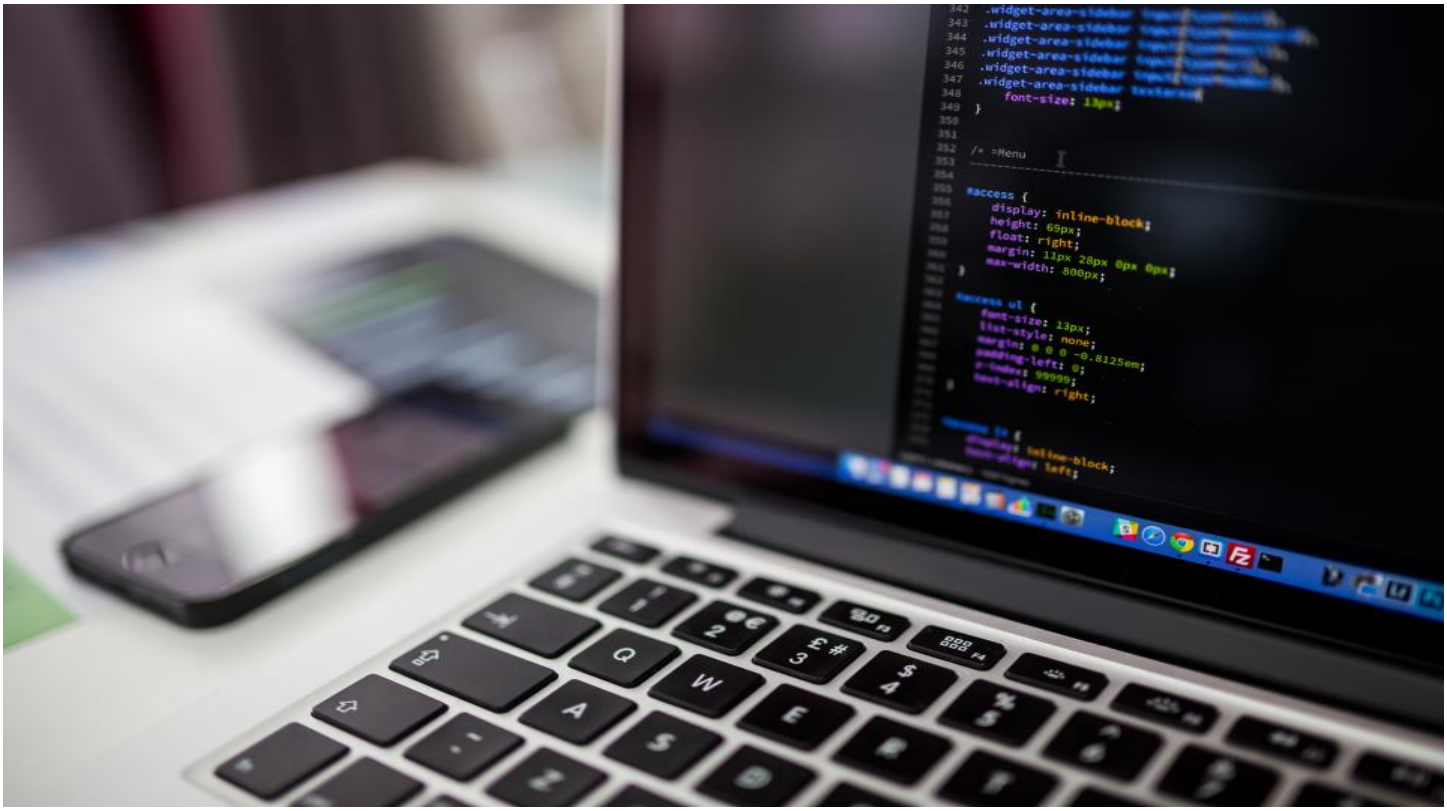

Build a Web Crawler

Let's talk about this popular system design interview question – How to build a web crawler?

Web crawlers are one of the most common used systems nowadays. The most popular example is that Google is using crawlers to collect information from all websites. Besides search engine, news websites need crawlers to aggregate data sources. It seems that whenever you want to aggregate a large amount of information, you may consider using crawlers.

There are quite a few factors when building a web crawler, especially when you want to scale the system. That's why this has become one of the most popular system design interview questions. In this post, we are going to cover topics from basic crawler to large-scale crawler and discuss various questions you may be asked in an interview.

#1 – Basic solution

How to build a rudimentary web crawler?

One simple idea we've talked about in 8 Things You Need to Know Before a System Design Interview (http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design-interviews/) is to start simple. Let's focus on building a very rudimentary web crawler that runs on a single machine with single thread. With this simple solution, we can keep optimizing later on.

To crawler a single web page, all we need is to issue a HTTP GET request (https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) to the corresponding URL and parse the response data, which is kind of the core of a crawler. With that in mind, a basic web crawler can work like this:

- Start with a URL pool that contains all the websites we want to crawl.

- For each URL, issue a HTTP GET request to fetch the web page content.
- Parse the content (usually HTML) and extract potential URLs that we want to crawl.
- Add new URLs to the pool and keep crawling.

It depends on the specific problem, sometimes we may have a separate system that generates URLs to crawl. For instance, a program can keep listening to RSS feeds and for every new article, it can add the URL into the crawling pool.

#2 – Scale issues

As is known to all, any system will face a bunch of issues after scaling. In a web crawler, there are tons of things that can make it wrong when scaling the system to multiple machines.

Before jumping to the next session, please spend a couple of minutes thinking about what can be bottlenecks of a distributed web crawler and how would you solve them. In rest of the post, we are going to talk about several major issues with solutions.

#3 – Crawling frequency

How often will you crawl a website?

This may not sound like a big deal unless the system comes to certain scales and you need very fresh content. For example, if you want to get the latest news from last hour, your crawler may need to keep crawling the news website every hour. But what's wrong with this?

For some small websites, it's very likely that their servers cannot handle such frequent request. One approach is to follow the robot.txt of each site. For people who don't know what robot.txt (https://en.wikipedia.org/wiki/Robots_exclusion_standard) is, basically it's a standard used by websites to communicate with web crawlers. It can specify things like what files should not be crawled and most web crawlers will follow the configuration. In addition, you can have different crawl frequency for different websites. Usually, there are only a few sites that need to be crawled multiple times per day.

#4 – Dedup

In a single machine, you can keep the URL pool in memory and remove duplicate entries. However, things become more complicated in a distributed system. Basically multiple crawlers may extract the same URL from different web pages and they all want to add this URL to the URL pool. Of course, it doesn't make sense to crawl the same page multiple times. So how can we dedup these URLs?

One common approach is to use Bloom Filter (https://en.wikipedia.org/wiki/Bloom_filter). In a nutshell, a bloom filter is a space-efficient system that allows you to test if an element is in a set. However, it may have false positive. In other words, if a bloom filter can tell you either a URL is definitely not in the pool or it probably in the pool.

To briefly explain how bloom filter works, an empty bloom filter is a bit array of m bits (all 0). There are also k hash functions that map each element to one of the m bits. So when we add a new element (URL) into the bloom filter, we will get k bits from the hash functions and set all of them to 1. Thus, when we check the existence of an element, we first get the k bits for it and if any of them is not 1, we know immediately that the element doesn't exist. However, if all of the k bits are 1, this can come from the combination of several other elements.

Bloom filter is a very commonly used technique and it's the perfect solution for deduping URLs in a web crawler.

#5 – Parsing

After fetching the response data from the site, the next step is to parse the data (usually HTML) to extract the information we care about. This sounds like a simple thing, however, it can be quite hard to make it robust.

The challenge is that you will always find strange markups, URLs etc. in the HTML code and it's hard to cover all corner cases. For instance, you may need to handle encode/decode issue when the HTML contains non-unicode characters. In addition, when the web page contains images, videos or even PDF, it can also cause weird behaviors.

In addition, some web pages are all rendered through Javascript like using AngularJS (https://en.wikipedia.org/wiki/AngularJS), your crawler may not be able
to get any content at all.

I would say there's no silver bullet that can make a perfect and robust crawler for all web pages. You need tons of robustness tests to make sure that it can work as expected.

Summary

There are many other interesting topics I haven't covered yet, but I'd like to mention few of them so that you can think about it. One thing is to detect loops. Many websites contain links like A→B->C->A and your crawler may end up running forever. Think about how to fix this?

Another problem is DNS lookup. When the system get scaled to certain level, DNS lookup can be a bottleneck and you may build your own DNS server.

Similar to many other systems, scaling the web crawler can be much more difficult than building a single machine version and there are lots of things that can be discussed in a system design interview. Try to start with some naive solution and keep optimizing on top of it, which can make things much easier than they seem to be.

Related Posts:

1. **Random ID Generator (http://blog.gainlo.co/index.php/2016/06/07/random-id-generator/)**
2. **Design a Key-Value Store (Part I) (http://blog.gainlo.co/index.php/2016/06/14/design-a-key-value-store-part-i/)**
3. **Create a TinyURL System (http://blog.gainlo.co/index.php/2016/03/08/system-design-interview-question-create-tinyurl-system/)**

One thought on "Build a Web Crawler"

SWATI
December 29, 2016 at 9:18 pm (http://blog.gainlo.co/index.php/2016/06/29/build-web-crawler/#comment-4401)

hello , please some body help me out. i want to make a search engine and i dont know from where to start and all what i need to start, which language can give better results etc?

Reply (http://blog.gainlo.co/index.php/2016/06/29/build-web-crawler/?replytocom=4401#respond)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

‹ Design a Key-Value Store (Part II) (http://blog.gainlo.co/index.php/2016/06/21/design-key-value-store-part-ii/)

Lowest Common Ancestor › (http://blog.gainlo.co/index.php/2016/07/06/lowest-common-ancestor/)

CATEGORIES

Coding Interview Questions (http://blog.gainlo.co/index.php/category/coding-interview-questions/)

Common Pitfalls (http://blog.gainlo.co/index.php/category/common-pitfalls/)

Common Questions (http://blog.gainlo.co/index.php/category/common-questions/)

Facebook Interview Questions (http://blog.gainlo.co/index.php/category/facebook-interview-questions/)

Mock Interview Thoughts (http://blog.gainlo.co/index.php/category/mock-interview-thoughts/)

Others (http://blog.gainlo.co/index.php/category/others/)

System Design Interview Questions (http://blog.gainlo.co/index.php/category/system-design-interview-questions/)

The Complete Guide to Google Interview Preparation (http://blog.gainlo.co/index.php/category/google-interview-preparation/)

Uber Interview Questions (http://blog.gainlo.co/index.php/category/uber-interview-questions/)