Introduction to Web Science

Assignment 5

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: November 30, 2016, 10:00 a.m.
Tutorial on: December 2, 2016, 12:00 p.m.

Please look at the lessons 1) **Dynamic Web Content** & 2) **How big is the Web?**

For all the assignment questions that require you to write code, make sure to include the code in the answer sheet, along with a separate python file. Where screen shots are required, please add them in the answers directly and not as separate files.

Team Name: Tango

1. Mariya Chkalova
   mchkalova@uni-koblenz.de
2. Arsenii Smyrnov
   smyrnov@uni-koblenz.de
3. Simon Schauß
   sschauss@uni-koblenz.de
4. Lukas Härtel
   lukashaertel@uni-koblenz.de

1 Creative use of the Hyptertext Transfer Protocol (10 Points)

HTTP is a request response protocol. In that spirit a client opens a TCP socket to the server, makes a request, and the server replies with a response. The server will just listen on its open socket but cannot initiate a conversation with the client on its own.

However you might have seen some interactive websites which notify you as soon as something happens on the server. An example would be Twitter. Without the need for you to refresh the page (and thus triggering a new HTTP request) they let you know that there are new tweets available for you. In this exercise we want you to make sense of that behaviour and try to reproduce it by creative use of the HTTP protocol.

Have a look at `server.py`[1] and `webclient.html` (which we provide). Extend both files in a way that after `webclient.html` is servered to the user the person controlling the server has the chance to make some input at its commandline. This input should then be send to the client and displayed automatically in the browser without requiring a reload. For that the user should not have to interact with the webpage any further.

1.1 webclient.html

```html
<html>
<head>
        <title>Abusing the HTTP protocol - Example</title>
</head>
<body>
        <h1>Display data from the Server</h1>
        The following line changes on the servers command line
        input: <br>
        <span id="response" style="color:red">
                This will be replaced by messages from the server
        </span>
</body>
</html>
```

1.2 Hints:

- This exercise is more like a riddle. Try to focuse on how TCP sockets and HTTP work and how you could make use of that to achieve the expected behaviour. Once you have an idea the programming should be straight-forward.
- The Javascript code that you need for this exercise was almost completely shown in one of the videos and is available on Wikiversity.

[1] you could store the code from http://blog.wachowicz.eu/?p=256 in a file called `server.py`

- In that sense we only ask for a ”proof of concept” nothing that would be stable out in the wilde.
    - In particular, don’t worry about making the server uses multithreading. It is ok to be blocking for the sake of this exercise.
- Without use of any additional libraries or AJAX framework we have been able to solve this with 19 lines of Javascript and 11 lines of Python code (we provide this information just as a way for you to estimate the complexity of the problem, don’t worry about how many lines your solution uses).

Listing 1 Task 1

```
#!/usr/bin/python

import socket  # Networking support
import signal  # Signal support (server shutdown on signal receive)
import time  # Current time


class Server:
    """ Class describing a simple HTTP server objects."""

    def __init__(self, port=80):
        """ Constructor """
        self.host = ''  # <-- works on all avaivable network interfaces
        self.port = port
        self.www_dir = 'www'  # Directory where webpage files are stored

    def activate_server(self):
        """ Attempts to aquire the socket and launch the server """
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:  # user provided in the __init__() port may be unavaivable
            print("Launching HTTP server on ", self.host, ":", self.port)
            self.socket.bind((self.host, self.port))

        except Exception as e:
            print("Warning: Could not aquite port:", self.port, "\n")
            print("I will try a higher port")
            # store to user provideed port locally for later (in case 8080 fails)
            user_port = self.port
            self.port = 8080

            try:
                print("Launching HTTP server on ", self.host, ":", self.port)
                self.socket.bind((self.host, self.port))

            except Exception as e:
                print("ERROR: Failed to acquire sockets for ports ", user_port, " and 8080. ")
                print("Try running the Server in a privileged user mode.")
                self.shutdown()
                import sys
                sys.exit(1)

        print("Server successfully acquired the socket with port:", self.port)
        print("Press Ctrl+C to shut down the server and exit.")
        self._wait_for_connections()

    def shutdown(self):
        """ Shut down the server """
        try:
            print("Shutting down the server")
            s.socket.shutdown(socket.SHUT_RDWR)

        except Exception as e:
            print("Warning: could not shut down the socket. Maybe it was already closed?", e)

    def _gen_headers(self, code, poll=False):
        """ Generates HTTP response Headers. Ommits the first line ! """

        # determine response code
        h = ''
        if (code == 200):
            h = 'HTTP/1.1 200 OK\n'
        elif (code == 404):
            h = 'HTTP/1.1 404 Not Found\n'

        # write further headers
        current_date = time.strftime("%a, %d %b %Y %H:%M:%S", time.localtime())
        h += 'Date: ' + current_date + '\n'
        h += 'Server: Simple-Python-HTTP-Server\n'
        if poll:
            h += 'Content-Type: application/octet-stream\n'
            h += 'Cache-Control: no-cache\n'
        h += 'Connection: close\n\n'  # signal that the conection wil be closed after compltting the request

        return h

    def _wait_for_connections(self):
        """ Main loop awaiting connections """
        while True:
            print("Awaiting New connection")
            self.socket.listen(3)  # maximum number of queued connections

            conn, addr = self.socket.accept()
            # conn - socket to client
            # addr - clients address

            print("Got connection from:", addr)

            data = conn.recv(1024)  # receive data from client
            string = bytes.decode(data)  # decode it to string

            # determine request method  (HEAD and GET are supported)
            request_method = string.split(' ')[0]
            print("Method: ", request_method)
            print("Request body: ", string)

            # if string[0:3] == 'GET':
            if (request_method == 'GET') | (request_method == 'HEAD'):
                # file_requested = string[4:]

                # split on space "GET /file.html" -into-> ('GET','file.html',...)
                file_requested = string.split(' ')
                file_requested = file_requested[1]  # get 2nd element

                # Check for URL arguments. Disregard them
                file_requested = file_requested.split('?')[0]  # disregard anything after '?'

                if (file_requested == '/'):  # in case no file is specified by the browser
                    file_requested = '/webclient.html'  # load index.html by default
                if (file_requested == '/poll'):
                    header = self._gen_headers(200, poll=True).
```

```
                        ↪ encode()
                    conn.sendall(header)
                    while True:
                        msg = input('msg: ').encode()
                        conn.sendall(msg)

                file_requested = self.www_dir + file_requested
                print("Serving web page [", file_requested, "]")

                ## Load file content
                try:
                    file_handler = open(file_requested, 'rb')
                    if (request_method == 'GET'):  # only read the
                        ↪ file when GET
                        response_content = file_handler.read()  #
                            ↪ read file content
                    file_handler.close()

                    response_headers = self._gen_headers(200)

                except Exception as e:  # in case file was not
                    ↪ found, generate 404 page
                    print("Warning, file not found. Serving
                        ↪ response code 404\n", e)
                    response_headers = self._gen_headers(404)

                    if (request_method == 'GET'):
                        response_content = b"<html><body><p>Error
                            ↪ 404: File not found</p><p>Python
                            ↪ HTTP server</p></body></html>"

                server_response = response_headers.encode()  #
                    ↪ return headers for GET and HEAD
                if (request_method == 'GET'):
                    server_response += response_content  # return
                        ↪ additional conten for GET only

                conn.send(server_response)
                print("Closing connection with client")
                conn.close()

            else:
                print("Unknown HTTP request method:",
                    ↪ request_method)


def graceful_shutdown(sig, dummy):
    """ This function shuts down the server. It's triggered
    by SIGINT signal """
    s.shutdown()  # shut down the server
    import sys
    sys.exit(1)


###########################################################
# shut down on ctrl+c
signal.signal(signal.SIGINT, graceful_shutdown)

print("Starting web server")
s = Server(80)  # construct server object
s.activate_server()  # aquire the socket
```

2 Web Crawler (10 Points)

Your task in this exercise is to ”crawl” the `Simple English Wikipedia`. In order to execute this task, we provide you with a mirror of the Simple English Wikipedia at `141.26.208.82`.

You can start crawling from `http://141.26.208.82/articles/g/e/r/Germany.html` and you can use the `urllib` or `doGetRequest` function from the last week’s assignment.

Given below is the strategy that you might adopt to complete this assignment:

1. Download `http://141.26.208.82/articles/g/e/r/Germany.html` and store the page on your file system.
2. Open the file in python and extract the local links. (Links within the same domain.)
3. Store the file to your file system.
4. Follow all the links and repeat steps 1 to 3.
5. Repeat step 4 until you have downloaded and saved all pages.

2.1 Hints:

- Before you start this exercise, please have a look at Exercise 3.
- Make really sure your crawler doesn’t follow external urls to domains other than `http://141.26.208.82`. In that case you would start crawling the entire web
- Expect the crawler to run about 60 Minutes if you start it from the university network. From home your runtime will most certainly be even longer.
- It might be useful for you to have some output on the crawlers commandline depicting which URL is currently being fetched and how many URLs have been fetched so far and how many are currently on the queue.
- You can (but don’t have to) make use of breadth-first search.
- It probably makes sense to take over the full paths from the pages of the Simple English Wikipedia and use the same folder structure when you save the html documents.
- You can (but you don’t have to) speed up the crawler significantly if you use multithreading. However you should not use more than 10 threads in order for our mirror of Simple English Wikipedia to stay alive.

Listing 2 Task 1

```
import curses
import json
import re
import os
import time
from threading import Thread
from urllib.parse import urlparse, urljoin, urldefrag
from urllib.request import urlopen


class Crawler:
    POOL_SIZE = 10

    def __init__(self, start_url):
        self.scr = curses.initscr()
        curses.cbreak()
        curses.noecho()
        curses.curs_set(0)
        self.scr.keypad(1)
        self.start_url = urlparse(start_url)
        self.stack = set()
        self.url_to_urls = dict()
        self.error_urls = set()
        self.worker_id_to_working = dict()
        self.compiled_regex = re.compile('<a[^>]+href="([^">]+)"')

    @staticmethod
    def get(url):
        with urlopen(url) as connection:
            return connection.read()

    def extract_urls(self, payload):
        return self.compiled_regex.findall(payload)

    def crawl(self, worker_id):
        while True:
            stack_length = len(self.stack)
            if len(self.stack) == 0 and True not in self.
               ↪ worker_id_to_working.values():
                break
            elif stack_length > 0:
                self.worker_id_to_working[worker_id] = True
                url = self.stack.pop()
                try:
                    document = self.get(url)
                    relative_urls = self.extract_urls(document.
                       ↪ decode())
                    absolute_urls = list(map(lambda href: urljoin(
                       ↪ url, href), relative_urls))
                    for u in absolute_urls:
                        defraged_u = urldefrag(u).url
                        if defraged_u not in self.url_to_urls and
```

```
                           ↪ urlparse(u).netloc == self.start_url
                           ↪ .netloc:
                            self.stack.add(defraged_u)
                    self.url_to_urls.update({url: absolute_urls})
                    self.write_document(url, document)
                except Exception as e:
                    self.url_to_urls.update({url: str(e)})
                finally:
                    self.worker_id_to_working[worker_id] = False

    @staticmethod
    def write_document(url, document):
        path = './output%s' % urlparse(url).path
        os.makedirs(os.path.dirname(path), exist_ok=True)
        with open(path, 'wb') as handle:
            handle.write(document)

    def write_statistics(self):
        with open('./statistics.json', 'w') as handle:
            json.dump(self.url_to_urls, handle)

    def log(self):
        start = time.time()
        while True:
            if len(self.stack) == 0 and True not in self.
               ↪ worker_id_to_working.values():
                break
            c_urls_len = len(self.url_to_urls)
            c_stack_len = len(self.stack)
            end = time.time()
            start_end_diff = end - start
            stack_delta_per_second = c_stack_len / start_end_diff
            urls_per_second = c_urls_len / start_end_diff
            self.scr.addstr(0, 0, 'stack size: %i' % c_stack_len)
            self.scr.addstr(1, 0, 'mean stack delta second: %i' %
               ↪ int(stack_delta_per_second))
            self.scr.addstr(2, 0, 'unique urls: %i' % c_urls_len)
            self.scr.addstr(3, 0, 'mean urls per second: %i' % int
               ↪ (urls_per_second))
            self.scr.refresh()
            time.sleep(1 / 10)

    def start(self):
        try:
            self.stack.add(self.start_url.geturl())
            workers = []
            for worker_id in range(0, self.POOL_SIZE):
                thread = Thread(target=self.crawl, args=[worker_id
                   ↪ ])
                self.worker_id_to_working.update({worker_id: False })
                workers.append(thread)
                thread.start()
            logger = Thread(target=self.log)
            logger.start()
            for thread in workers:
                thread.join()
            logger.join()
            self.write_statistics()
        finally:
            curses.nocbreak()
            self.scr.keypad(0)
            curses.echo()
            curses.endwin()


if __name__ == '__main__':
    crawler = Crawler('http://141.26.208.82/articles/g/e/r/Germany.html')
    # crawler = Crawler(b'http://localhost:8000/index.html')
    crawler.start()
```

3 Web Crawl Statistics (10 Points)

If you have successfully completed the first exercise of this assignment, then please provide the following details. You may have to tweak your code in the above exercise for some of the results.

3.1 Phase I

1. Total Number of *webpages* you found.
2. Total number of links that you encountered in the complete process of crawling.
3. Average and median number of links per web page.
4. Create a *histogram* showing the distribution of links on the crawled web pages. You can use a bin size of 5 and scale the axis from 0-150.

3.2 Phase II

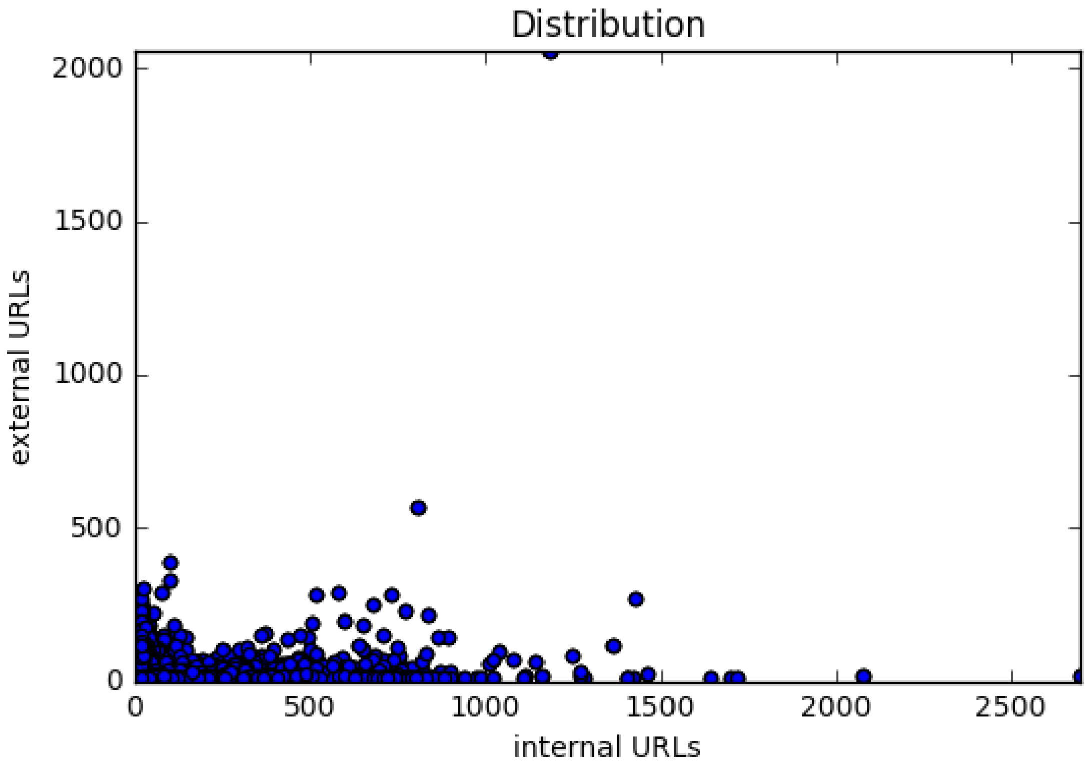
1. For every page that you have downloaded, count the number of internal links and external links.
2. Provide a *scatter plot* with number of internal links on the X axis and number of external links on the Y axis.

total number of webpages: 83895

total number of links 6042511

average number of links 72.0246856189

mean number of links 57.0



Listing 3 Task 1

```
import json
import pandas
import numpy as np
from matplotlib import pyplot as plot
from urllib.parse import urlparse

statistics_path = './statistics.json'
url = urlparse('http://141.26.208.82/articles/g/e/r/Germany.html')

def load_statistics():
    with open(statistics_path, 'r') as f:
        return json.load(f)

def filter_accessible_links(stats):
    return {k: [urlparse(i) for i in v] for k, v in stats.items()
        ↪ if isinstance(v, list)}

def filter_dead_links(stats):
    return {k: v for k, v in stats.items() if isinstance(v, str)}

statistics = load_statistics()

```

```python
accessible_links = filter_accessible_links(statistics)
dead_links = filter_dead_links(statistics)

print('total number of webpages:', len(accessible_links))

links_per_link = [(len(v)) for k, v in accessible_links.items()]

total_links = sum(links_per_link)
print('total number of links', total_links)

average_per_webpage = np.mean(links_per_link)
print('average number of links', average_per_webpage)

median_per_webpage = np.median(links_per_link)
print('average number of links', median_per_webpage)

internal_to_external = [(len([i for i in v if i.netloc == url.netloc]),
                         len([i for i in v if i.netloc != url.netloc])) for k, v in accessible_links.items()]
internal_urls = [i for i, e in internal_to_external]
external_urls = [e for i, e in internal_to_external]

print(sum(internal_urls))
print(sum(external_urls))

plot.scatter(internal_urls, external_urls)
plot.xlabel("internal URLs")
plot.ylabel("external URLs")
plot.title("Distribution")
plot.xlim((0, max(internal_urls)))
plot.ylim((0, max(external_urls)))
plot.show()
```

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment5/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of ”Web Information Retrieval”.

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.