University *of Ljubljana*
Faculty *of Computer and Information Science*

Web Crawling

Matjaž Zupanič, Jure Tič, Miha Malenšek

Abstract

In this report we will describe our python implementation of a web-crawler that indexes government and related websites, extracting images and links. We will describe our approaches and decisions in designing our web crawler, parameters needed for it to work and the problems we encountered. We will analyze the results of our crawl, namely number of sites and pages, number of duplicates and number of binary documents and the number of images. We will try to visualize the data in a way that helps the reader understand the presented parameters. And finally we will discuss possible improvements for our web-crawler.

Keywords
web-crawling, Python, Selenium, threading

Advisors: Slavko Žitnik

Technologies

In this section we will describe some technologies that we used for implementation of our web craler.

0.1 PostgreSQL

PostreSQL is an open source relational database, great for more complicated projects due to its advanced features in ensuring data integrity and concurent processing. For this project we used a pre-prepared schema provided by Faculty of Computer and Information Science. [1]

0.2 Python Anaconda

Python Anaconda is python distribution that greatly simplifies project management with its advanced and intuitive package management system. [2]

0.3 Jupyter notebook

Jupyter notebook is an enviroment for developing Python scripts, frequently used with Python Anaconda. It can feature mutltiple kernes which can each run its own programming language. Working enviroment is devided into multiple cells, which can be eather code or text, allowing one to write well documented code and execute only the desired parts.[3]

0.4 Python libraries

For our project we used multiple python libraries that made our work a little easier. Some of these libraries are.

0.4.1 Selenium

A python library for web browser automation, originally designed to automate web applications for testing purposes. With it we define our user-agent and other options, as well as utilize our Chrome webdriver.[4]

0.4.2 Beautiful soup

A library for parsing html page content. We use it to find links and images on the site.[5]

0.4.3 Urllib

A library used for dealing with url adresses. We use it to recognize different kinds of urls and to convert relative url into absolute.[6]

0.4.4 Psycopg2

A very useful python library used for dealing with the PostgreSQL database. We use it to establish connection to the database and to execute queries on the database.[7]

0.4.5 Pandas

Is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. [8]

0.4.6 Threading

A library that enables threading and supporting functions.[9]

Design decisions

Our web-crawler runs in a Jupyter Notebook in an Anaconda environment. It stores required information such as images

and page data in a locally-run database. The web-crawler uses a queue object as its frontier (FIFO [10]), which it updates with filtered urls as it crawls given pages. It filters pages based on given criteria and additional filters, such as filtering out telephone numbers.

We implemented threading for up to twenty concurrent threads, chosen given our hardware capabilities and we adjust that value based on the number of urls in the frontier.

When visiting a page we first check if we have already checked its url. All visited urls are stored in a dictionary and each new page checks if the url is contained in that dictionary before visiting it. We check for duplicates using comparison of html content hashed with sha256. Similarly to visited urls these hashes are stored in a dictionary and in this way duplicates are determined.

Results

We had processed 14707 pages on 193 sites. Our crawler did this in around 20 hours, netting an average of 735 pages per hour. Along with page data, we logged 20742 images. Of the 14707 pages crawled 3180 were marked as duplicates. We detected multiple types of files, the representation of which seen in **Figure** 1.

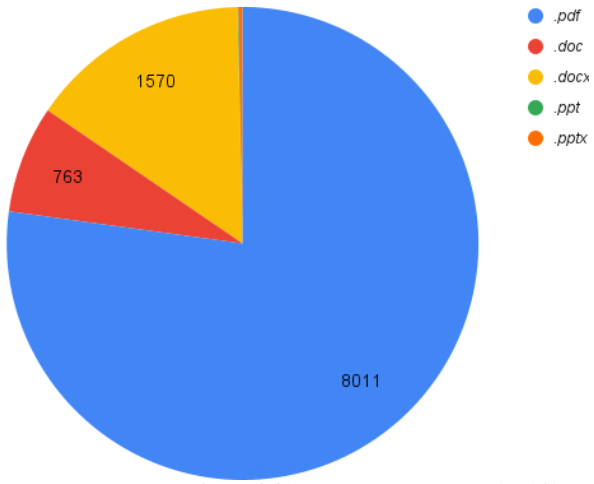

Figure 1. Representation of file types encountered while web-crawling.

As expected, most sites returned the ’200’ status codes, but we detected a few other status codes as well, as represented in **Figure** 2.

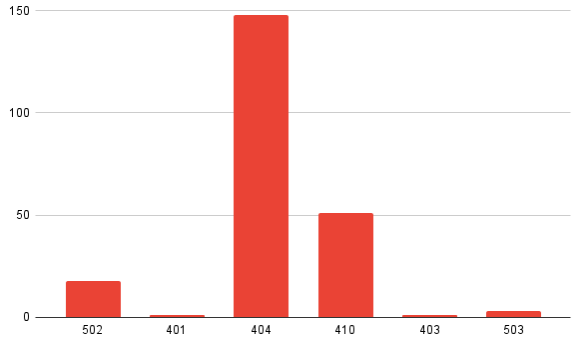

Figure 2. Number of encountered status codes

Discussion

We could potentially make a faster web-crawler. First thing to do, would be to lower the number of requests per page. As it currently stands, we make one request for page data, and another for its status code.

We also had some errors during crawling, namely SSL errors [11], which our web-crawler still tried to parse, instead of skipping. We could also optimize *robots.txt* parsing, since we are doing multiple conversions there, although that would not have a significant impact on web-crawling speed.

If we were to use multiprocessing or multiple computers while web-crawling, we’d also see a significant increase in the speed of data collection. With faster web-crawling speeds, we could also write our data into our database in batches, since single I/O operations are slow.

Another error we encountered was the occurrence of duplicate domains within sites. We have logged 33 duplicates out of 193 domains encountered. This occurred due to our duplicate checking algorithm, which decided if the domain had already been logged, by checking the database for the occurrence of domain address in order to return its id. If the database call failed, the domain was marked as not a duplicate, as nothing was returned. In this regard 33 errors out of 20.000+ queries is not abysmal, but we should have definitely checked this, or developed another way to avoid duplicates entirely.

Finally, by refactoring the recursive *run* function into an iterative one, we would avoid the maximum recursion depth error, which ultimately ended our crawl prematurely.

References

[1] PostgreSQL. https://www.postgresql.org/. Accessed: 2022-03-27.

[2] Anaconda. https://www.anaconda.com/. Accessed: 2022-03-27.

[3] Project Jupyter. https://jupyter.org/. Accessed: 2022-03-27.

[4] Selenium. https://www.selenium.dev/. Accessed: 2022-03-27.

[5] Beautiful Soup. https://beautiful-soup-4.readthedocs.io/en/latest/. Accessed: 2022-03-27.

[6] urllib. https://docs.python.org/3/library/urllib.html. Accessed: 2022-03-27.

[7] Psycopg. https://www.psycopg.org/. Accessed: 2022-03-27.

[8] Pandas. https://pandas.pydata.org/. Accessed: 2022-03-27.

[9] Threading. https://docs.python.org/3/library/threading.html. Accessed: 2022-03-27.

[10] Expert Verilog and Clifford Cummings. Simulation and synthesis techniques for asynchronous fifo design. 01 2002.

[11] Sivasamy Kaliappan. What are ssl certificate errors. https://sematext.com/blog/ssl-certificate-error/. Accessed: 2022-03-27.