**Chapter 9**

Design a Web Crawler

Web Crawler – used by search engines to discover new or updated content on the web

* Content can be web pages, images, videos, PDF files, etc.
* Starts on a page and traverses to other pages via links
* Many purposes:
  + Search engine indexing
    - Most common use-case
    - To create local indexes for search engines
    - E.g. Googlebot is the crawler behind Google’s search engine
  + Web archiving
    - To preserve data for future uses
    - E.g. many national libraries run crawlers to archive web sites such as the US Library of Congress or the EU web archive
  + Web mining
    - To discover useful knowledge
    - E.g. top financial firms use crawlers to download shareholder meetings and annual reports to learn key company initiatives
  + Web monitoring
    - To monitor copyright and trademark infringements
    - E.g. Digimarc utilizes crawlers to discover pirated works and reports
* Complexity of development depends on the scale we intend to support

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sample Interview**   |  |  | | --- | --- | | **Questions** | **Answers** | | What is the main purpose of the crawler? | Search engine indexing | | How many web pages the web crawler collect per month? | 1 billion pages | | What content types are included? | HTML only | | Should we consider newly added or edited web pages? | Yes | | Do we need to store HTML pages crawled from the web? | Yes, up to 5 years | | How do we handle web pages with duplicate content? | Pages with duplicate content should be ignored | |

**Design Considerations**

* Scalability – web crawling should be extremely efficient using parallelization
* Robustness – must handle edge cases such as bad HTML, unresponsive servers, server crashes, or malicious links
* Politeness – should not make too many requests to a website within a short time interval
* Extensibility – should be flexible so that minimal changes are needed to support new content type

|  |
| --- |
| **Back-of-the-envelope Estimation**   * QPS   + 1,000,000,000 / 30 days / 24 hours / 3600 sec ≈ 400 pages per sec   + Peak QPS = 2 \* QPS = 800 pages per sec * Storage   + Assuming average web page size is 500k   + Monthly – 1 billion pages \* 500k = 500TB per month   + Lifetime storage – 500TB \* 12 months \* 5 years = 30PB |
|  |

**High-Level Design**

A diagram of a computer flow

Description automatically generated

* Seed URLs – starting points for crawlers
  + Seed selection can be important
    - General strategy is to divide the entire URL space into smaller ones
    - Locality-based – different countries have different popular websites
    - Topic specific – dividing the space into categories like shopping, etc.
* URL Frontier – component storing URLs to be downloaded
  + Most modern web crawlers split the crawl state to two
    - To be downloaded
    - Already downloaded
* HTML Downloader – downloads web pages from the internet.
  + URLs are provided by the URL frontier
* DNS Resolver – translates an URL to an IP address
* Content Parser – parses and validates web pages
  + Aim to ignore malformed web pages or malicious sites
  + Content parsers in a crawl server would slow down the crawling process
* Content Seen?
  + ~29% of web pages are duplicated content
  + Eliminates data redundancy and shorten processing time
  + Compares two HTML documents
    - Hashed values are much faster
* Content Storage – storage system for HTML content
  + Both disk and memory are used
  + Choice of storage system depends on data type, data size, access frequency, life span, etc.
* URL Extractor – parses and extracts links from HTML pages
* URL Filter – excludes certain content types, file extensions, error links and URLs in blacklisted sites
* URL Seen? – keeps track of URLs that are visited before or already in the frontier
  + Helps running into infinite loops
  + Bloom filter and hash table are common techniques
* URL Storage – stores already visited URLs

**Design deep dive**

*DFS vs BFS*

* The web as a directed graph where web pages are nodes and hyperlinks are edges
* Crawling can be described as traversing this directed graph
* Two common graph traversal algorithms – DFS and BFS

|  |  |
| --- | --- |
| **DFS (Depth first search)** | **BFS (Breadth first search)** |
| * Generally not a good choice * Depth of DPS can be very deep | * Commonly used by web crawlers * Implemented by a first-in-first-out (FIFO) queue * Vulnerabilities   + Impolite – most links in a web page are linked back to the same host. A crawler trying to download web pages in parallel will flood the servers with requests   + Standard BFS does not consider priority of URLs (some URLs should have higher level of importance) |

*URL Frontier*

* Solves politeness and URL prioritization problems
* Politeness
  + Too many requests can be treated as a DOS attack
  + Map different website hostnames to download to downloader threads (worker)
    - Each downloader thread has a separate FIFO queue and only downloads URLs obtained from that queue
    - Enforces downloading to occur one page at a time from the same host
    - A delay can be added between two download tasks
    - Queue router – ensures each queue only contains URLs from the same host
    - Mapping table – maps each host to a queue
* Priority
  + Prioritizer – prioritizes URLs based on PageRank, website traffic, and update frequency
    - Takes URLs as inputs and computes the priorities
  + Queues – each has an assigned priority
  + Queue selector – Random chooses queue. High priority queues have a higher chance to selected
* Freshness – a crawler must periodically recrawl downloaded pages to keep our data up to date
  + Recrawling is time consuming and resource intensive
  + Strategies
    - Recrawl based on web pages’ update history
    - Prioritize URLs and recrawl important pages first and more frequently
* Storage
  + Putting everything in memory is neither durable nor scalable
  + Keeping everything on disk is undesirable as disk is slow and can become bottleneck for the crawl
  + Hybrid approach
    - Majority of URLs are stored on disk
    - Maintain buffers in memory for enqueue/dequeue operations
    - Data in the buffer is periodically written to the disk

*HTML Downloader*

* Downloads web pages using the HTTP protocol
* Robots Exclusion Protocol – standard used by websites to indicate to visiting web crawlers and other robots which portions of the website they are allowed to visit
  + Before crawling, a crawler should check its robots.txt to follow its rules
  + Contents of robots.txt tends to be cache to prevent repeated downloading

*Optimizing Performance*

* Distributed crawl
  + Distribute crawlers into multiple servers with each server running multiple threads, allowing the URL space to be partitioned into smaller pieces and improving performance
* Cache DNS Resolver
  + DNS Resolver is a bottleneck because DNS requests might take time due to the synchronous nature of many DNS interfaces
  + DNS response time ranges from 10-200ms
  + Once a request to DNS is carried out by a crawler thread, other threads are blocked until the first request is completed
  + Maintaining our DNS cache to avoid calling DNS frequently is key for speed optimization
  + Our DNS cache keeps the domain name to IP address mapping and is updated periodically by cron jobs
* Locality
  + Distribute crawl servers geographically
  + Servers closer to the host experiences faster crawling and download speed
  + Design locality applies to most of the system components – crawler servers, cache, queue, storage, etc.
* Short timeout
  + Some web servers respond slowly or may not response at all
  + To avoid long wait time, a maximum wait time is specified
  + If a host does not respond within a predefined time, the crawler will stop the job and crawl some other pages

*Improving Extensibility*

* The crawler can be extended by plugging in new modules

A diagram of a computer process

Description automatically generated

* PNG Downloader – downloads PNG files
* Web Monitor – monitors the web and prevent copyright and trademark infringement

*Detect and avoid problematic content*

* Redundant content
  + Nearly 30% of the web pages are duplicates.
  + Hashes or checksums help to detect duplication
* Spider traps – a web page that causes crawlers in an infinite loop
  + Can be avoided by setting a maximum length of URLs
  + No one-size-fits-all solution exists to detect spider traps
* Data noise
  + Some of the contents have little or no value, such as advertisements, code snippets, spam URLs

*Other Considerations*

* Server-side rendering
  + Numerous websites use scripts JavaScript to generate links on the fly
  + To solve this, we perform server-side rendering first before parsing a page
* Filter out unwanted pages
  + An anti-spam component to filter out low quality and spam pages
* Database replication and sharing
  + Improve the data layer availability, scalability, and reliability
* Horizontal scaling
  + Hundreds or even thousands of servers may be needed
  + Keep servers stateless