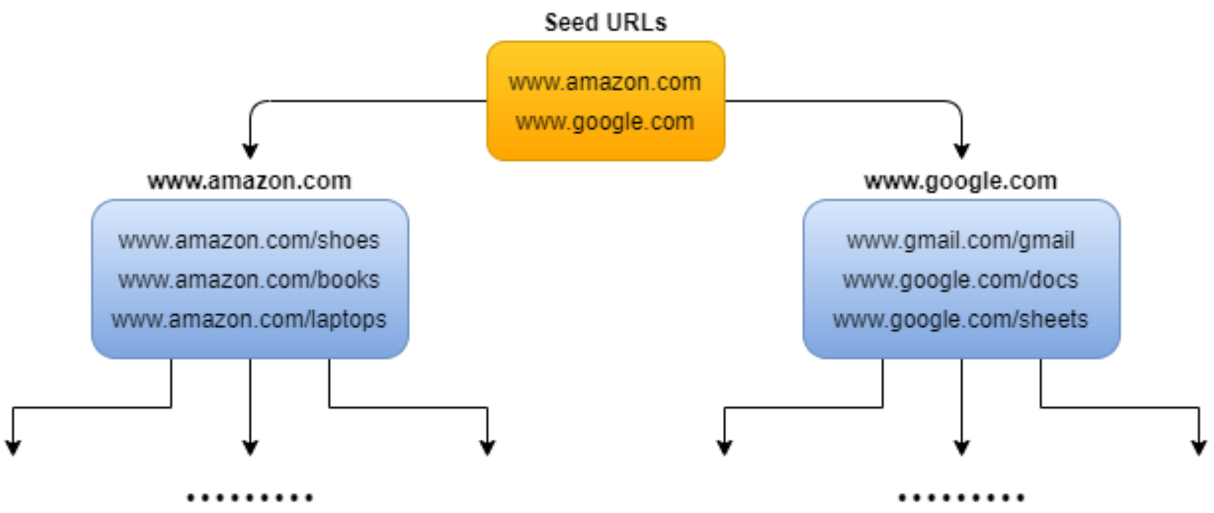
Web Crawler System Design

What is Web Crawler?

Web crawler (also known as a spider) is a system for downloading, storing, and analyzing web pages. It performs the task of organizing web pages that allow users to easily find information. This is done by collecting a few web pages and following links to gather new content.

Web crawlers are used for a variety of purposes, but they are commonly used as a key component of search engines. These search engines compile a collection of web pages, index them, and allow users to search the index for pages that match their queries.



Use cases of web crawler

Search engine indexing: Search engines uses web crawlers to collect web pages and generate a local index. For example, Google uses Googlebot web crawler.

Web archiving: We can use web crawlers for collecting web-based information and storing it for future use. The US Library of Congress and the EU web archive often use crawlers for this purpose.

Web monitoring: Web crawlers can monitor the internet for copyright and trademark violations. For example, Digimarc uses crawlers to identify and report pirated activities.

Requirements of the System

A simple design of a web crawler should have following functionalities:

- Given a set of URLs, visit the URL and store the web page.
- Now, extract URLs in these web pages.
- Append new URLs extracted to the list of URLs to be visited.
- Repeat the process.

System analysis and characteristics

When designing a web crawler system, we should consider several important characteristics:

- System should be scalable because there are billions of web pages that need to be crawled and analyzed. In other words, a web crawler should efficiently handle this task by using parallelization.
- Web crawler should be robust to handle various challenges like poorly formatted HTML, unresponsive servers, crashes, and malicious links.
- Web crawler should avoid making too many requests to a website within a short time interval because this can lead to DDoS attacks.
- System needs to be extensible so that it can be flexible to any future changes. For example, the ability to crawl images or music files may be required in the future.
- Web crawler should be manageable and reconfigurable i.e. it should have a good interface for monitoring crawl like statistics about hosts and pages, crawler speed, and sizes of the main data sets. This is important for monitoring the performance of the web crawler.

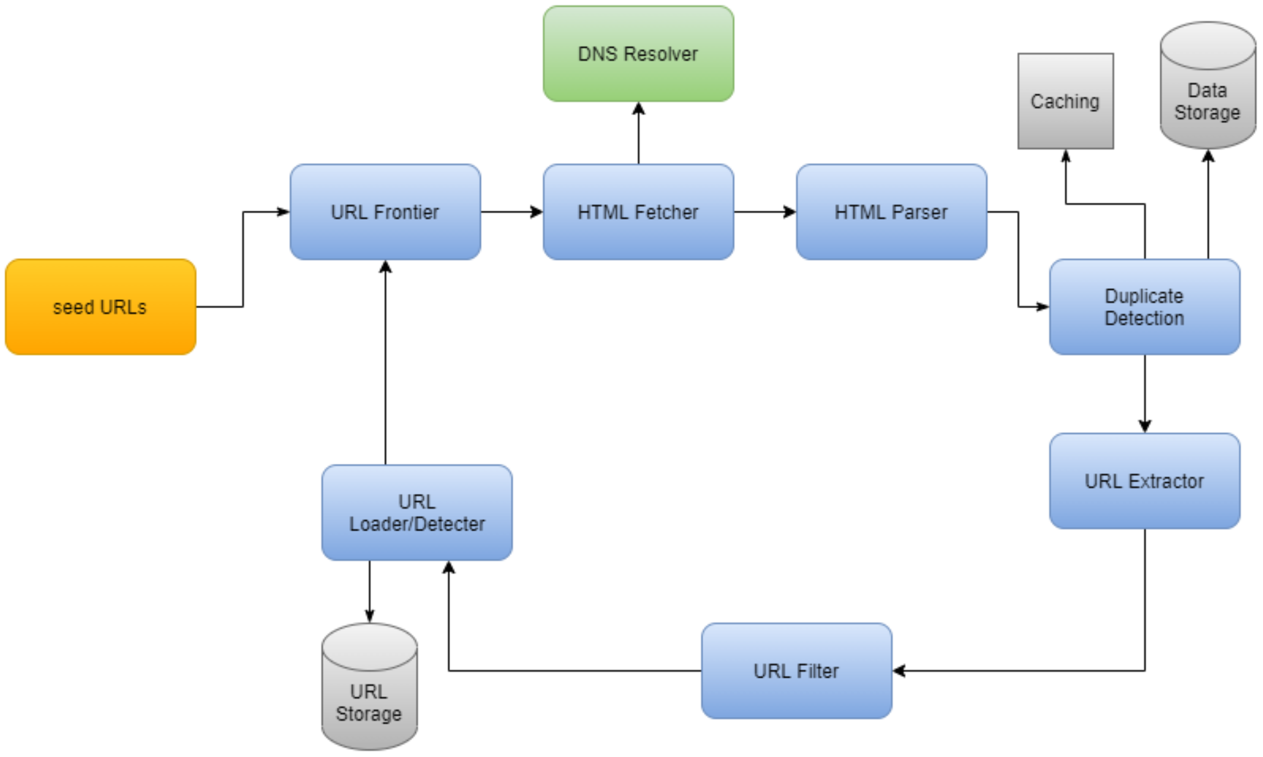
Estimation of scale and constraints

To estimate the scale of a web crawling system, we can make assumptions based on a discussion with the interviewer. For example:

- Assuming that 1 billion web pages need to be crawled every month, we can estimate the average number of queries per second (QPS) to be approximately 400 pages per second: QPS = 1,000,000,000 / 30 days / 24 hours / 3600 seconds = 400.
- We can assume that the peak value of queries per second is twice the average or 800 pages/second.
- Assuming that average web page is 500 Kb in size, we can estimate the storage required per month to be 500 TB: 1-billion-page * 500k = 500 TB storage per month.
- If we assume that the data needs to be stored for five years, then total storage needed would be 30 PB: 500 TB * 12 months * 5 years = 30 PB.

High Level Design

Once we have a clear understanding of our requirements and have made estimations on the scale, we can design a high-level architecture overview of the system. This architecture will outline the overall structure and components of the web crawler.



Let us explore the individual component of the system:

Seed URLs: To begin the crawl process, we need to provide a set of seed URLs to the web crawler. One way to do this is to use a website's domain name to crawl all of its web pages. To make our system more efficient, we should be strategic in choosing the seed URL because it can impact the number of web pages that are crawled. The selection of the seed URL can depend on various factors, such as geographical location, categories (such as entertainment, education, sports, food), content type, etc.

URL Frontier: Component that stores URLs to be downloaded is called the URL Frontier. One way to crawl the web is to use a breadth-first traversal, starting from the seed URLs. This can be implemented by using the URL Frontier as a first-in, first-out (FIFO) queue, where URLs will be processed in the order that they were added to the queue (starting with the seed URLs).

HTML Fetcher: HTML fetcher is a component that is responsible for downloading web pages corresponding to a given URL, as provided by the URL Frontier. It does this by using a network protocol such as HTTP or HTTPS. In simple words, HTML fetcher retrieves the actual web page content that needs to be analyzed and stored.

DNS Resolver: Before a web page can be downloaded, URL must be translated into an IP address. For this, HTML fetcher component initiates download process by calling the DNS Resolver. After this, DNS Resolver converts URL into the corresponding IP address, which is then used to access the web page. This process is necessary because computers communicate with each other using IP addresses.

HTML Parser: After HTML fetcher has downloaded a web page, it is important to parse, analyze, and validate the content to ensure the integrity and security of the stored data. For this, HTML Parser will check issues like poorly formatted HTML or malware that could cause problems with the storage system. Through this process, we ensure that the data being stored is of high quality.

Duplicate Detection: Studies have shown that around 30% of web pages contain duplicate content, which can lead to inefficiencies and slowdowns in the storage system if the same content is stored multiple times. To avoid this problem, we can use a data structure to check for redundancy in the downloaded content. For example, we can use MD5 hashing to compare the content of pages that have been previously seen, and check if the same hash has occurred before. This can help to identify and prevent the storage of duplicate content.

Data Storage: After web pages have been downloaded and parsed, they need to be stored in a storage system. The choice of a specific storage system will depend on the use cases for the data. For example, if we want to use the content for offline archiving or analysis, we can compress and store data with a low-cost cloud storage provider.

On the other side, if we want to use content for real-time search or query, we can store the data in a large distributed database, such as HDFS, Google's BigTable, or Apache Cassandra. These types of databases are designed to support querying and searching. But the key thing is: Regardless of the storage system we choose, it is important to ensure that we have enough space to store large amounts of data that is likely to be collected.

Caching: To improve the efficiency of web crawler, we can use a cache to store recently processed URLs. This allows us to quickly look for a URL in the cache rather than crawling the web to find it again. The type of cache will depend on the specific use case for the web crawler.

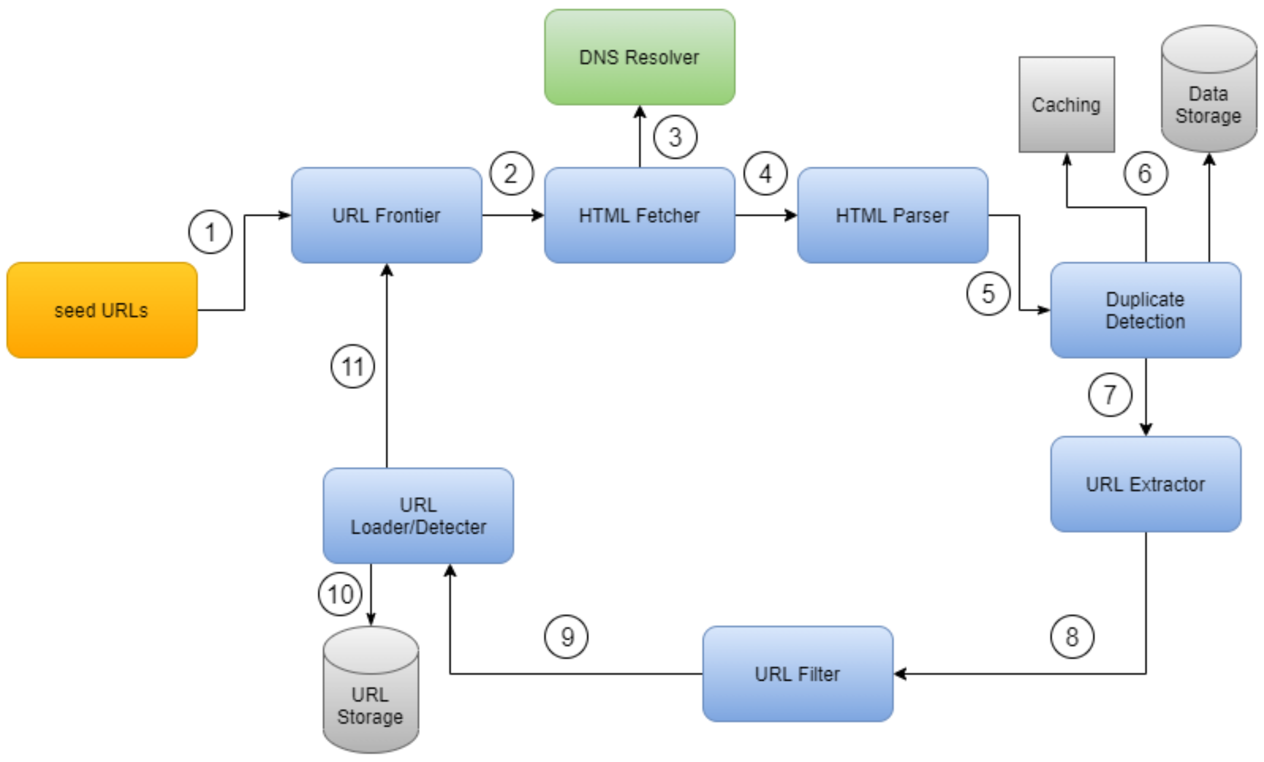
URL Extractor: URL Extractor parse and extract links from HTML pages. Once the links have been extracted, they are filtered and added to the URL Frontier. This process allows web crawler to expand the scope of its crawl by following the links on each web page and collecting new content. In other words, URL Extractor enables system to discover new content and continue expanding the collection of web pages.

URL Filter: URL filter is used to filter out unwanted content types, faulty links, and URLs from unsafe sites. This helps to ensure that the system only collects and stores high-quality, relevant content. The URL filter can be customized to meet the specific needs of the web crawler, such as by excluding certain content types or blocking access to unsafe sites. In simple words: By using a URL filter, we can improve the efficiency and effectiveness of the web crawler by limiting the amount of unnecessary or irrelevant content that is collected.

URL Detector: URL Detector is used to filter out URLs that have already been visited. This helps to prevent the system from getting stuck in an infinite loop where the same URL is processed repeatedly. There are a few different techniques that can be used to implement the URL Detector, such as Bloom Filters and Hash Tables. These techniques efficiently identify already visited URLs, so that they can be skipped in the crawling process.

URL Storage: URL storage is used to store the URLs of web pages that have already been visited. This allows the system to keep track of the URLs that have been processed, so that they can be skipped in the future if they are encountered again.

Workflow of Web Crawler System



The crawling process of a web crawler consists of several worker threads that perform repeated cycles of work. The following is a summary of the steps involved in a work cycle:

1. At the beginning of the cycle, worker thread sends seed URLs to the URL Frontier. URL Frontier then retrieves URLs according to its priorities and politeness policies.
2. HTML Fetcher module is called to fetch URLs from the URL Frontier.
3. HTML Fetcher calls DNS resolver to resolve the host address of the web server associated with the URL.
4. HTML Parser parses the HTML page and analyzes content of the page.
5. The content is validated and then passed to the duplicate detection component to check for duplicity.
6. The duplicate detection component checks the cache, and if the content is not found there, it checks the data storage to see if the content is already stored there.
7. If content is not in the storage, web page is sent to the Link Extractor,

which extracts any outgoing links from the page.

8. The extracted URLs are passed to the URL filter, which filters out unwanted URLs such as file extensions of no interest or blacklisted sites.
9. After links are filtered, they are passed to the URL Detector component.
10. URL Detector checks if a URL has already been processed and stored. If it has, no further action is needed. If URL has not been processed before, it is added to the URL Frontier to be crawled in a future work cycle.

Now URL Frontier is responsible for managing the queue of URLs that are waiting to be crawled. It assigns specific positions to the URLs in its data structure based on certain priority rules. These rules can be customized to meet the specific needs of the web crawler. For example, URL Frontier might prioritize URLs from certain domains or categories, or it might prioritize URLs that have been waiting in the queue for a longer time. By allocating positions to the URLs according to fixed priority rules, the URL Frontier helps to ensure that the web crawler is able to efficiently and effectively crawl the web.

Thanks to Chiranjeev and Navtosh for their contribution in creating the first version of this content. If you have any queries or feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!