Design of a Web Crawler

Get an overview of the building blocks and components of the web crawler system, and learn about the interaction that takes place between them during the design process of a web crawler.

We'll cover the following

Design

This lesson describes the building blocks and the additional components involved in the design and workflow of the web crawling process with respect to its requirements.

Components

Here are the details of the building blocks and the components needed for our design:

- **Scheduler**: This is one of the key building blocks that schedules URLs for crawling. It's composed of two units: a priority queue and a relational database.
  1. **A priority queue (URL frontier)**: The queue hosts URLs that are made ready for crawling based on the two properties associated wit each entry: priority and updates frequency.

2. **Relational database**: It stores all the URLs along with the two associated parameters mentioned above. The database gets populated by new requests from the following two input streams:
    - The user's *added URLs*, which include seed and runtime added URLs.
    - The crawler's *extracted URLs*.

Points to Ponder

Question 1

Can we estimate the size of the priority queue? What are the pros and cons of a centralized and distributed priority queue.

Show Answer

1 of 2

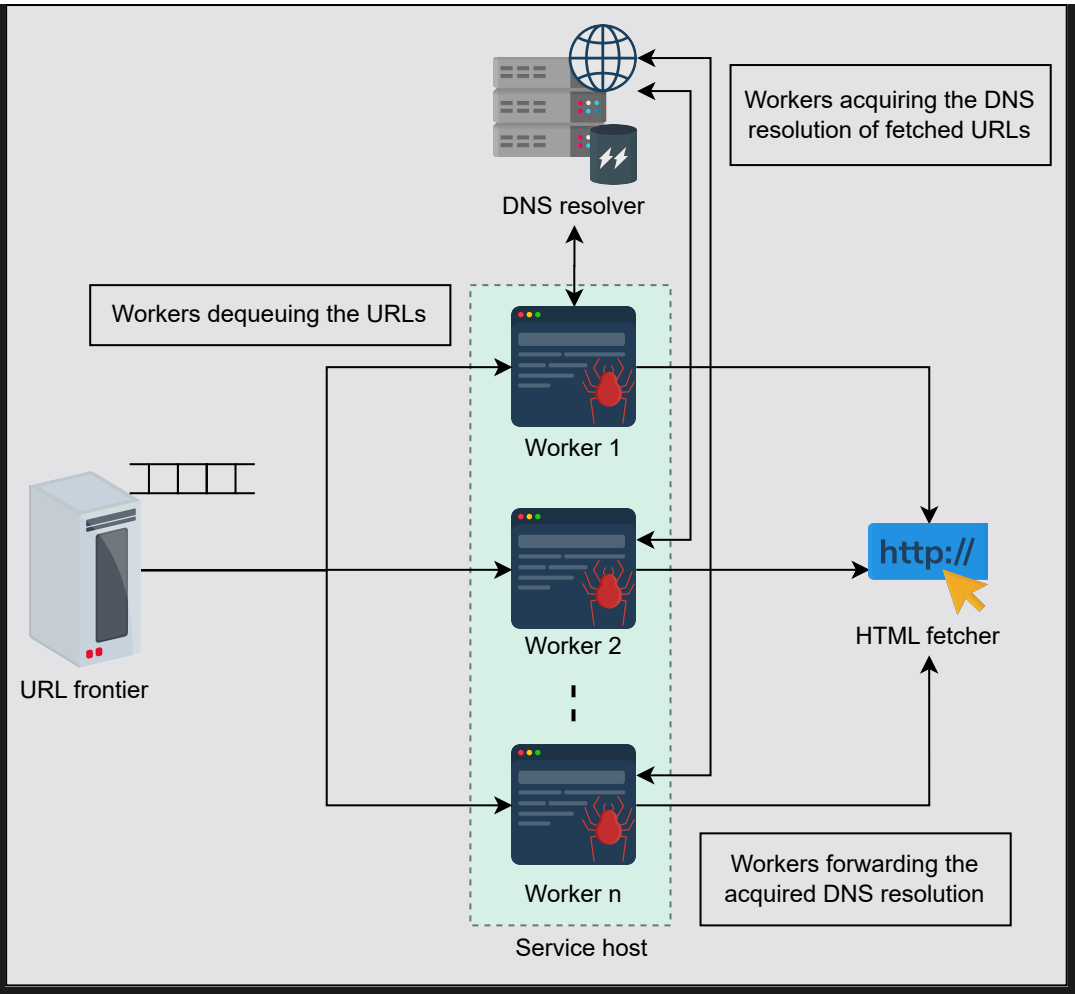
- **DNS resolver**: The web crawler needs a DNS resolver to map hostnames to IP addresses for HTML content fetching. Since DNS lookup is a time-consuming process, a better approach is to create a customized DNS resolver and cache frequently-used IP addresses within their time-to-live because they're bound to change after their time-to-live.
- **HTML fetcher**: The HTML fetcher initiates communication with the server that's hosting the URL(s). It downloads the file content based on the underlying communication protocol. We focus mainly on the *HTTP protocol* for textual content, but the *HTML fetcher* is easily extendable to other communication protocols, as is mentioned in the section on the non-functional requirements of the web crawler.

Question

How does the crawler handle URLs with variable priorities?

Show Answer

- **Service host**: This component acts as the brain of the crawler and is composed of worker instances. For simplicity, we will refer to this whole component or a single worker as a crawler. There are three main tasks that this service host/crawler performs:
    1. It handles the multi-worker architecture of the crawling operation. Based on the availability, each worker communicates with the URL frontier to dequeue the next available URL for crawling.
    2. Each worker is responsible for acquiring the DNS resolutions of the incoming URLs from the DNS resolver.
    3. Each worker acts as a gateway between the scheduler and the HTML fetcher by sending the necessary DNS resolution information to the HTML fetcher for communication initiation.



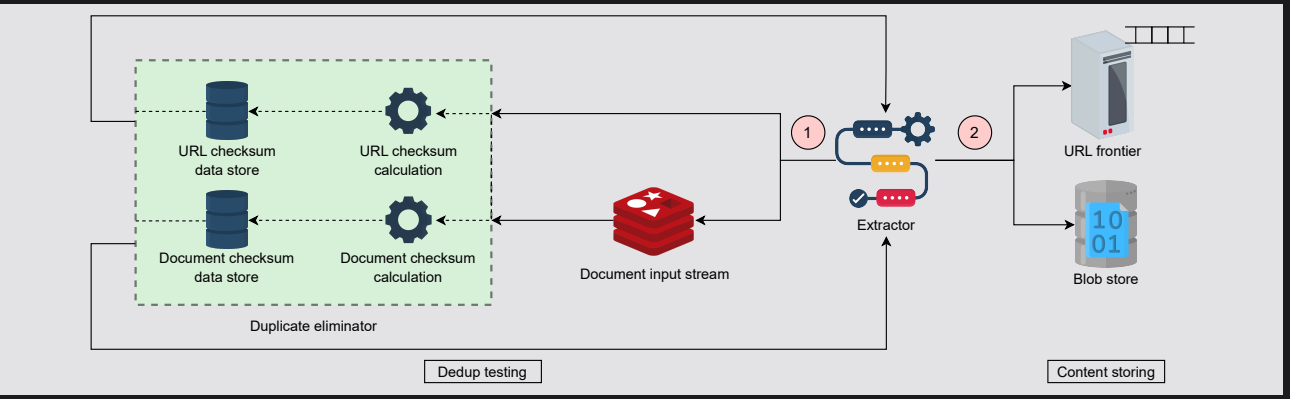
Service host interaction with other components

- **Extractor**: Once the HTML fetcher gets the web page, the next step is to extract two things from the webpage: URLs and the content. The extractor sends the extracted URLs directly and the content with the **document input stream (DIS)** to the duplicate eliminator. DIS is a cache that's used to store the extracted document, so that other components can access and process it. Over here, we can use Redis as our cache choice because of its advanced data structure functionality.

  Once it's verified that the duplicates are absent in the data stores, the extractor sends the URLs to the task scheduler that contains the URL frontier and stores the content in blob storage for indexing purposes.

- **Duplicate eliminator**: Since the web is all interconnected, the probability of two different URLs referring to the same web page or different URLs referring to various web pages having the same content is evident. The crawler needs a component to perform a dedup test to eliminate the risk exploiting resources by storing and processing the same content twice. The

duplicate eliminator calculates the checksum value of each extracted URL and compares it against the URLs checksum data store. If found, it discards the extracted URL. Otherwise, it adds a new entry to the database with the calculated checksum value.



Dedup testing in action

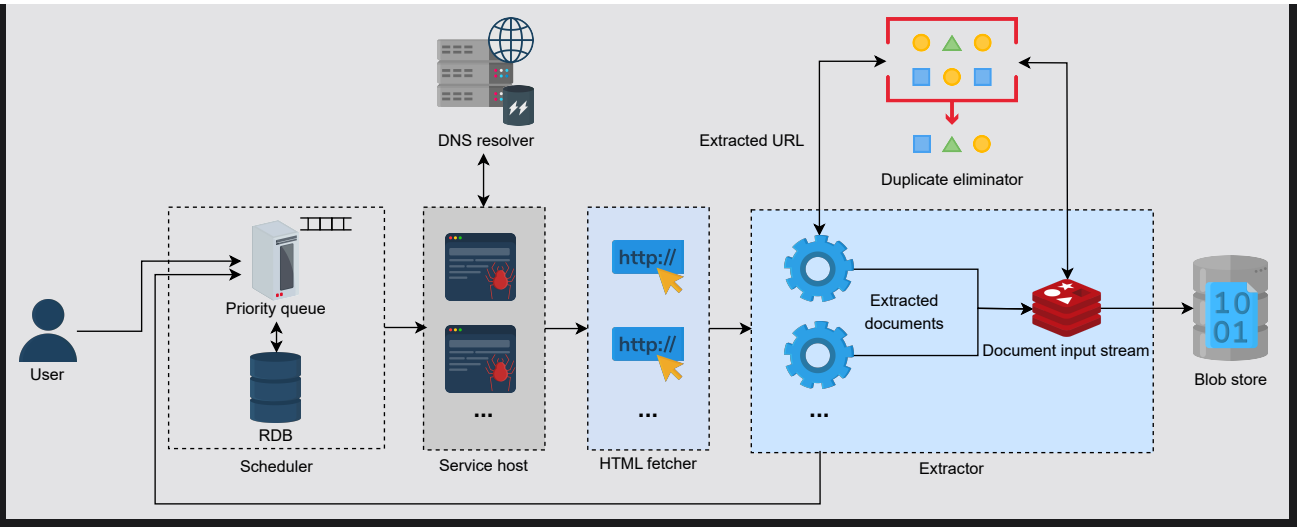
The duplicate eliminator repeats the same process with the extracted content and adds the new webpage's checksum value in the document checksum data store for future matchings.

Our proposed design for the duplicate eliminator can be made robust against these two issues:

1. By using URL redirection, the new URL can pass through the URL dedup test. But, the second stage of the document dedup wouldn't allow content duplication in the blob storage.
2. By changing just one Byte in a document, the checksum of the modified document is going to come out different than the original one.

- **Blob store**: Since a web crawler is the backbone of a search engine, storing and indexing the fetched content and relevant metadata is immensely important. The design needs to have a distributed storage, such as a blob store, because we need to store large volumes of unstructured data.

The following illustration shows the pictorial representation of the overall web crawler design:



The overall web crawler design

Workflow

1. **Assignment to a worker**: The crawler (service host) initiates the process by loading a URL from the URL frontier's priority queue and assigns it to the available worker.
2. **DNS resolution**: The worker sends the incoming URL for DNS resolution. Before resolving the URL, the DNS resolver checks the cache and returns the requested IP address if it's found. Otherwise, it determines the IP address, sends it back to the worker instance of the crawler, and stores the result in the cache.

Point to ponder!

Point to Ponder

Question

Can we use DFS instead of BFS?

Show Answer

3. **Communication initiation by the HTML fetcher**: The worker forwards the URL and the associated IP address to the HTML fetcher, which initiates the communication between the crawler and the host server.
4. **Content extraction**: Once the worker establishes the communication, it extracts the URLs and the HTML document from the web page and places the document in a cache for other components to process it.
5. **Dedup testing**: The worker sends the extracted URLs and the document for dedup testing to the duplicate eliminator. The duplicate eliminator calculates and compares the checksum of both the URL and the document with the checksum values that have already been stored.

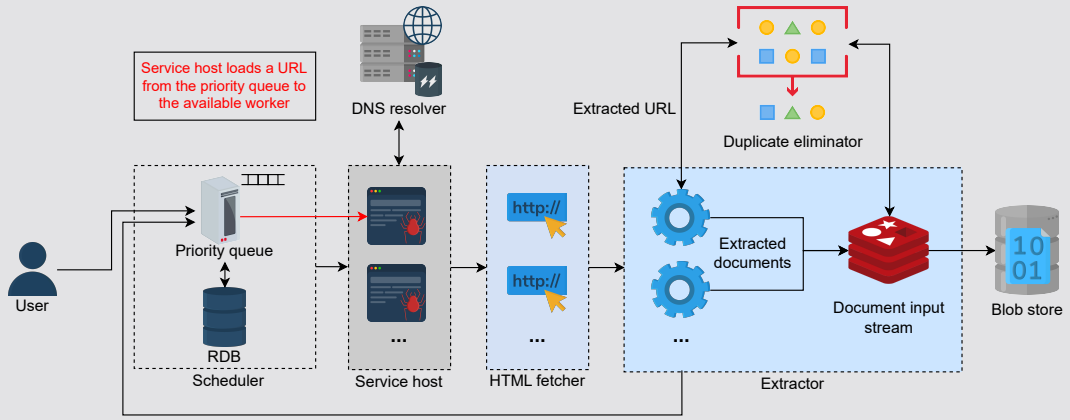
   The duplicate eliminator discards the incoming request in case of a match. If there's no match, it places the newly-calculated checksum values in the respective data stores and gives the go-ahead to the extractor to store the content.
6. **Content storing**: The extractor sends the newly-discovered URLs to the scheduler, which stores them in the database and sets the values for the priority and recrawl frequency variables.

   The extractor also writes the required portions of the newly discovered document—currently in the DIS—into the database.
7. **Recrawling**: Once a cycle is complete, the crawler goes back to the first point and repeats the same process until the URL frontier queue is empty.

   The URLs stored in the scheduler's database have priority and periodicity assigned to them. Enqueuing new URLs into the URL frontier depends on these two factors.

> **Note:** Because of multiple instances of each service and microservices architecture, our design can make use of client-side load balancing (see Client-side Load Balancer for Twitter).

The following slideshow gives a detailed overview of the web crawler workflow:



Assignment to a worker

1 of 7

Recrawling is the process of revisiting and updating indexed pages. Keeping this in mind, try to answer the following questions:

1. Why do we need to recrawl?
2. How do we decide when to recrawl?

Want to know the correct answer?

Why and when do we need to recrawl?