Gryphon - An Advanced Web Crawler

Introduction

Web crawler is a program or programmed script that browses the World Wide Web in a systematic, automated manner. The structure of the WWW is a graphical structure, i.e. the links presented in a web page may be used to open other web pages.

By following the linked structure of the Web, a web crawler may traverse several new web pages starting from a webpage. A web crawler moves from page to page by the use of graphical structure of the web pages.

Web crawlers are used for a variety of purposes. Most prominently, they are one of the main components of web search engines, systems that assemble a corpus of web pages, index them, and allow users to issue queries against the index and find the web pages that match the queries. A related use is web archiving, where large sets of web pages are periodically collected and archived for posterity.

Understanding of Problem Statement

Building an automated tool/website for improving the quality of web pages. We can run it against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web apps, SEO, Website score and many more. We give the tool a URL to audit, it runs a series of audits against the page, and then it generates a report on how well the page did. From there, we can use the failing audits as indicators on how to improve the page. Each audit has a reference doc explaining why the audit is important, as well as how to fix it.

Scope

Surfing: We start at a set of well-connected initial pages and follow each of their links to new pages. Then we follow the links on those new pages to even more pages and so on, until we have a huge list of links. Even after removing exact duplicates, there might be a trillion unique URLs, and the number of individual web pages out there is growing by several billion pages per day.

Link Graph: The mathematical representation of what links to what.

We will give the user three options before clicking on ‘check now’ button:

- For Heavy JS and Ajax (dynamic)websites - we will be using Selenium at the backend.

- Medium - we will use the Scrapy module available in python.
- Basic HTML Page (without dynamic content) - we will be using requests module.

If the user does not select any of the options, then by default it will use Selenium at the backend. We will use Selenium by default as it can simulate a browser as a real person which can also validate Captcha.

Cloud Based web crawling - Using a cloud-based web crawler means that you can login to the site from any device and any location. Not limited by users or devices, you can set a crawl going from your mobile phone and export the report on your laptop.

Accessing our tool through web browsers reduces the potential for compatibility issues, runtime errors and problems with OS (Operating System) updates. Because you don't need to install anything, you can always switch to another device if your computer throws up a blue screen of death! Also, once you login the history of URLs crawled can be viewed along with their analysis.

Features:

- *Cookie Checker*: Verify cookies being used by the website, the cookie checker will scan the cookies on the website, and cookie consent verification links.
- *ADA compliance*: Check websites for:
    - Alt text in all images.
    - Color contrast for the site as per w3.org guidelines.
    - Accessibility issues to check the site markup for null tab index
- *WCAG Compliance*: They have 12-13 guidelines that are organized under 4 principles:
    - perceivable
    - operable
    - understandable, and
    - robust
- *ARIA*: This specification defines the authoring rules (author conformance requirements) for the use of Accessible Rich Internet Applications.
- *CCPA Cookie Check*: CCPA is an opt out jurisdiction for storing user's personal data like IP Address.
- *GDPR Cookie Check*: GDPR is an opt-in jurisdiction for storing user's personal data.
- Console Errors
- *Are all web-based*: meaning that you will not need to download or install anything to use them.
- *Browser Compatibility*: Android, Chrome, Firefox, IE, Opera etc
- *Errors*: Server Configuration, Content Issues, HTTP Status Codes, IETF RFCs, Script Errors.

- *Search Engine Guidelines*: Google Webmaster Guidelines, RObots.txt Standard, Bing Webmaster Guidelines, Yahoo Webmaster Guidelines
- *Web Standards*: W3C CSS/HTML Features, HTML5.
- SSL certificate checks and info about expiration date etc.
- Obfuscated JS redirects, User Agent Filtering, AJAX Loading, Infinite Scrolling Support.
- *Cloud Services*: Scrape and access data on Cloud Platform 24/7.
- *Schedule Crawling*: Schedule tasks to crawl at any specific time.
- *IP Rotation*: Automatic IP rotation to prevent IP from being blocked.
- *Point-and-Click Interface*: Anyone who knows how to browse can navigate through the interface along with the info on fixing critical issues/warnings. No coding needed.
- *Politeness*: It will not overburden the Web servers with frequent requests in a short amount of time. There will be enough delay between consecutive requests to the same server.
- Displays website score based on various factors.
- *Warnings*:
    - HTTPs pages have internal links to HTTP
    - Fix Broken Links
    - Island Pages
    - href lang defined but html lang missing
    - Canonical is a relative URL
    - URL receives both follow and nofollow internal links
    - Description Duplicates
    - Description is missing or empty
    - Logo/Page title missing (favicon)
    - Empty HTML Tags

It will display details about the crawl:

- Crawled number of pages
- Current crawling (last crawl date for a particular URL)
- Number of pages not crawled
- Total Google indexed pages
- Page Health
- URL Load Time (which scripts are taking the most time and how can we optimize it)
- SEO Score
- Total Time taken for Crawl

Database entries:

- Page Title

- Title Status
- URL
- Base URL
- URL Load Time
- Status Code
- Meta Description
- Description Status
- SSL Status
- Redirect Check
- Domain Authority Check and many more.

Dashboard:

- *Category*: Errors, Accessibility/Compatibility/Usability/Search/Standard Issues.
- *Issues*: Percentage of errors and issues shown in a progressbar.
- *Pages*: Number of pages with issues (e.g. W3C standard issue)
- *Benchmark*: Average benchmark score of the URL.

Design

The following are the aspects to be considered for the design of a crawler:

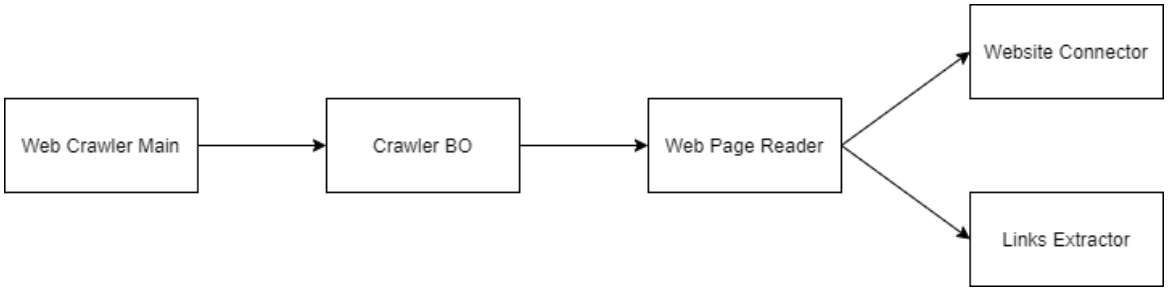
- *Politeness*: does not overburden the Web servers with frequent requests in a short amount of time.
- *Scalability:* Performance of the crawler should not degrade if there is an increase in the number of domains to be crawled.
- *Rule-abiding*: The robot rules in robot exclusion files (robots.txt) on the servers should be followed without any exceptions. The crawler should not fetch documents which it (or any other spider) is not allowed to access.
- *Speed*: The crawler should be able to gain a good speed for crawling URLs and should be able to maintain that speed even if the number of URLs increase or there is an increase in the number of domains to be crawled.
- *Priority based frontier*: Frontier is the part of a crawler which holds the URLS to be crawled. The crawler should make a sound choice regarding which URL should be crawled next. The URLs will have a weight or a rank, based on which URL will be chosen next. The URL with the highest rank will be crawled before any other URLs. Logic defines how the rank of a URL is determined. Any URL which is encountered in other URLs but not yet crawled gets a higher rank and will be shifted nearer the

front of the frontier.

- *Detection of non-functional servers*: A good crawler should detect that a Web server is offline and is not processing requests or takes a long time to process the requests.
- *Selective crawling of documents*: The crawler should crawl documents where there is a possibility to find links. i.e. documents having content-type text/html, text/javascript and others and should have logic to avoid the download of audio/ video files or files which might not contain links.
- *Load Balancing*: The task of crawling should be equally distributed among the machines based on their performance.

Simple workflow of the Crawler (a more detailed workflow is defined under #Architecture Diagram):

1. Request the HTML for the Page
2. Parse the page for every link for every link in the returned list, check if it's already in the crawled list
3. If it is then discard it, if not then add it to the list of links to be crawled.
4. This will continue until the number of links to be crawled is zero and thus all pages of said website have been crawled.

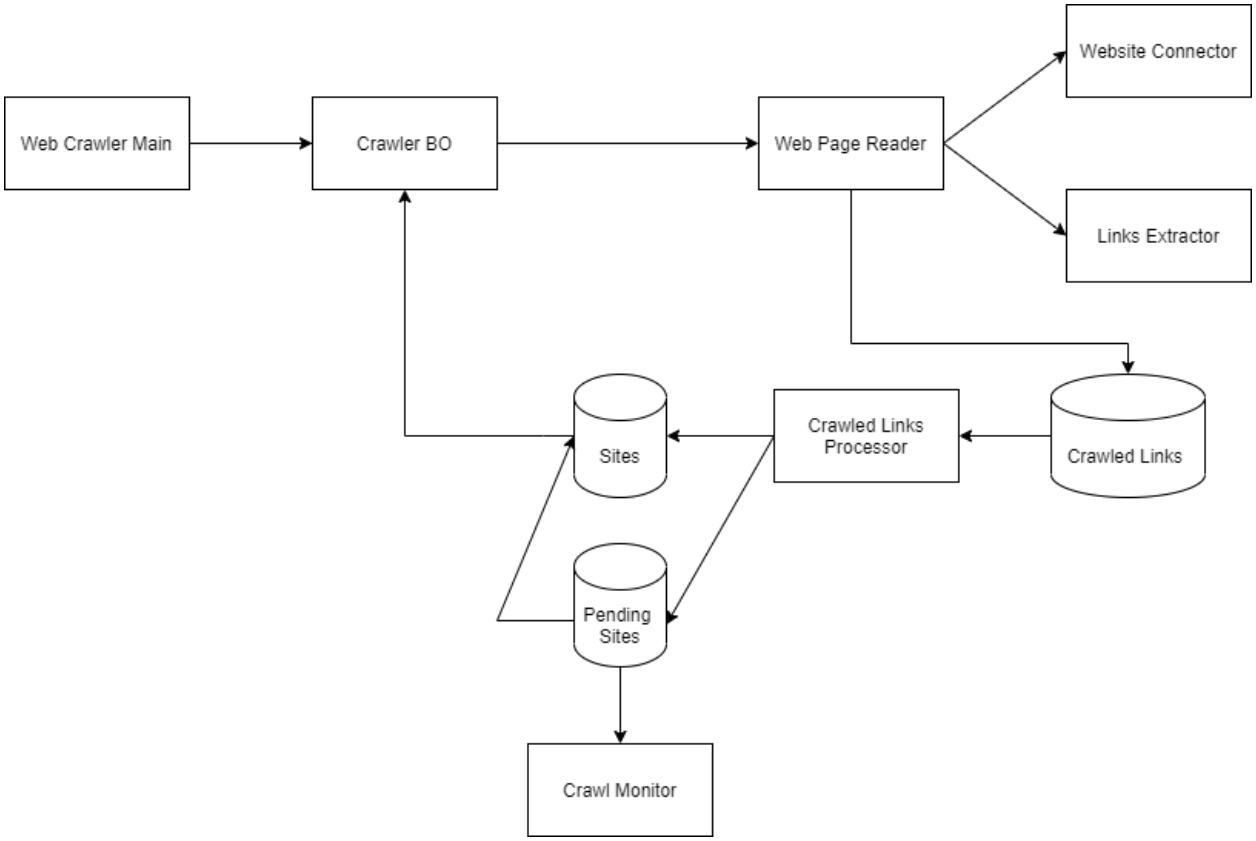


Depending on the web crawler functionality, we can add more extractors. If we want to count the number of words, create an extractor to return a map of words or add any other functionality we can always extend the functionality of your web crawler, by easily adding more extractor components, without updating and complicating the code base.
In this web crawler, as it extracts links from a website, we want to also crawl these links. To perform this efficiently, we need to make the program concurrent. Then, we can increase the number of threads to crawl the extracted links in parallel. We will add 3 channels to do this:

1. Sites : Contains the website URLs to crawl.
2. CrawledLinks : Contains the links that are found on the crawled website.

3. Pending Sites : Contains the number of pending sites to crawl.

We need Crawled Links, so that we can filter out duplicate links, before submitting it for crawling. In that way, it will not go into loops. We will maintain an in-memory map to store the list of crawled websites, to filter the duplicates. To make it production-ready, we will use a distributed database instead of in-memory map.



Architecture Diagram

There are two major design pieces for this application: The Application Code and the PostgreSQL Server. The PostgreSQL server is hosted locally and can be started using a command line shell or by navigating to /admin. Once the server is started, connections to it can be made. One connection is maintained while the application is running. Once the user exits the application, this connection is closed to the server. Application code runs on the heroku server and uses a Web Page (HTTPs) to display options to the user:

- Text Field for entering the seed website URL.

- After clicking on ‘check now’ or ‘analyze’, it will show all the web pages crawled along with their information.
- The tool will show how critical the errors or warnings are (red - very critical, yellow - warnings/low critical, green - perfect) and also show suggestions as to how we can fix them.

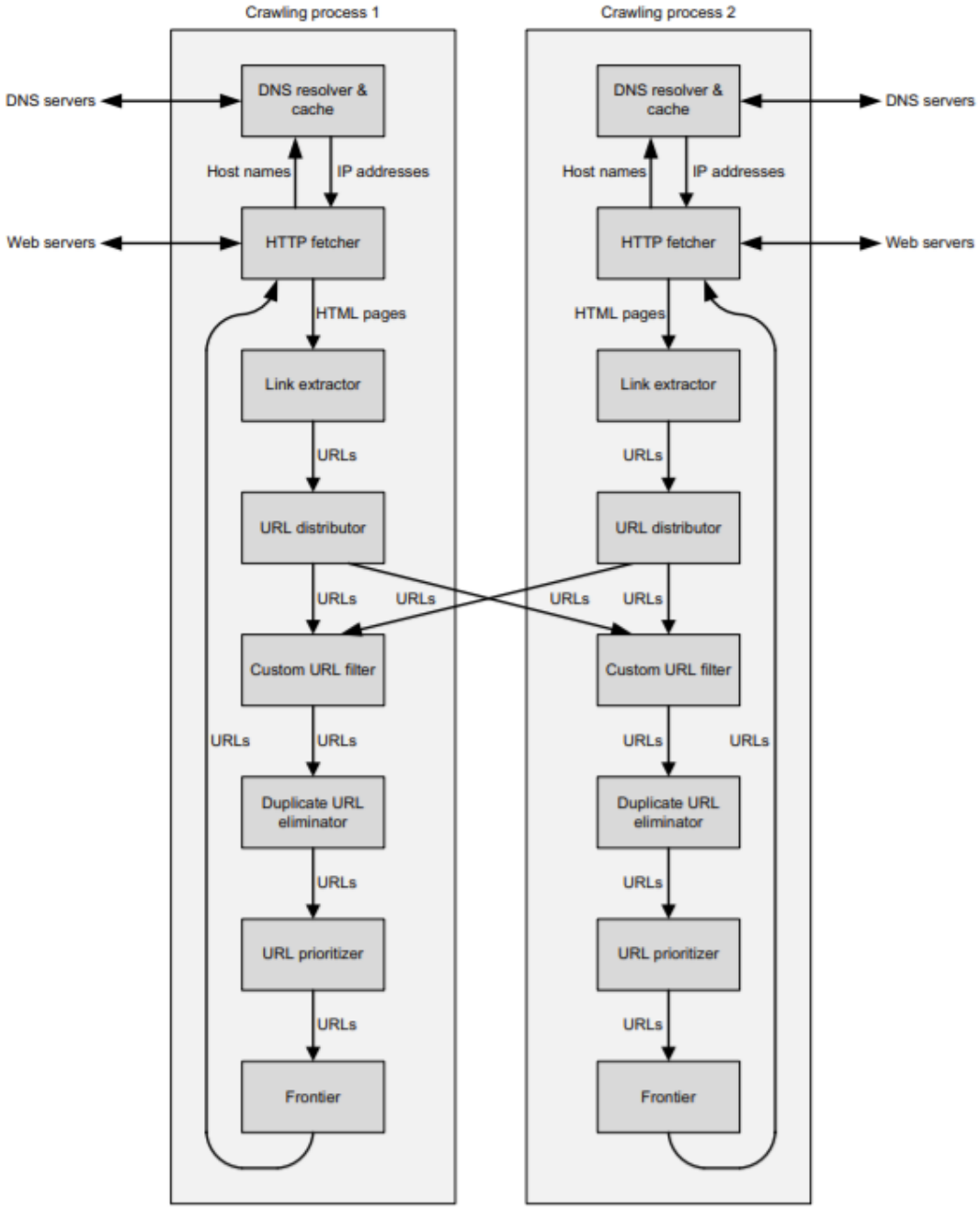
The code itself consists of a business layer and a data access layer. Only the data access layer can open and close connections to the running PostgreSQL server. The business layer contains all of the objects used to scrape the Internet for information based on which analysis is done.

The Web Crawler that we will be building will take in the base URL of a website and will try to crawl every subsequent linked page on that website. The process we’ll be following for each page will look something like this:

1. Scrape Web for Data - Selecting a starting seed URL or URLs and adding it to the frontier.
2. Pick URL from frontier.
3. Fetching the web-page corresponding to that URL.
4. Parsing that web-page to find new URL links.
5. Adding all the newly found URLs into the frontier.
6. Go to step 2 and reiterate till the frontier is empty.
7. View Database - Scraped Internal Links.
8. Analyze Scraped Data - check if there are any broken links, SSL Certificate Check etc

Below is the basic Architecture of the Web Crawler:

Gryphon - An Advanced Web Crawler



Detailed Explanation:

At the beginning of each work cycle, a worker obtains a URL from the Frontier Homepage, which dispenses URLs according to their priority and to politeness policies. The worker thread then invokes the HTTP fetcher. The fetcher first calls a DNS sub-module to resolve the host component of the URL into the IP address of the corresponding web server (using cached results of prior resolutions if possible), and then connects to the web server, checks for any robots exclusion rules (which typically are cached as well), and attempts to download the web page. If the download succeeds, the web page may or may not be stored in a repository of harvested web pages. In either case, the page is passed to the Link extractor, which parses the page's HTML content and extracts hyperlinks contained therein. The corresponding URLs are then passed to a URL distributor, which assigns each URL to a crawling process. Since most hyperlinks refer to pages on the same web site, assignment to the local crawling process is the common case. Next, the URL passes through the Custom URL filter (e.g., to exclude URLs belonging to "black-listed" sites, or URLs with particular file extensions that are not of interest) and into the Duplicate URL eliminator, which maintains the set of all URLs discovered so far and passes on only never-before-seen URLs. Finally, the URL prioritizer selects a position for the URL in the Frontier, based on factors such as estimated page importance or rate of change.

Tools and Technology Stack

A technology stack is the combination of technologies used to build an application. The two main components of any app are client-side and server-side.

Python:
The most tedious thing is server-side code, not only we have to handle multiple requests at the same time but also we have to 'analyze' the website and return it in a short amount of time. Therefore we chose python as our first preference due to its speed, performance, simplicity and also last but the most important - urllib.request module which is very essential for our needs: It offers features like cookies handling, session management, requests history and redirection management, HTTP method calls, and handling response codes and headers. Also, the combination of being free, open-source, and crossplatform: this ensures that Python is accessible to the widest range of developers possible. Important python modules that we are planning to use:

- *urllib.request*: It offers features like cookie handling, session management, requests history and redirection management, HTTP method calls, and handling response codes and headers.

- *scrapy*: It is easier to build and scale large crawling projects. It handles the requests asynchronously and it is fast. It automatically adjusts crawling speed using Auto-throttling mechanism.
- *selenium*: WebDriver can simulate a real user working with a browser, can scrape a website using a specific browser, can scrape complicated web pages with dynamic content.
- *csv*: Cases where base URL exceeds 1000 URLs then we can use csv to store the URLs and retrieve it easily for better performance.
- *postgresql*: It supports ACID and transactions, advanced indexing techniques

Django:
Web framework written in python i.e Django. Python, as it is famous for simplicity, also supports asynchronous code which will be helpful for analysis of multiple webpages at the same time. Some features of Django framework which made us choose this are: AJAX support, Free API, URL routing, Easy Database Migrations, Session handling, HTTP libraries and templating libraries, Code Layout (you can plug new capabilities by using applications), Default Admin section and more.
One of the nicest advantages of Django is that it can handle traffic and mobile app API usage of more than 400 million+ users helping maximize scalability and minimize web hosting costs. And when talking about hosting, we need to mention that the number of hosts is high and hosting price is relatively cheap and even free.

Heroku:
We can easily deploy, manage, and scale apps by following simple procedures in Heroku. Deploy apps from Git, CI systems, or GitHub. Heroku runs all the applications in a dynamic, secure, and smart container – Dyno. We also get to choose our preferred Python version. Manage applications from a detailed dashboard or by using a CLI. Hence, we found Heroku as the best option for hosting our web crawler.

Client-Side - Frontend:
The client-side of the web scraper, we will be using HTML5, CSS, Javascript, JQuery and Bootstrap.

Visual Studio Code:
Visual Studio Code is a powerful IDE for coding in multiple languages. It is free of cost. Its features consist of syntax highlighting, auto code indentation, matching of braces, code folding, error checking, ability to handle multiple secure remote connections, ability to remember open documents while restarting and a very powerful search and replace facility with regular

expressions. Also, it can be connected with Github so that we can directly commit the changes from the Code Editor itself.

Tools used for Project Report:
Microsoft Visio was used for the diagrams and Microsoft Word (or rather Google Docs) was used to write the project report.

References

- https://www.researchgate.net/publication/258789938_Web_Crawler_A_Review
- http://www.cs.uccs.edu/~jkalita/work/StudentResearch/PatwaMSProject2006.pdf
- https://docs.scrapy.org/en/latest/topics/spiders.html#crawlspider
- https://michaelnielsen.org/ddi/how-to-crawl-a-quarter-billion-webpages-in-40-hours/
- https://www.w3.org/WAI/standards-guidelines/wcag/