Web Crawler

Starts by collecting few web page then follow links to this page to collect new content.

1. Content can be a web page, an image, a video, a PDF file
2. Uses
    1. Search engine indexing - create local index for search engine.
    2. Web archiving - collect information from web to preserve data for future use.
    3. Web mining - discover useful knowledge from internet
    4. Web monitoring - monitor copyright and trademark infringement over internet.

Understand problem and establish design scope

1 given set of URL - download all web page address by URL
2. Extract url from these web page
3. Add new URL to list of URL to be downloaded. Repeat these 3 steps

questions

1. Main purpose of web crawler
Ans: search engine indexing

2. How many web pages does crawler collect per month
Ans: 1 bn

3. Content types include
Ans: html only

4. Consider new added and edited web page
Ans: yes

5. How long to store html page collected?
Ans: upto 5 years

6. handle duplicated web page
page with duplicate content should be ignored

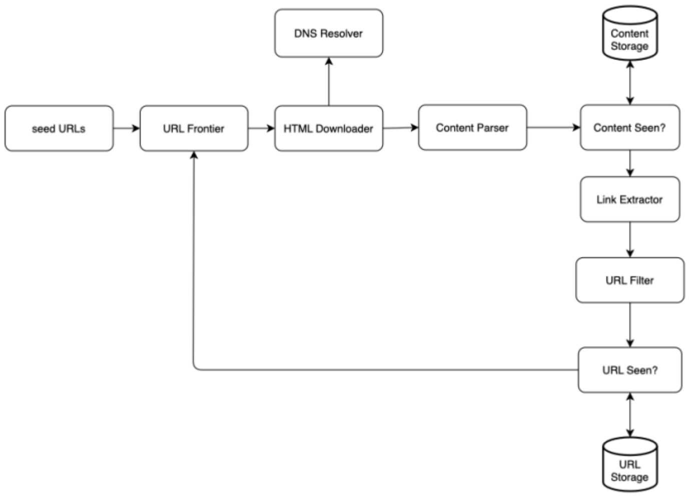
non function requirements

1. Scalability - extremely efficient using parallelization
2. Robustness - traps: based on html, unresponsive server, crash, malicious link etc. - handle those edge case.
3. Politeness - not take too many request to a website within short time
4. Extensibility - flexible with minimal change to support new content types

Back of envelope estimation

1 bn web page
- QPS : 1,000,000,000 / 30 days / 24 hr / 3600 sec = ~400 page per second
- peak ups = 800
- avg page size = 500k
- total storage = 1 bn * 500k = 500 TB
- storage for 5 years = 5000 tb *12 * 5 = 30 PB

High level design



- Seed url - start with selected seed url which will cover as many links as possible - divide based on locality / country / or topics
- URL frontier - split crawl state into : 1. Downloaded 2. To be download. URL frontier store to be downloaded refer this as FIFO queue.
- html downloader - down web page from internet provided by url frontier
- DNS resolver - to download web page, url must translate to dns resolver HTML downloader call DNS resolver to get corresponding IP address for URL.
- content parser - once downloaded, content must parse and validated to avoid malformed page - this might slow crawl process, thus is a separate component
- content seen - 29% of web page are duplicate content - this will remove redundancy and shorten processing time
- content storage - most content store in disk, popular being stored in memory
- url extractor - parse and extract link from html page - relative path converted to absolute path by adding prefix
- url filter - exclude certain content type, file extension, error link and url In blacklist
- url seen - avoid adding same URL multiple times to reduce server load and cause infinite loop. Bloom filter is common topic
- url storage.- store already visited url

Step 3: Design Deep Dive

1. DFS vs BFS
2. URL frontier
3. HTML downloader
4. Robustness
5. Extensibility
6. Detect and avoid problematic content

1. **BFS vs DFS**
    1. Crawling = visiting directed graph with nodes (web page) and edges (url).
    2. DFS not good choice - since can be very deep
    3. BFS implemented by FIFO queue, URL dequeued in orde of enqueue
        1. Problem - most link are linked back to same host
        2. - also not take priority or URL into consideration
2. **URL frontier**
    1. Store to be downloaded URL
    2. Ensure politeness, prioritization and freshness
        1. Politeness - not send too many req to same host in short time period - DDoS attack. Add delay to make 1 download at a time. Maintain a map to download worker (thread), each thread has separate FIFO queue
            1. Queue router - ensure each queue only contain URL from same host.
            2. Mapping table - map host to queue
            3. Queue selector - each worker thread mapped to FIFO queue, only download url from that queue
            4. Worker thread - download web page one by one from the same host

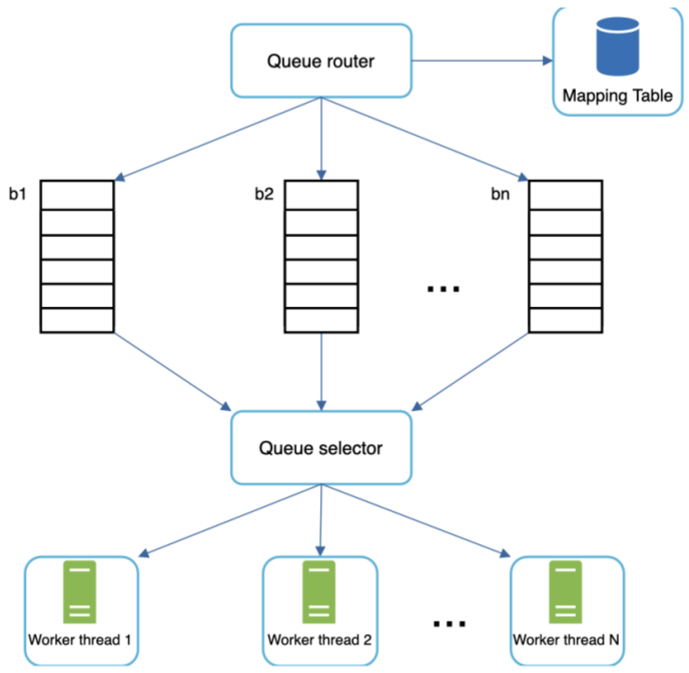
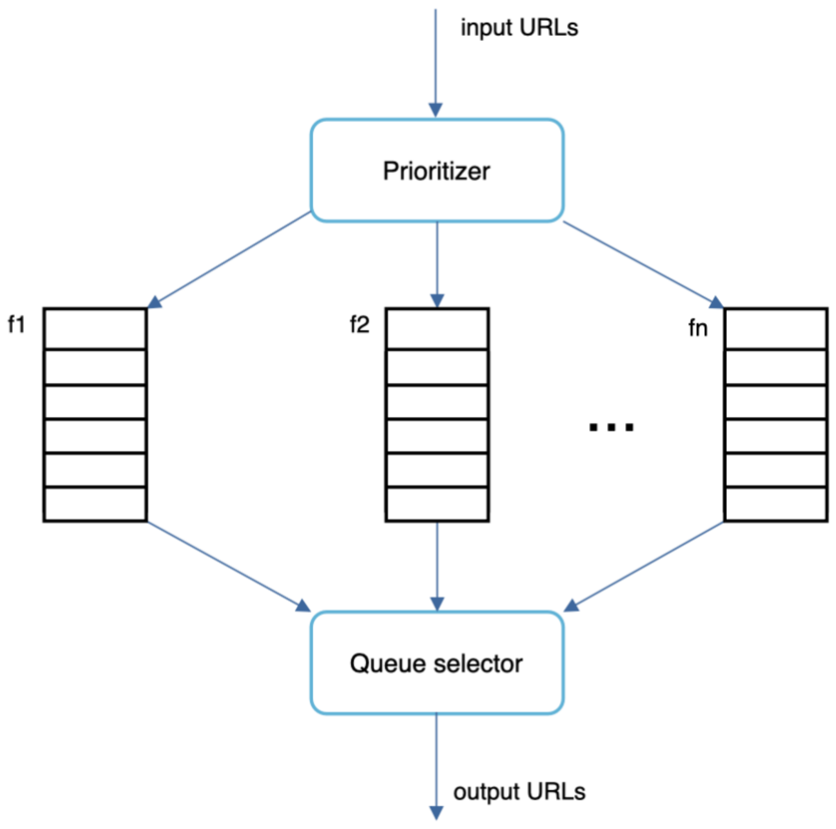


Figure 6

-

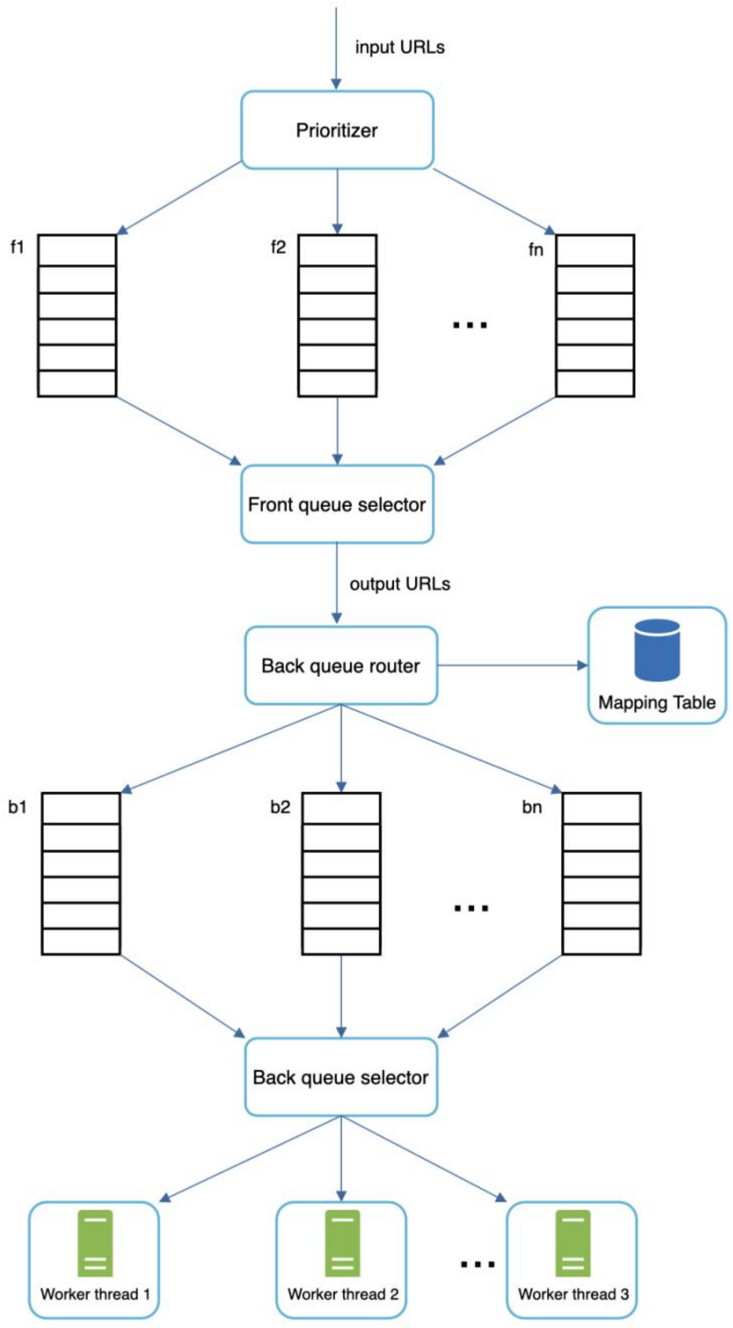
2. Priority
- measure based on page rank, website traffic, update frequency



- each queue has assigned priority. Queue with high priority are selected with high probability.
- queue selector randomly choose queue with a bias towards queue with higher priority.

3. Freshness - redrawn based on web page update history, prioritize URL and redrawn important page first and more frequently.

4. **URL frontier storage** - to reduce cost - maintain buffer in memory for enqueue/dequeue operation, so that buffer is periodically written



HTML downloader

download web page using HTTP protocol.

- cache result of robots.txt file (robot exclusion protocol), which specify which page to download

- performance optimization

1. Distributed crawl

crawl job distributed into multiple server, each run on multiple threads

url space partitioned into smaller piece, so each downloaded responsible for subset of URL

2. Cache DNS resolver -

DNS response time range = 10 ms to 200 ms

once crawler call DNS - other thread are blocked until first request complete.

Maintain DNS cache to avoid calling DNS frequently

3. Locality

server close to website host - fast download time

4. Robustness

1. Consistent hashing to distribute load among downloader
2. Save crawl state and data to storage system
3. Exception handling gracefully without crash system
4. Data validation to prevent error
5. Extensibility

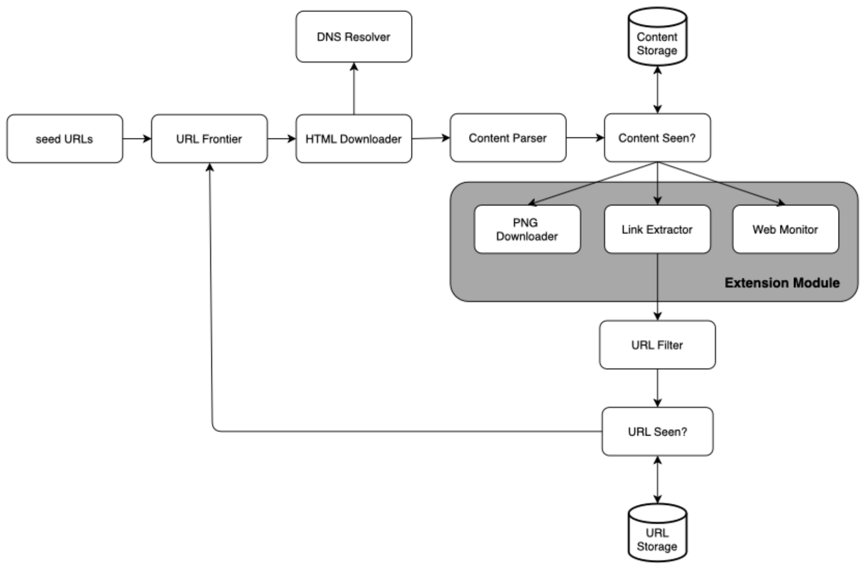


Figure 10

Detect and avoid problematic content

1. Redundant content - hashes and checksum to solve
2. Spider traps - user manual verify and exclude from list
3. Data noise - remove adv, code snippet, spam URL

Wrap up

1. Server side rendering (dynamic rendering)
2. Filter out unwanted page - anti spam component
3. Db replication and sharing
4. Horizontal scaling
5. Availability, consistency and reliability
6. Analytics - collect and analyze data

Reference
http://bytebytego.com