Tuple>> Index,List<String> Files) {  
 String Result="";  
  
 for (String word : words) {  
 String SubResult ="\n";  
 Set<String> answer = new HashSet<>();  
 String lowWord = word.toLowerCase();  
 List<Tuple> idx = Index.get(lowWord);  
 if (idx != null) {  
 for (Tuple t : idx) {  
 answer.add(Files.get(t.fileno));  
 }  
 }  
 SubResult +=lowWord+": ";  
 for (String f : answer) {  
 SubResult +=f+" ";  
 }  
 Result+="\n"+SubResult;  
 }  
 return Result;  
 }  
  
  
 public static InvertedIndex main() {  
 Map<String, List<Tuple>> index = new ConcurrentHashMap<>();  
 List<String> files = new ArrayList<>();  
 InvertedIndex Index = new InvertedIndex(index,files);  
 int ProcessNum=8;  
  
 try {  
 String path =new File("").getAbsolutePath();  
 File Dir =new File(path.concat("\\Files"));  
 File [] DirList = Dir.listFiles();  
 ArrayList AllFileList =new ArrayList();  
 for(File Folder : DirList){  
 File [] SubDirList = Folder.listFiles();  
 for(File SubFolder : SubDirList) {  
 AllFileList.add(SubFolder.getAbsoluteFile());  
 }  
 }  
  
 ArrayList<Worker> Workers = new ArrayList<>();  
 for (int i=0; i<ProcessNum; i++){  
 ArrayList<String> FileList = new ArrayList<>();  
 for(int m=0; (i+m)<AllFileList.size(); m+=ProcessNum){  
 FileList.add(AllFileList.get(i+m).toString());  
 }  
 Worker Work =new Worker(i,FileList,Index.Index,Index.Files);  
 Workers.add(Work);  
 }  
 for (int i=0; i<ProcessNum; i++){  
 Workers.get(i).start();  
 }  
 for(int i=0; i<ProcessNum;i++){  
 Workers.get(i).join();  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return Index;  
 }  
  
  
 public static class Worker extends Thread{  
 int Number;  
 Map<String, List<Tuple>> index;  
 ArrayList<String> PersonalList;  
 List<String> files;  
  
 Worker(int i, ArrayList<String> List,Map<String, List<Tuple>> in,List<String>f){  
 this.Number=i;  
 this.index=in;  
 this.PersonalList=List;  
 this.files=f;  
 }  
  
  
 public void run(){  
 InvertedIndex idx = new InvertedIndex(index,files);  
  
 for(int i=0; i<PersonalList.size(); i++){  
 try {  
 idx.indexFile(PersonalList.get(i));  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 public class Tuple {  
 public int fileno;  
 private int position;  
  
 public Tuple(int fileno, int position) {  
 this.fileno = fileno;  
 this.position = position;  
 }  
 }  
}

Клієнтська частина

Клас Main

import java.io.\*;  
import java.net.Socket;  
  
public class Main {  
  
 public static String ipAddr = "localhost";  
 public static int port = 7002;  
  
 private Socket socket;  
 private BufferedReader input;  
 private BufferedWriter output;  
 private BufferedReader inputUser;  
 private String ip;  
 private int portConnect;  
  
  
 public Main(String ip, int portConnect) {  
 this.ip = ip;  
 this.portConnect = portConnect;  
  
 try {  
 this.socket = new Socket(ip, portConnect);  
 } catch (IOException e) {  
 System.err.println("Помилка з'єднання");  
 }  
 try {  
 inputUser = new BufferedReader(new InputStreamReader(System.in));  
 input = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 output = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
  
 new ReadFromServer().start();  
 new WriteToServer().start();  
 } catch (IOException e) {  
  
 Main.this.closeConnection();  
 }  
  
 }  
  
 private void closeConnection() {  
 try {  
 if (!socket.isClosed()) {  
 socket.close();  
 input.close();  
 output.close();  
 }  
 } catch (IOException ignored) {}  
 }  
  
 private class ReadFromServer extends Thread {  
 public void run() {  
 String str;  
 try {  
 while (true) {  
 str = input.readLine();  
 if (str.equals("Стоп")) {  
 Main.this.closeConnection();  
 break;  
 }  
 System.out.println(str);  
 }  
 } catch (IOException e) {  
 Main.this.closeConnection();  
 }  
 }  
 }  
  
 public class WriteToServer extends Thread {  
  
 public void run() {  
 while (true) {  
 String userWord;  
 try {  
 userWord = inputUser.readLine();  
 if (userWord.equals("Стоп")) {  
 output.write("Стоп" + "\n");  
 Main.this.closeConnection();  
 break;  
 } else {  
  
 output.write(userWord + "\n");  
 }  
 output.flush();  
 } catch (IOException e) {  
 Main.this.closeConnection();  
 }  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 new Main(ipAddr, port);  
 }  
}