1) Architecture of Web Crawlers

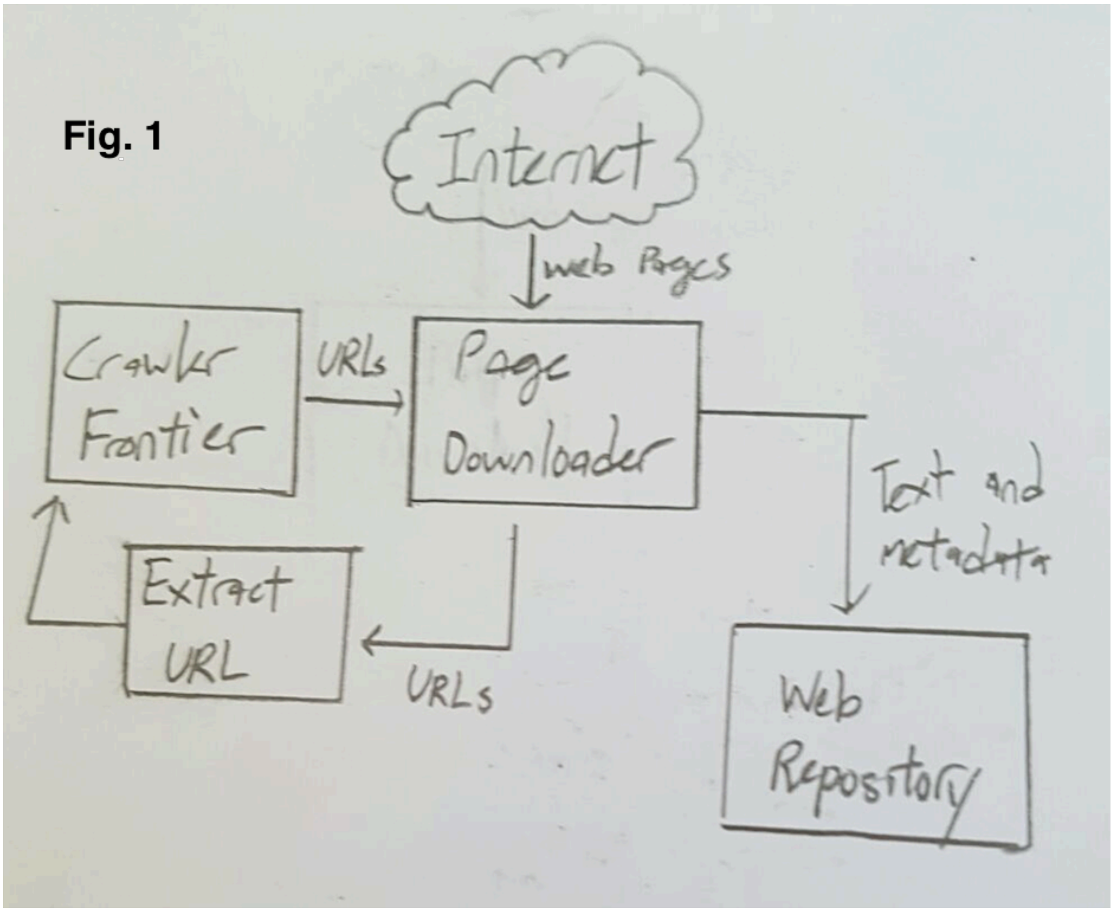

Figure 1: Overall architecture

First of all, there are some crawling policies that we will be following. This is to ensure effective and legal web crawling.

1) Selection policy: Choosing which pages to download

2) Re-visit policy: Choosing when to check for changes to the pages

3) Politeness policy: Knowing how to avoid overloading web sites

4) Parallelization policy: States how to coordinate distributed web crawlers

These policies will be there to ensure that the data we scrape from the web sites are the more relevant and more updated versions of data. The politeness policy will ensure that we will not be crossing any legal boundaries and that we will follow the various web sites' guidelines and at the same time prevent any overloading on their servers. The parallelization policy is to ensure that crawling will be more effective, by sending out multiple crawlers simultaneously and visiting multiple web sites at the same time.

According to Figure 1, we will take a look at the different components of the architecture of most web crawlers. These components are: a Page Downloader, a Crawler Frontier, a component to extract URLs, and a web repository to store text and metadata.

- **Crawler Frontier:** Contains list of unvisited URLs. User or another program sets the list with seed URLs determined. The crawler retrieves the URL and the page corresponding to the URL is fetched from the Web and the unvisited URLs extracted from the same page are added to the frontier. Cycle of fetching and extracting URLs continue until frontier is empty or some other condition causes it to stop. Order of extraction from the frontier is based on a prioritization scheme.

- **Page Downloader:** Downloads page from the Internet corresponding to the URLs that is retrieved from the crawler frontier. Page downloader requires a HTTP client for sending the HTTP request and to read the response. Should have timeout periods that are set by the client in order to ensure that it will not take unnecessary time to read large files or wait for response from slow server.

- **Web Repository:** Used to store and manage a large pool of data "objects", in case of web crawlers, the objects are web pages. The repository stores only standard HTML pages, and the crawler ignores all other media and document types. It stores the crawled pages as distinct files, and the storage manager stores the up-to-date version of every page retrieved by the crawler.

- **Extract URL:** Used to extract URLs from the current page downloaded from the Web that was retrieved from the URL from the crawler frontier. These extracted URLs are then passed to the crawler frontier and stored as the next URLs to be visited by the crawler in the future. These URLs are obtained through parsing of the current page.

2) Revised Architecture

In the above architecture, the web crawler retrieves and stores pages that are referenced by the URLs in the crawler frontier and then parses the pages to detect other URLs in the page for the web crawler. We may be able to structure our web crawler to work in this way, however in the case of our web crawler there is more information that we are required to gather from the web site's pages, i.e. company name, company contact number, company location, etc. For that, we will require the page to first pass through a parser, which will retrieve all these relevant information as well as extract URLs to other web pages, which may be relevant to the user's search requirements. For that reason, our web crawler's architecture will look similar to the diagram below.

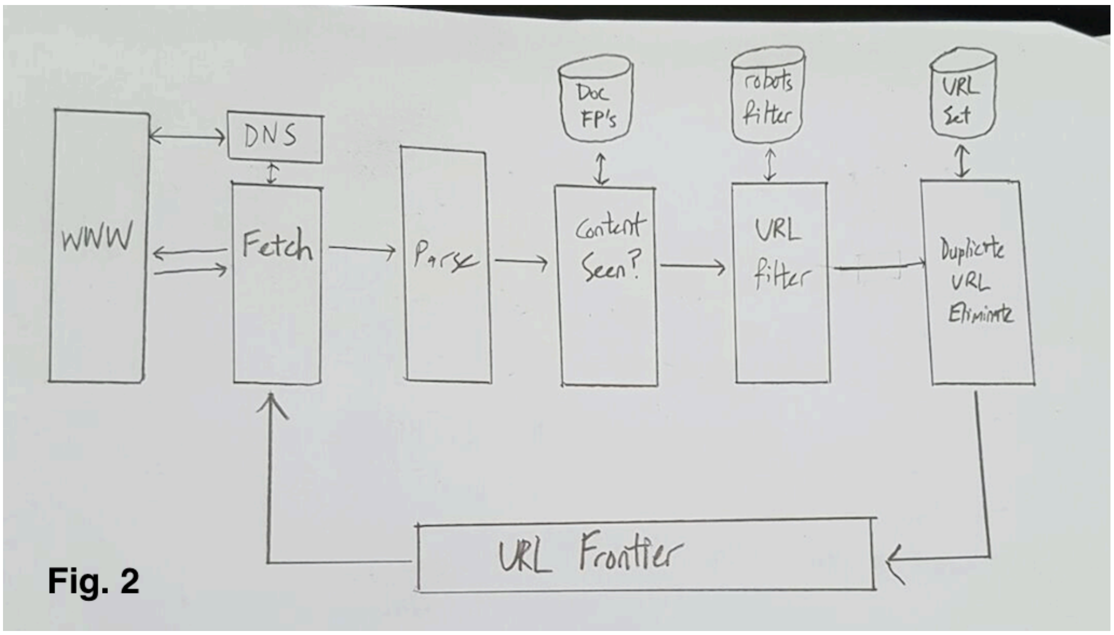

Figure 2: Our architecture for the web crawler

As you can see from Figure 2, the revised architecture has almost the same architecture as the one in Figure 1, with exception to some new additional components in the architecture.

In the new architecture, we have a component solely dedicated to handling the parsing of the web pages retrieved from the Web through the stored URLs in the URL frontier. This parser will extract the relevant information of companies, i.e. companies' name, the company's contact information, the company's location, etc. It will also extract URLs within the page to add to the crawler frontier later on.

All these relevant information will then be stored into a database. Information related to companies will be stored into a separate table from a table storing the visited web pages. We are keeping a table to store the visited URLs so that we know that these web pages have been visited before and will not have to be visited again until past expiration, which will be determined by our re-visit policy.

3) Selection of URLs

To ensure that the efficiency of our web crawlers, we should carefully choose the right metrics to base our search algorithm. This is to ensure that the web pages that our web crawler crawls through are more relevant to what the user is requesting for and that it doesn't waste any extra time and/or memory space going through web pages that are not useful for the user. The Web contains billions of web pages and we want to cut down on the amount of time and number of pages that the web crawler goes through. This also helps to ensure that the data collected will not be diluted and will be more accurate to what the user requires.

A few ordering systems that we can follow are: Breadth-first ordering, backlink ordering, PageRank ordering and random ordering.

Breadth-first ordering states that pages are crawled in the order that they are discovered, with accordance to the topic of interest. Backlink ordering visits pages with the highest number of known links to them. PageRank ordering visits pages according to their PageRank. Random ordering visits pages from the set of uncrawled pages that we have stored.

After some consideration, we have decided that we should go with PageRank ordering, as it would give us the "best" pages in a shorter amount of time as compared to the other ordering systems.

PageRank is based on three things: content, popularity and connectivity. Metrics based on connectivity have the advantages that they do not require information that is not easily accessible (for example page popularity data), and that they are easy to compute, so they scale well to even very large page collections. They also require retrieving only the links on each page, not the full-page contents. PageRank of a page is high if many pages with a high PageRank contain links to it, and a page containing few outgoing links contributes more weight to the pages it links to than a page containing many outgoing links. A possible problem with this is however that Google no longer provides the PageRank value of a web page and gathering a value for each web page may be challenging task.

4) Current Challenges in Web Crawling

During the process of web crawling, there are some things that we should take into consideration in order to ensure that our web crawler retrieves the best data and also not cross any legal boundaries.

One example would be that we cannot overload any web servers and this may be challenging, as the distribution of web pages on web servers is non-uniform. We also need to avoid web spam, as these are not only useless but consume resources and often spoil the collected content.

We also should be wary of detecting malicious web sites, as these web sites may be harmful to our web crawler. We should build a list of such web sites and inform a user about the potential threat of visiting such web sites.

Content providers possess some control over crawlers via special protocols to define access to parts of a site and direct banning of agents hitting a site too often. Every web site contains a robots.txt which dictates what can or cannot be crawled.