ПРОЕКТ
ПО МРЕЖОВО ПРОГРАМИРАНЕ

на тема

Търсене на зависимост на поведение на потребител чрез TCP протокол за комуникация между клиент и сървър

на Антон Детелинов Чернев
ф.н. 81505, група 5, поток 2, КН

Съдържание:

1. Тема на проекта
2. Теоретично описание на решението на проблема
3. Проектиране
4. Бележки по реализацията
5. Ръководство за инсталация
6. Резултати
7. Литература
8. Приложение

1. Тема на проекта

Задачата е да се реализират клиентско и сървър приложения за търсене на зависимост на поведението на потребител. Клиентското приложение изпраща лог файл в csv формат, съдържащ следните колони: Time, Event context, Component, Event name, Description, Origin, IP address, който достига до сървъра чрез TCP сокети. Сървърът анализира поведението на базата на данните от лог файла, като използва алгоритъма за извличане на често срещащи се данни AprioriTID Algorithm. След извличането резултатите се предават обратно на клиента и се визуализират в подходящ за потребителя формат.

2. Теоретично описание на решението на проблема

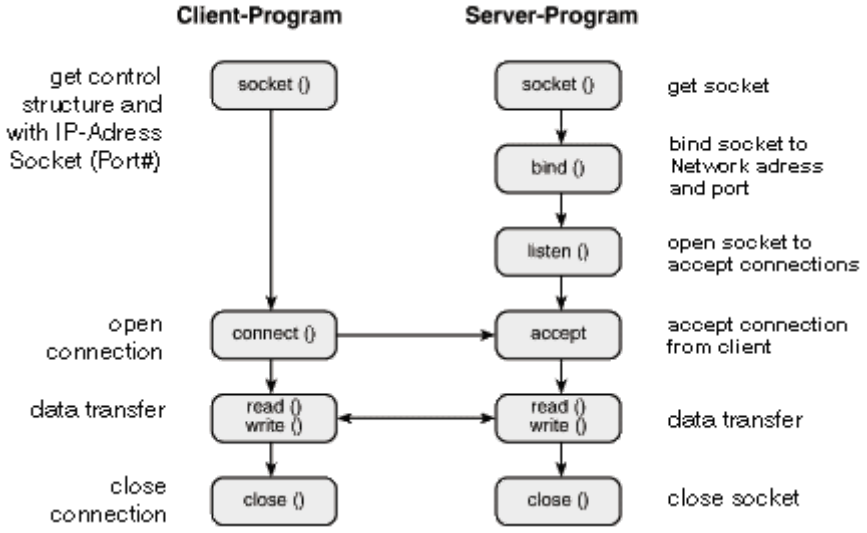
Комуникационният аспект от решението на задачата се моделира чрез известния модел в мрежовото програмиране “Клиент-Сървър” с едновременна (concurrent) обработка. Сървър приложението отваря TCP сокет и го закача за определен порт, който е познат за клиента. Слуша за заявки от клиента за стартиране на комуникация. При получаване на такава заявка, сървърът отваря нов сокет, който е пригоден специално за комуникацията с текущия клиент. За да извърши преноса и обработката на данни с текущия клиент, без да блокира възможността за отваряне на връзки с нови клиенти, работата с новия сокет се извършва изцяло от нова нишка, която работи едновременно с главната.

За анализ на данните се използват три полета от лог файла – това са Event context, Event name и IP address. Чрез прилагането на ApriotiTID Algorithm получаваме всички тройки от елементи от тези колони, които се срещат във файла определен брой пъти. Тъй като посоченият алгоритъм за извличане на данни работи с транзакции

от положителни числа, е нужно да представим текстовите данни като числа. Изграждаме биективно съответствие между низовете, които са ни дадени, и числата, които ще подадем като вход за алгоритъма.

3. Проектиране

Основните взаимоотношения при комуникацията между клиента и сървъра са представени в следната схема:



3.1. Клиентско приложение.

За работата на клиентското приложение отговаря класът Client.

Първата стъпка от работата на клиентското приложение е събиране на информация от потребителя, нужна за комуникация със сървъра – това са IP адресът и портът. Следва пътят до лог файла, който съдържа информацията, от която трябва да бъде извлечено поведението на потребителя. Последният параметър, който се подава при стартирането на приложението, е величината minOccurrences, която указва кои данни да бъдат предадени на клиента. На екрана се визуализират тези данни, които се срещат поне minOccurrences брой пъти.

Следва стъпката за предаване на информацията към сървъра, където тя трябва да бъде обработена. За удобство и поради

естеството на работата на двете приложения, данните се предават в текстов формат. Чрез TCP сокет се прави връзка със сървъра и се достъпват входния и изходния поток за комуникация. Предава се стойността minOccurrences по изходния поток към сървъра. Отваря се файла с логовете и се прочита, като заедно с това се изпраща по изходния поток. След като всички данни са предадени към сървъра се указва край на заявката чрез изпращане на празен ред.

Последната стъпка на клиента е да прочете извлечените от сървъра данни, които се предават по входния поток на сокета. Те са предварително форматирани и готови за визуализиране пред потребителя, затова директно се отпечатват на стандартния вход. Аналогично на предаването на данни от клиента към сървъра, и тук край на изпращането на данни се бележи с получаването на празен ред.

3.2. Сървър приложение.

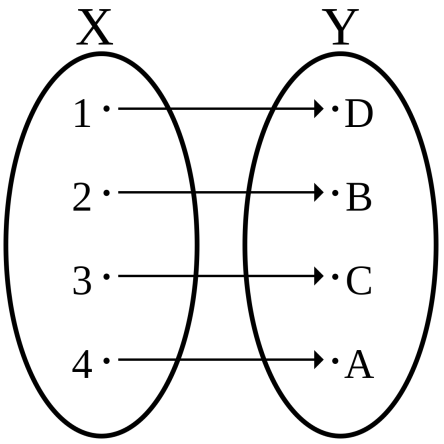
Коренът на йерархията от класове, отговарящи за работата на сървъра, е класът Server. Неговата функция е да изгради сървърния сокет и да слуша за заявки за връзка. При получаване на такава заявка Server създава нов сокет, който отговаря за новопоявилата се връзка и се обръща към класа Handler. Задачата на обект от тип Handler е да обработи една конкретна заявка от клиента. Затова този клас имплементира интерфейса Runnable и се подава на клас за управление на нишки, който стартира изпълнението на Handler в нова нишка.

В класа Handler е съсредоточена основната работа на сървър приложението. Първата цел на класа е да се прочете заявката на сървъра (нейния формат е описан по-горе). И тук разполагаме с входен и изходен поток от сокета, които отговарят за комуникацията. Първо се прочита стойността minOccurrences, а след това и редовете от лог файла. За да се преработят и запишат, се използва отделен клас EventLog, който държи списък от всички обекти от тип Event, служещи се по-нататъшна обработка. EventLog разполага с методи за добавяне на Event и за записване на всички

Event обекти във файл (това се налага, тъй като алгоритъмът работи с входни данни от файл).

Да разгледаме структурата на класа Event. Той съдържа три полета от тип символен низ, в които се съхранява Event context, Event name и IP address от лог файла. Имаме също два метода за кодиране и декодиране на обект Event, като тези операции се извършват с помощта на клас Encoder. Причината това да се налага е, че алгоритъма AprioriTID работи с цели положителни числа, а не със символни низове. При записването на данните за събитията във файл се налага да се кодират символните низове като числа, а при анализа на резултатите от алгоритъма трябва да разполагаме с метод за декодиране, който да превърне получените числа обратно в обекти от тип Event. За удобство се добавя и метод за сериализация на данните от обект Event, чрез който се получава символен низ, удобен за изпращане до клиента и за разчитане от потребителя.

Преобразуването на символните низове в числа и обратно е абстрахирано чрез класа Encoder.



Обекти от този тип разполагат със структури за преобразуване на низ в число (HashMap), за преобразуване на число в низ (ArrayList) и за откриване на типа на низ, кодиран с някакво число (ArrayList) – това е всъщност типът информация, която е кодирана, т.е. контекст, име или адрес. Тук трябва да се отбележи, че всеки един обект от тип Event не се кодира самостоятелно, а като част от всички останали обекти от тип Event. Това е и причината на класът

EventLog да се подава Encoder, който да отговаря за кодирането на всички събития, а по-късно да се използва и за декодиране.

След като всички данни, получени от клиента и съхранени в EventLog, са записани във файл, класът Handler е готов да изпълни алгоритъма AprioriTID. Използва се реализацията SPMF [1] – свободна библиотека с алгоритми за извличане на данни. Алгоритъмът работи с число minsup, което указва какъв дял от всички транзакции трябва да представлява някое множество от данни, за да бъде то представено в резултатите от изпълнението. Стойността minsup се изчислява тривиално като частно на minOccurrences и общия брой записи в лог файла. Тъй като алгоритъмът връща множества с произволна мощност, а приложението се интересува само от тройките числа (тройка отговаря на контекст, име и адрес), извличаме само резултатите с три елемента. За да имаме по-ясна презентация на резултатите, сортираме получените данни според това колко пъти са се срещали в целия лог файл. Остава само да се декодират резултатите, да се сериализират и да се изпратят към клиента.

4. Бележки по реализацията

Приложенията са реализирани на Java. Обръща се внимание на следните моменти от имплементацията:

4.1. TCP комуникация.

Осъществена е чрез инструментите от вградената библиотека java.net. Класът ServerSocket изпълнява ролята на посрещащ сокет, който слуша на конкретен порт. По подразбиране се задава слушане на всички адреси на устройството. Методът accept() получава обект от тип Socket, който е сокетът за връзка. Класът Socket предоставя методи за получаване на входния и изходния поток.

4.2. Входно-изходни операции.

Реализирани са с помощта на вградената библиотека java.io. Тъй като работата на приложението предразполага използването на потоци за текстови данни, за удобство и ефективност е избрано да се работи с поток за четене BufferedStream и поток за писане PrintStream. За четенето и писането от файлове се използват FileReader и FileWriter. Внимание се отделя на метода flush() за освобождаване на вътрешните буфери на потоците при приключване на предаването на данни.

4.3. Реализация на Encoder.

Кодирането от низ към число изисква използването на структура с ключ и стойност. Подходящ е класът HashMap, тъй като не се използва наредба на съхраняваните низове, т.е. не се нуждаем от функционалността на балансираното дърво при SortedMap. За обратната операция, т.е. декодирането, няма нужда от по-сложния HashMap, а може да се използва ArrayList, тъй като числата могат да се възприемат като индекси в структурата.

4.4. AprioriTID Algorithm.

За да извикаме алгоритъма, използваме пакета ca.pfv.spmf.algorithms.frequentpatterns.aprioriTID. За удобство указваме резултатът от него да се запише в списък от обекти Itemset от пакета ca.pfv.spmf.patterns.itemset_array_integers_with_tids. Обръщаме внимание на методите getAbsoluteSupport() и getItems() от класа Itemset, съответно за намиране на броя срещания на конкретно множество и за получаване на самото множество.

4.5. Работа с временни файлове.

Преди да извикаме алгоритъма, записваме входните му данни във временен файл. За да не се получават колизии между работата на различните нишки, името на файла бива генерирано чрез произволно число от 0 до 999. След работата на алгоритъма файлът се изтрива, защото той повече не е нужен.

4.6. Работа с нишки.

Използват се средствата на пакета java.util.concurrent и по-точно Executors.newCachedThreadPool(), който връща Executor. Той извършва метода execute(), който приема като аргумент клас, имплементиращ интерфейс Runnable. Предимството на този метод за работа е, че свободните нишки могат повторно да се използват, което води до по-ефикасно разпределяне на ресурсите.

5. Ръководство за инсталация

- Разархивирайте kn_81505_1.2.zip;
- През конзолата се позиционирайте в директорията netprog/out/production/netprog;
- Изпълнете java server.Server, за да стартирате сървър приложението;
- Изпълнете java client.Client, за да стартирате клиентското приложение, задайте адреса на сървъра и порт 8080, след това пътя до лог файла и стойността minOccurrences.

6. Резултати

Изпълнението на клиентското приложение при условие, че се свърже успешно със сървър приложението и че всички данни са в подходящия формат, отпечатва на екрана списък с действията, които потребителят е извършвал най-често, и по-точно, поне minOccurrences брой пъти. При засичане на грешка, програмата връща информация за настъпилото изключение и приключва работата си.

7. Литература

[1] SPMF Documentation – http://www.philippe-fournier-viger.com/spmf/index.php?link=documentation.php

[2] Java 8 API Specification https://docs.oracle.com/javase/8/docs/api

[3] Лекции по Мрежово програмиране, ФМИ, доц. д-р Аделина Алексиева

8. Приложение

- client.Client

```
package client;

import java.net.*;
import java.io.*;
import java.util.Scanner;

public class Client {
    private static String host;
    private static int port;
    private static String logfile;
    private static BufferedReader input;
    private static PrintStream output;

    private static int minOccurrences;

    public static void main(String[] argv) throws IOException {
        getUserInput();

        Socket socket = null;
        try {
            socket = new Socket(host, port);
            input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            output = new PrintStream(socket.getOutputStream());

            output.println(minOccurrences);
            sendFile(logfile);
            output.println();
            output.flush();

            receiveMinedData();

        } catch(UnknownHostException e) {
            System.out.println("Unknown host");
            e.printStackTrace();
        } catch(IOException e) {
            e.printStackTrace();
        } finally {
            if(socket != null) {
                socket.close();
            }
        }
```

```
    }

    private static void getUserInput() {
        Scanner userInput = new Scanner(System.in);

        System.out.print("Set host: ");
        host = userInput.nextLine();
        System.out.print("Set port: ");
        port = Integer.parseInt(userInput.nextLine());
        System.out.print("Set log file name: ");
        logfile = userInput.nextLine();
        System.out.print("Set min occurrences: ");
        minOccurrences = Integer.parseInt(userInput.nextLine());
    }

    private static void sendFile(String filename) throws IOException {
        BufferedReader fileStream = new BufferedReader(new FileReader(filename));
        String line;

        while((line = fileStream.readLine()) != null) {
            if(!line.trim().isEmpty()) { output.println(line); }
        }
    }

    private static void receiveMinedData() throws IOException {
        String line = null;
        while(!(line = input.readLine()).isEmpty()) {
            System.out.println(line);
        }
    }
}
```

- server.Server

```
package server;

import java.io.*;
import java.net.*;
import java.util.concurrent.*;

public class Server {
    private final int port;
    private boolean isRunning = false;

    public Server(int port) {
        this.port = port;
    }

    public void start() {
        if(!isRunning) {
```

```
            isRunning = true;
            try {
                listen();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }

    private void listen() throws IOException {
        Executor executor = Executors.newCachedThreadPool();
        ServerSocket welcomingSocket = new ServerSocket(port);
        Socket connectionSocket = null;

        while(true) {
            try {
                connectionSocket = welcomingSocket.accept();
                Handler handler = new Handler(connectionSocket);
                executor.execute(handler);

            } catch(IOException e) {
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] argv) {
        Server server = new Server(8080);
        server.start();
    }
}
```

- server.Handler

```
package server;

import java.net.*;
import java.io.*;
import java.io.PrintStream;
import java.util.Comparator;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import ca.pfv.spmf.algorithms.frequentpatterns.aprioriTID.AlgoAprioriTID;
import ca.pfv.spmf.patterns.itemset_array_integers_with_tids.*;

public class Handler implements Runnable {
    private final int eventSize = 3; // Number of fields per event

    private Socket socket;
    private BufferedReader input;
```

```
private PrintStream output;

private int minOccurrences;

private EventLog eventLog;
private Encoder encoder;

public Handler(Socket socket) throws IOException {
   System.out.println("New connection from " + socket.getInetAddress());

   this.socket = socket;
   input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
   output = new PrintStream(socket.getOutputStream());

   encoder = new Encoder();
   eventLog = new EventLog(encoder);
}

public void run() {
   try {
      getRequest();
      mineData();
      output.println();
      output.flush();

   } catch (IOException e) {
      e.printStackTrace();
   }

   close();
}

private void getRequest() throws IOException {
   String line;
   boolean isFirst = true;

   minOccurrences = Integer.parseInt(input.readLine());
   while(!(line = input.readLine()).isEmpty()) {
      if(!isFirst) {
         eventLog.addEvent(line);
      } else {
         isFirst = false;
      }
   }
}

private void mineData() throws IOException {
   // Write the data into a temporary file
   String inputFile = "input." + ThreadLocalRandom.current().nextInt(0,999) + ".in";
   File tmpFile = new File(inputFile);
   if(!tmpFile.createNewFile()) throw new IOException("Temporary file already exists");
```

```
        eventLog.writeToFile(inputFile);

        // Run the mining algorithm
        AlgoAprioriTID algo = new AlgoAprioriTID();
        double minSupport = (double) minOccurrences / eventLog.size();
        List<Itemset> result = algo.runAlgorithm(inputFile, null, minSupport).getLevels().get(eventSize);

        // Send mined data
        result.sort(Comparator.comparing(Itemset::getAbsoluteSupport).reversed());
        for(Itemset element : result) {
          Event event = Event.getDecoded(element.getItems(), encoder);
          output.println(event.serialize() + " " + element.getAbsoluteSupport() + " times.");
        }

        // Delete temporary file
        if(!tmpFile.delete()) throw new IOException("Could not delete temporary file");
    }

    private void close() {
        try {
            if(socket != null) { socket.close(); }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- server.EventLog

```
package server;

import java.io.*;
import java.util.ArrayList;

public class EventLog {
    private ArrayList<Event> log;
    private Encoder encoder;

    public EventLog(Encoder encoder) {
        log = new ArrayList<Event>();
        this.encoder = encoder;
    }

    public void addEvent(String line) {
        String[] columns = line.split(",");
        if(columns.length == 8) {
            log.add(new Event(columns[2].trim(), columns[4].trim(), columns[7].trim()));
        }
    }
```

```
    public void writeToFile(String filename) throws IOException {
        PrintWriter writer = new PrintWriter(new FileWriter(filename));

        for(Event element : log) {
            writer.println(element.getEncoded(encoder));
        }
        writer.flush();
    }

    public int size() {
        return log.size();
    }
}
```

- server.Event

```
package server;

import java.util.Arrays;

public class Event {
    private String context;
    private String name;
    private String address;

    public Event(String context, String name, String address) {
        this.context = context;
        this.name = name;
        this.address = address;
    }

    public String getEncoded(Encoder encoder) {
        int[] numbers = {
                encoder.encode(context, "context"),
                encoder.encode(name, "name"),
                encoder.encode(address, "address")
        };
        Arrays.sort(numbers);
        return numbers[0] + " " + numbers[1] + " " + numbers[2];
    }

    public static Event getDecoded(int[] numbers, Encoder encoder) {
        String context = "undefined", name = "undefined", address = "undefined";
        for(int i = 0; i < 3; i++) {
            switch(encoder.getType(numbers[i])) {
                case "context":
                    context = encoder.decode(numbers[i]);
                    break;
                case "name":
                    name = encoder.decode(numbers[i]);
```

```
                break;
            case "address":
                address = encoder.decode(numbers[i]);
        }
      }

      return new Event(context, name, address);
   }

   public String serialize() {
      return "User with IP " + address + " performed action " + name + " on " + context;
   }
}
```

- server.Encoder

```
package server;

import java.util.ArrayList;
import java.util.HashMap;

public class Encoder {
   private HashMap<String, Integer> strToInt;
   private ArrayList<String> intToStr;
   private ArrayList<String> types;
   private int lastInt;

   public Encoder() {
      strToInt = new HashMap<>();
      intToStr = new ArrayList<>();
      types = new ArrayList<>();
      lastInt = 1;
   }

   public int encode(String word, String type) {
      if(strToInt.containsKey(word)) { return strToInt.get(word); }
      else {
         strToInt.put(word, lastInt);
         intToStr.add(word);
         types.add(type);
         return lastInt++;
      }
   }

   public String getType(int number) {
      number--;
      if(0 <= number && number < types.size()) { return types.get(number); }
      else { return null; }
   }
```

```
    public String decode(int number) {
        number--;
        if(0 <= number && number < intToStr.size()) { return intToStr.get(number); }
        else { return null; }
    }
}
```