Determining the order of requesting external resources that are required by a web page.

Angad Singh

July 2012

Objective:

To generate a priority of fetching external resources required by a web page.

Code: https://github.com/angad/webPriority

Introduction

A typical web page today has a good number of external resources which it requests after it has downloaded the page source. These external resources typically include scripts, stylesheets, advertisements, images etc. The web browser then sequentially goes through the source and sends a GET request for each of the elements.

Not all these resources are of equal importance for rendering a page. The idea here is to give each of these requests an order with which they can be sent. This requires us to attach a priority value to each external resource.

The basic idea is to follow the **execution sequence of a web browser**[1,2,3]

- Downloads the HTML document
- Starts parsing the document
- DOM is rendered, CSS is rendered in parallel
- Javascript is executed as it is seen. The browser may stop parsing the HTML when it is executing javascript.[4,5,6]
- Resources such as images, embeds and iframes are requested as and when the parser encounters them. They are requested asynchronously.

Basic Priority model:

```
SCRIPT = 1
STYLESHEET = 2
IMAGE = 3
EMBED = 4
IFRAME = 5
```

It is clear from the execution sequence that for the rendering of the DOM and CSS, the script (JS) and the stylesheet (CSS) should be requested first. Images are requested next as they may form an important part of the article and give the page its basic design.
Video, audio and other object embed tags (e.g. a flash player) require more CPU resources and are rendered with a lower priority. Some websites have advertisements using iframe tags and they can be requested towards the end. We discuss the case of Images and Advertisements in the later section.

Priority Adjustments

Image Size

One of the methods that we decide to set priority for an Image is by its size. The basic idea is that a large image (width, height) is more important as compared to a small image. A large image needs to be rendered first as it could be the central image for a blog article or a Facebook
page. We arbitrarily choose 250px as the height and width of a “medium” sized image. Any image larger than this gets a higher priority. Any image smaller than this does not get any change in its priority.
TODO: we can choose the dimensions of a “medium” sized image as the average of all the heights and widths of the images in the entire page. The images with dimensions larger than the medium

Advertisements

A large number of requests for external resources are due to the advertisements on a page. Ads can be of various types –Text, Images, JavaScript that injects an ad into the DOM, Flash embed object and IFrame.
The basic idea is that we give lesser importance to the advertisements and thus give them a higher numeric value in the priority table.
At first, we give the elements their standard priority value as described above and then we adjust the priority if we find out that the element is an advertisement.
The priority table now looks like this

```
SCRIPT = 1
STYLESHEET = 2
IMAGE = 3
EMBED = 4
IFRAME = 5
AD = 6
```

We do not care about advertisements that are Text and are downloaded as part of the page source. Similarly, for JavasScript that is downloaded as part of the source. Any attempt to put these advertisements in lower priority is likely to be counter-productive.

Identifying if a request is for an advertisement is not a trivial job. There have been previous attempts to identify and completely block ads using browser extensions that use a dictionary based approach. They have been successful and the list of request URLs and elements is regularly updated and maintained by an active community.

We refer to AdBlockPlus[7] - a popular browser extension that blocks advertisements and has a standardized filter[8] format. It can use filter lists as subscriptions[9] from external sources. AdBlockPlus was written in JavaScript that needs to be executed by the JS engine of the browser (e.g. SpiderMonkey). To reduce the dependency on external resources, we decided to write our own filter parser in Python.

We use one of the most popular subscription list called the EasyList[10] – which has both whitelisted as well as blacklisted elements. It follows the AdBlockPlus filter format. We converted the AdBlockPlus filter format to Regular Expressions against which the request URL can be matched.

TODO: There is a drawback in using our own parser for the filter list - it is difficult to successfully convert all AdblockPlus filter formats to Regular expressions. The filter also specifies additional options that cannot be covered with Regular Expressions.
Also, there are over 15000 filters to be matched per URL. A single page may have close to a 100 URLs. Currently it is a slow algorithm that matches against each URL. For future iteration, we plan to use Hashing[11] and Bloom Filter to optimize the filter-matching algorithm.

Another faster approach can be directly running the original AdBlockPlus javascript implementation using SpiderMonkey. AdBlockPlus is a Firefox extension and we can create a command line[13] interface to it using XPCOM[14] (Cross Platform component object model). So we can pass the URL to be checked for an ad to this command line interface and then read the STDOUT for the result. This will ensure that we use the latest EasyList as well as the latest optimized version of AdBlockPlus.

TODO
Page Classification

a. Video
    i. Entire page is just a video stream e.g. Netflix video player
    ii. Page has a central video with other text around it e.g. Youtube video
b. Blog
    i. Single article e.g. a TechCrunch article
    ii. List of articles e.g. TechCrunch homepage
c. News
    i. Column arrangement e.g. NyTimes, CNN
d. Wiki
    e.g. Wikipedia, Tutorial articles etc.

Sources and Resources

[1] Load and execution sequence of a web page
http://stackoverflow.com/questions/1795438/load-and-execution-sequence-of-a-web-page

[2] The Rendering Engine of a Web Browser
http://taligarsiel.com/Projects/howbrowserswork1.htm#The_rendering_engine

[3] Rendering a web page step by step
http://friendlybit.com/css/rendering-a-web-page-step-by-step/

[4] Javascript execution sequence
http://stackoverflow.com/questions/1307929/javascript-dom-load-events-execution-sequence-and-document-ready
[5]http://stackoverflow.com/questions/2734025/is-javascript-guaranteed-to-be-single-threaded
[6]http://stackoverflow.com/questions/2342974/when-does-the-browser-execute-javascript-how-does-the-execution-cursor-move

[7] AdBlockPlus http://adblockplus.org/
[8] AdBlockPlus Filters http://adblockplus.org/en/filters
[9] AdBlockPlus Subscriptions http://adblockplus.org/en/subscriptions
[10] https://easylist.adblockplus.org/en/development
[11] http://adblockplus.org/blog/investigating-filter-matching-algorithms

[12] https://developer.mozilla.org/en/Building_an_Extension
[13] https://developer.mozilla.org/en/Chrome/Command_Line
[14] https://developer.mozilla.org/en/XPCOM

Other interesting resources
https://developer.mozilla.org/en/The_life_of_an_HTML_HTTP_request
http://ejohn.org/blog/browser-page-load-performance/
http://javascript.about.com/od/hintsandtips/a/exeorder.htm

Code Documentation

The code is written in Python and was developed for Python2.7.3. It uses a number of modules such as
- HTMLParser http://docs.python.org/library/htmlparser.html
- URLParse http://docs.python.org/library/urlparse.html
- URLlib2 http://docs.python.org/library/urllib2.html

The main file is html.py - it contains the HTMLParser that parses the page and extracts out the external resources tags such as img, embed, iframe etc. It then makes a dictionary list of all these elements and attaches a priority to them based on the priority table defined above.

Using this list, the HTTP requests are constructed for each element and stored in a JSON format and written to a file using the file HTTPHeader/headerBuilder.py.

In the project, a JavaScript parser exists but is unused. It was an experimental try to extract all the external resources referenced by JavaScript - but it was eventually abandoned because most of the JavaScript itself is in external files that need to be GET and thus this program would incur unnecessary costs of getting JS files.

The EasyList module contains the easylistParser.py which essentially creates Regular expressions from the filters in the easylist folder.

```
usage: html.py [-h] [--url URL] [--sort] [--file URLFILE] [--ad]

Web Elements Priority Generator

optional arguments:
  -h, --help            show this help message and exit
  --url URL, -u URL     URL
  --sort, -s            Sort By priority
  --file URLFILE, -f URLFILE
                        URL List file name
  --ad, -a              Check resources for Ads
```

Code Structure

```
├── HTTPHeader //Creating the HTTP request header
│   ├── __init__.py
│   └── headerBuilder.py
├── JavaScript //JavaScript Parser, abandoned
│   ├── __init__.py
│   ├── js.py
│   └── jsparser.py
├── easylist //EasyList Parser, checks for advertisements
│   ├── __init__.py
│   ├── easylistParser.py
│   ├── easylist //EasyList filter lists
│   │   ├── easylist_adservers.txt
│   │   ├── easylist_general_block.txt
│   │   ├── easylist_general_hide.txt
│   │   ├── easylist_specific_block.txt
│   │   ├── easylist_specific_hide.txt
│   │   ├── easylist_thirdparty.txt
│   │   └── easylist_whitelist.txt
│   └── easytest.txt
├── html.py //Main script
├── req_feeder.py
└── url_file.txt
```

TODO

~~1. JS and CSS external resource requests~~

//Not needed, I dont want to request the external JS/CSS files

~~2. Detect advertisements and give them a lower priority~~

Done - TODO : Optimize the ad checking algorithm - use hashing / bloom filter

~~3. Prioritize Image loading based on size~~

Should the priority by a floating point?

4. Prioritize external embeds based on type

5. Guess the page type - classify the page type.
