Requirements of a Web Crawler's Design

Learn about the design requirements of a web crawler.

We'll cover the following

Requirements

Let’s highlight the functional and non-functional requirements of a web crawler.

Functional requirements

These are the functionalities a user must be able to perform:

- **Crawling**: The system should scour the WWW, spanning from a queue of seed URLs provided initially by the system administrator.

Points to Ponder

Question 1

Where do we get these seed URLs from?

Show Answer

1 of 3

- **Storing**: The system should be able to extract and store the content of a URL in a blob store. This makes that URL and its content processable by the search engines for indexing and ranking purposes.
- **Scheduling**: Since crawling is a process that's repeated, the system should have regular scheduling to update its blob stores' records.

Non-functional requirements

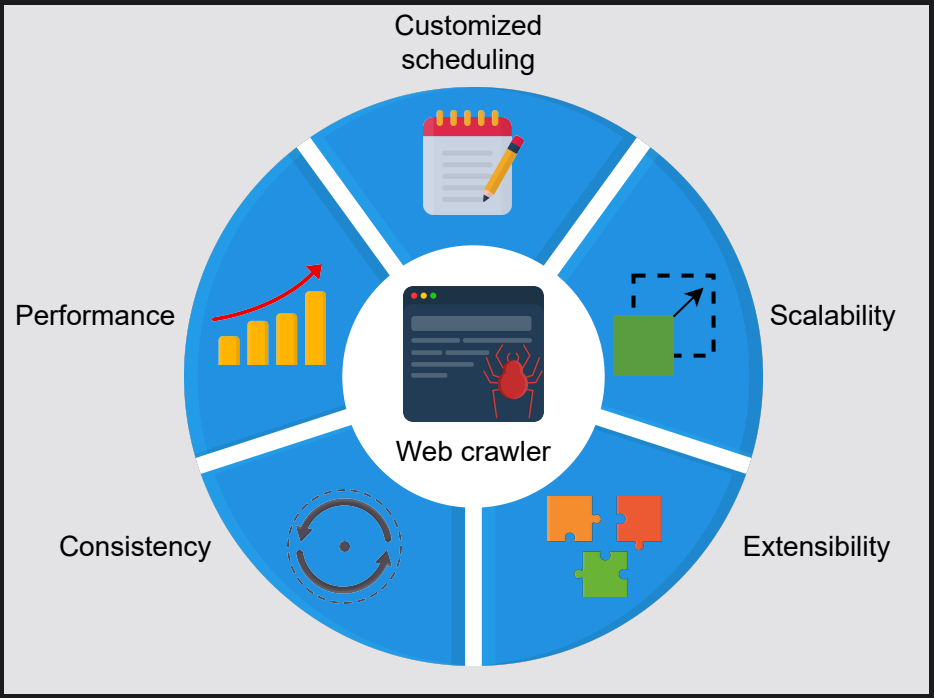
- **Scalability**: The system should inherently be distributed and multithreaded, because it has to fetch hundreds of millions of web documents.
- **Extensibility**: Currently, our design supports HTTP(S) communication protocol and text files storage facilities. For augmented functionality, it should also be extensible for different network communication protocols, able to add multiple modules to process, and store various file formats.
- **Consistency**: Since our system involves multiple crawling workers, having data consistency among all of them is necessary.

  In the general context, data consistency means the reliability and accuracy of data across a system or dataset. In the web crawler's context, it refers to the adherence of all the workers to a specific set of rules in their attempt t generate consistent crawled data.
- **Performance**: The system should be smart enough to limit its crawling to domain, either by time spent or by the count of the visited URLs of that

domain. This process is called **self-throttling**. The URLs crawled per second and the throughput of the content crawled should be optimal.

Tip

- **Improved user interface—customized scheduling**: Besides the default recrawling, which is a functional requirement, the system should also support the functionality to perform non-routine customized crawling on the system administrator's demands.



The non-functional requirements of the web crawler system

Resource estimation

We need to estimate various resource requirements for our design.

Assumptions

These are the assumptions we'll use when estimating our resource requirements:

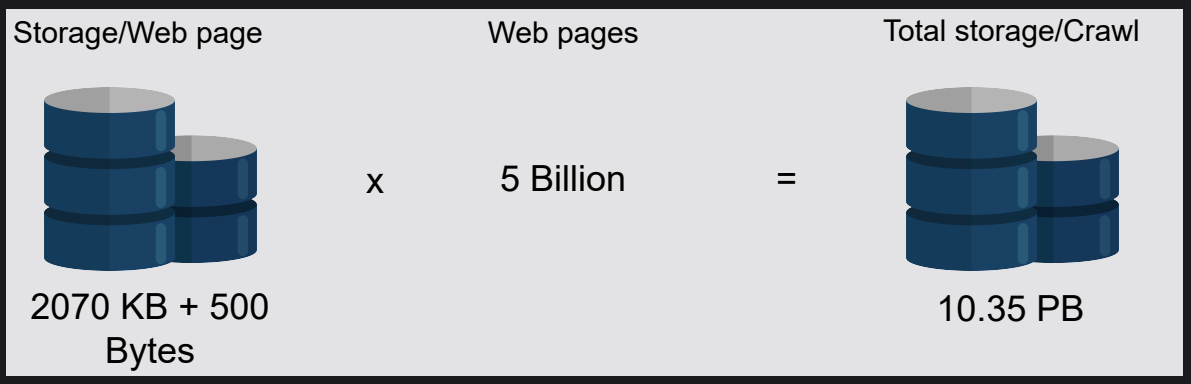
- There are a total of 5 billion web pages.
- The text content per webpage is 2070 KB.

- The metadata for one web page is 500 Bytes.

Storage estimation

The collective storage required to store the textual content of 5 billion web pages is:

$$Total\ storage\ per\ crawl = 5\ Billion \times (2070\ KB + 500B) = 10.35PB$$



The total storage required by the web crawler system

Traversal time

Since the traversal time is just as important as the storage requirements, let's calculate the approximate time for one-time crawling. Assuming that the average HTTP traversal per webpage is 60 ms, the time to traverse all 5 billion pages will be:

$$Total\ traversal\ time = 5\ Billion \times 60\ ms = 0.3\ Billion\ seconds = 9.5\ years$$

It'll take approximately 9.5 years to traverse the whole Internet while using one instance of crawling, but we want to achieve our goal in one day. We can accomplish this by designing our system to support multi-worker architecture and divide the tasks among multiple workers running on different servers.

Number of servers estimation for multi-worker architecture

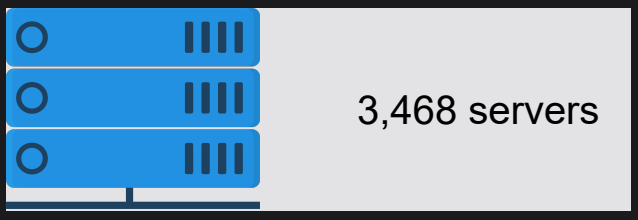
Let's calculate the number of servers required to finish crawling in one day. Assume that there is only one worker per server.

$$No.\ of\ days\ required\ by\ 1\ server\ to\ complete\ the\ task = 9.5\ years \times 365\ days \approx 3468\ days$$

One server takes 3,468 days to complete the task.

How many servers would we need to complete this same task in one day?

We would need 3,468 servers to complete the same task in just one day.



The number of servers required for the web crawler system

Case for multi-threaded server!

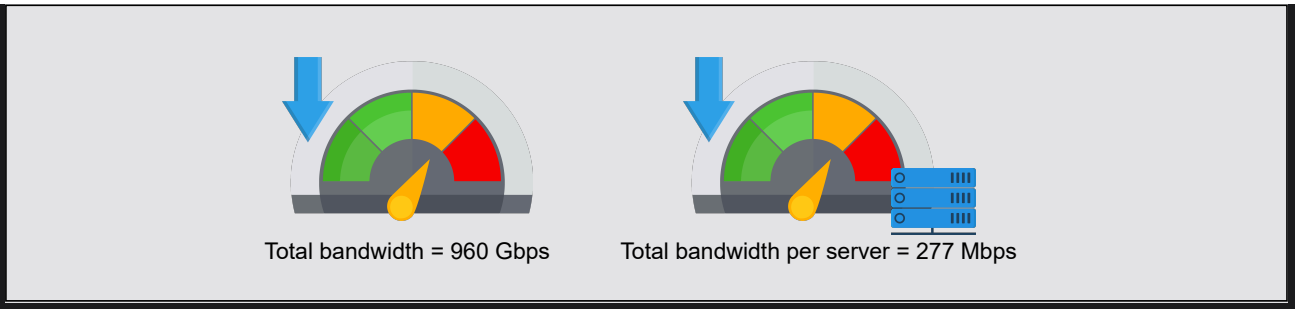
Bandwidth estimation

Since we want to process 10.35PB of data per day the total bandwidth required would be:

$$\frac{10.35PB}{86400sec} \approx 120GB/sec \approx 960Gb/sec$$

$960Gb/sec$ is the total required bandwidth. Now, assume that the task is distributed equally among $3468\ servers$ to accomplish the task in one day. Thus, the per server bandwidth would be:

$$\frac{960Gb/sec}{3468\ server} \approx 277Mb/sec\ per\ server$$



The total bandwidth required for the web crawler system

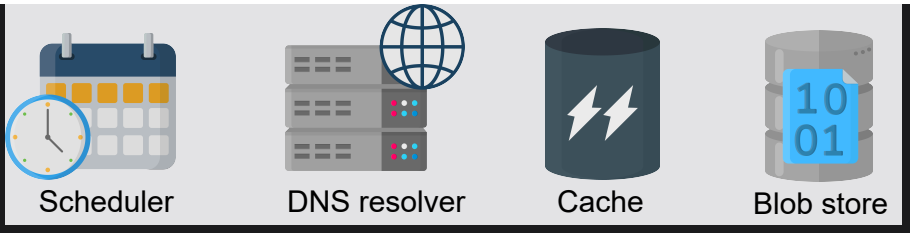
Let's play around with the initial assumptions and see how the estimates change in the following calculator:

Estimates Calculator for the Web Crawler

Number of Webpages	5	Billion
Text Content per Webpage	2070	KB
Metadata per Webpage	500	Bytes
Total Storage	*f* 10.35	PB
Total Traversal Time on One Server	*f* 9.5	Years
Servers Required to Perform Traversal in One Day	*f* 3468	Servers
Bandwidth Estimate	*f* 958.33	Gb/sec

Building blocks we will use

Here is the list of the main building blocks we'll use in our design:

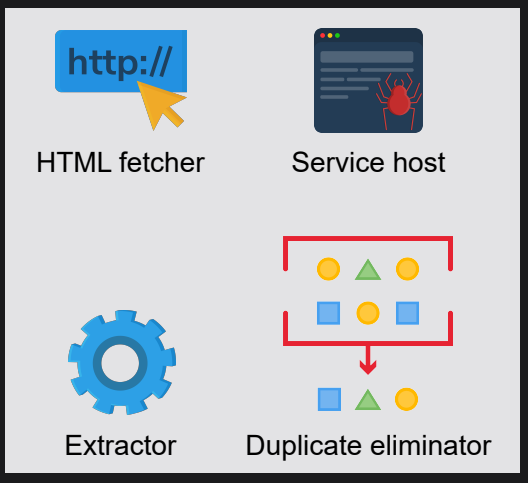


Building blocks in high-level design

- **Scheduler** is used to schedule crawling events on the URLs that are stored in its database.
- **DNS** is needed to get the IP address resolution of the web pages.
- **Cache** is utilized in storing fetched documents for quick access by all the processing modules.
- **Blob store**'s main application is to store the crawled content.

Besides these basic building blocks, our design includes some additional components as well:

- The **HTML fetcher** establishes a network communication connection between the crawler and the web hosts.
- The **service host** manages the crawling operation among the workers.
- The **extractor** extracts the embedded URLs and the document from the web page.
- The **duplicate eliminator** performs dedup testing on the incoming URLs and the documents.



The components in a high-level design

In the next lesson, we'll focus on the high-level and detailed design of a web crawler.