udp Escuela de Informática y Telecomunicaciones

Universidad Diego Portales

Inteligencia artificial

Tarea 1

Autor:
Italo Piermartiri

Profesor:
Martin Gutierrez

Ayudante:
Rodrigo Quezada

31 de octubre de 2022

Índice

1. Problema y solución

Para esta tarea se pide desarrollar un *web search engine* que funcione de manera distribuida para manejar cargas altas.

Para esto se utilizará; docker como contenedor de la base de datos, el servidor y el cliente de nuestro servicio, python para programar el servidor y cliente, la librería *grcpio* de python para la comunicación RCP, el framework *Scrapy* de python para extraer los datos, postgreSQL como base de datos y ...

Lo primero que se hace es elegir un set de datos, el set de datos escogido es **user-ct-test-collection-04.txt.gz**. Para manejar este set de datos se utilizará un *Script* en python. Una vez limpio el set de datos se crea un script en python también para crear las tablas, indexarlas e insertar los datos en la base de datos. Para esto se crea un documento *Dockerfile* con una imagen de python, se instala POSTGRES, se configura y se ejecuta el script para crear la base de datos.

2. Explicación de módulos de código

En el script *toCSV.py* se utilizarán expresiones regulares para seleccionar todos los URL dentro del set de datos y agregarlas a un arreglo llamado *URL_ noFilter* ya que este tiene URL's repetidos. Luego se utiliza la función *unique* de la librería *panda* de python para eliminar los datos repetidos y finalmente se escribe un archivo de texto llamado *'URLs_ filter.txt'* en donde se guardan todos los URL's distintos.

```
1   import csv
2   import re
3   import pandas as pd
4   from bs4 import BeautifulSoup as bs
5   import requests as req
6   import time
7   time_init = time.time()
8   URL_noFilter = []
9   URLs_filter = []
10  new_URLs = []
11  with open('user.csv') as csvfile:
12      reader = csv.reader(csvfile, delimiter='\t',quotechar='\t')
13      for row in reader:
14          x = re.search("^http.*",row[4])
15          if x:
16              URL_noFilter.append(row[4])
17          else:
18              next
19  URLs_filter = pd.unique(URL_noFilter)
20  print("URLs_filter se ha realizado.\n")
21  csvfile.close()
22
23  f = open("URLs_filter.txt","w")
24  for url in URLs_filter:
25      f.write(url+"\n")
26  f.close()
```

Figura 1: toCSV.py - Limpieza de datos.

El archivo de texto creado anteriormente será utilizado para hacer la extracción de datos contenidos en el tag *<head>* html de cada URL. Para esto se utiliza el framework *Scrapy*. Se crean 4 'Spiders' ya que para acelerar la ejecución de estas arañas se dividirá el arhivo de texto en 4 conjuntos. Se consultan los campos: *title, description, keywords* y el *URL* de cada sitio visitado, almacenandolos en formato JSON. El output de cada Spider se escribe sobre un archivo .json (datos1.json, datos2.json, datos3.json y datos4.json).

```python
import scrapy
import csv, os, json

from crawler.items import CrawlerItem

class Spiders(scrapy.Spider):
    name = "spider1"
    start_urls = []
    with open("/mnt/c/Users/Drach/Desktop/Sistemas Distribuidos/Tarea 1/BDD/URLs_filter.txt","r") as r:
        data = r.readlines()[0:90000]
    r.close()
    for linea in data:
        start_urls.append(linea.strip("\n"))



    def parse(self, response):
        for item in response.xpath("/html/head"):
            yield {
                'title':response.xpath("//head/title/text()").get(),
                'description': response.xpath('/html/head/meta[@name="description"]/@content').get(),
                'keywords': response.xpath('/html/head/meta[@name="keywords"]/@content').get(),
                'URL': response.request.url
            }
```

Figura 2: spider.py - Spider 1.

A continuación se crea un documento Dockerfile para crear la imagen de nuestro contenedor de la base de datos. Para esto se utiliza una imagen de python 3.9 ya que se deben ejecutar unos scripts para crear la base. Se instalan las dependencias necesarias para su funcionamiento, se configura un usuario 'Docker' en la base de datos, se copian los archivos que contienen los datos extraidos por las Spider's en el paso anterior y los scripts de python para crear la base dentro del contenedor y se expone el puerto usado por POSTGRES, el 5432.

```
FROM python:3.9

RUN apt-get update\
    && apt-get install -y sudo\
    && apt install -y vim\
    && sudo apt install -y postgresql postgresql-contrib

RUN sudo pip install psycopg2 &&\
    sudo pip install pandas


COPY ./psqlScripts/ /home/psqlScripts/

USER postgres

RUN /etc/init.d/postgresql start &&\
    psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" &&\
    createdb -O docker docker

RUN echo "host all  all    0.0.0.0/0  md5" >> /etc/postgresql/13/main/pg_hba.conf

RUN echo "listen_addresses='*'" >> /etc/postgresql/13/main/postgresql.conf

EXPOSE 5432

VOLUME  ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]

CMD ["/usr/lib/postgresql/13/bin/postgres", "-D", "/var/lib/postgresql/13/main", "-c",

WORKDIR /home/psqlScripts
RUN python /home/psqlScripts/createTables.py
```

Figura 3: Dockerfile BDD.

La carpeta copiada al contenedor contiene los siguientes archivos:

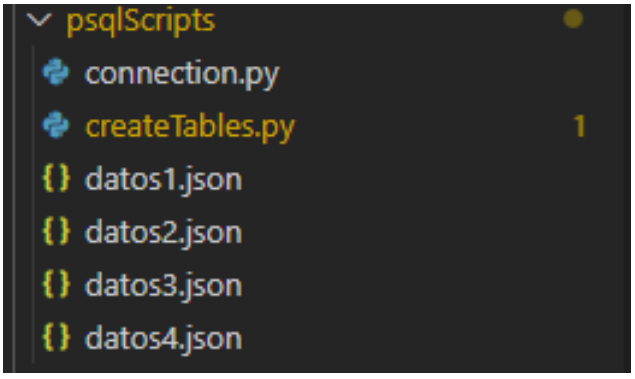

Figura 4: spider.py - Spider 1.

El archivo connection.py realiza la conexión a la base de datos creada en el dockerfile.

Primero se crea la tabla 'URLS' que contendrá el título, descripción y URL del sitio visitado, además se crea un indice por el URL.

```
try:
    conn = Connection()
    cur = conn.cursor()
    #CREATE TABLE URLS
    try:
        cur.execute(
            '''
            CREATE TABLE URLS
            (ID SERIAL PRIMARY KEY,
            TITLE TEXT,
            DESCRIPTION TEXT,
            URL TEXT);
            '''
        )
        conn.commit()
        print("TABLA URLS CREADA")

    except:
        print("ERROR CREATE TABLE URLS")
```

Figura 5: spider.py - Spider 1.

```
29      #CREATE INDEX ON URLS BY URL
30      try:
31
32
33          cur.execute(
34              '''
35              CREATE INDEX index_url ON URLS
36              (URL);
37              '''
38          )
39          conn.commit()
40          print("INDEX URLS CREADA")
41
42      except:
43          print("ERROR INDEX ON URLS")
```

Figura 6: spider.py - Spider 1.

Luego se crea la tabla 'KEYWORDS' que contiene: la keyword y la id del URL asociado a esa keyword. Se crea un indice por keyword.

```
45      #CREATE TABLE KEYWORDS
46      try:
47
48
49          cur.execute(
50              '''
51              CREATE TABLE KEYWORDS
52              (ID SERIAL PRIMARY KEY,
53              KEYWORD TEXT NOT NULL,
54              URL_ID INTEGER NOT NULL);
55              '''
56          )
57          conn.commit()
58          print("TABLA KEYWORDS CREADA")
59
60      except:
61          print("ERROR CREATE TABLE KEYWORDS")
```

Figura 7: spider.py - Spider 1.

```
63      #CREATE INDEX ON KEYWORDS BY KEYWORD
64      try:
65
66
67          cur.execute(
68              '''
69              CREATE INDEX index_keyword ON KEYWORDS
70              (KEYWORD);
71              '''
72          )
73          conn.commit()
74          print("INDEX KEYWORDS CREADA")
75
76      except:
77          print("ERROR INDEX ON KEYWORDS")
```

Figura 8: spider.py - Spider 1.

Luego se hace un ciclo *for* que recorra los 4 archivos de datos creados, se extraen los campos de cada item del JSON y se verifica si existe el campo *Keywords*, de no existir no se inserta en la base.

Si el campo existe, se inserta el URL en la tabla URLS y se pide que retorne la ID del URL ingresado.

```
80      try:
81
82          for i in range(1,5):
83              f = open('datos'+str(i)+'.json')
84              data = json.load(f)
85              for j in data:
86                  title = j['title']
87                  description = j['description']
88                  keywords = j['keywords']
89                  URLs = j['URL']
90                  if(keywords):
91
92                      #INSERTAR TABLA URLS
93                      SQLs = '''INSERT INTO URLS(TITLE,DESCRIPTION,URL) VALUES(%s,%s,%s)
94                      insert = (str(title), str(description), str(URLs))
95                      try:
96                          cur.execute(SQLs,insert)
97                      except:
98                          print("ERROR INSERT URLS")
99                      finally:
100                         conn.commit()
101                         id_url = cur.fetchone()[0]
```

Figura 9: spider.py - Spider 1.

Luego, del enorme string que compone el campo 'Keywords' se extraen todas las keywords utilizando expresiones regulares (ya que estan separados por comas) y para cada conjunto de keywords de un URL se utiliza la libreria de panda para limpiar los datos keywords repetidos. Luego, se recorre el conjunto de keywords pertenecientes a un URL y se hace ingreso de cada uno en la tabla KEYWORDS utilizando la ID del URL solicitado en la consulta anterior.

```
109             r1 = re.findall(r"\w+",keywords)
110             r1 = pd.unique(r1)
111             for key in r1:
112                 SQLs2 = ''' INSERT INTO KEYWORDS(KEYWORD,URL_ID) VALUES(%s,%s); '''
113                 insert2 = (str(key),id_url)
114                 try:
115                     cur.execute(SQLs2,insert2)
116                 except:
117                     print(id_url)
118
119                 finally:
120                     conn.commit()
```

Figura 10: spider.py - Spider 1.

Quedando así con un total de 46291 URL's distintos, de un total de más de un millon de URL's que contenía el archivo original.

```
docker=# select count(*) from urls;
 count
-------
 46291
(1 row)
```

Figura 11: spider.py - Spider 1.

```
docker=# select URL from URLS,KEYWORDS where URLS.ID = KEYWORDS.URL_ID and KEYWORDS.KEYWORD = 'early' limit 8;
                url
-----------------------------------
 http://lithiccastinglab.com
 http://www.firstschoolyears.com
 https://www.trocadero.com/
 https://www.pregnancy-info.net/
 http://www.earlychildhoodlinks.com
 https://www.kaplanco.com/
 https://www.cvcc.edu/
 http://www.nh.searchroots.com
(8 rows)
```

Figura 12: spider.py - Spider 1.

3. Explicación del funcionamiento caché

4. Resultados y análisis en las configuraciones de caché

5. Conclusión

6. Referencias

Erdogan Taskesen. (2022). *bnlearnś documentation* `https://erdogant.github.io/bnlearn/pages/html/index.html`

Python Software Foundation. (sf). *python documentation.* `https://docs.python.org/3/contents.html`

M. Gutierrez. (2022). *Probabilidades. canvas/modulos/probabilidades.pdf*

M. Gutierrez. (2022). *Cadenas de Markov. canvas/modulos/cadenasMarkov.pdf*