

CSE 7/5337: Information Retrieval and Web Search

Web crawling and indexes (IIR 20)

Michael Hahsler

Southern Methodist University

These slides are largely based on the slides by Hinrich Schütze
Institute for Natural Language Processing, University of Stuttgart
<http://informationretrieval.org>

Spring 2012

Outline

1 A simple crawler

2 A real crawler

How hard can crawling be?

- Web search engines must crawl their documents.
- Getting the content of the documents is easier for many other IR systems.
 - ▶ E.g., indexing all files on your hard disk: just do a recursive descent on your file system
- Ok: for web IR, getting the content of the documents takes longer . . .
- . . . because of latency.
- But is that really a design/systems challenge?

Basic crawler operation

- Initialize queue with URLs of known seed pages
- Repeat
 - ▶ Take URL from queue
 - ▶ Fetch and parse page
 - ▶ Extract URLs from page
 - ▶ Add URLs to queue
- Fundamental assumption: The web is well linked.

Exercise: What's wrong with this crawler?

```
urlqueue := (some carefully selected set of seed urls)
while urlqueue is not empty:
    myurl := urlqueue.getlastanddelete()
    mypage := myurl.fetch()
    fetchedurls.add(myurl)
    newurls := mypage.extracturls()
    for myurl in newurls:
        if myurl not in fetchedurls and not in urlqueue:
            urlqueue.add(myurl)
    addtoinvertedindex(mypage)
```

What's wrong with the simple crawler

- Scale: we need to **distribute**.
- We can't index everything: we need to **subselect**. How?
- Duplicates: need to integrate **duplicate detection**
- Spam and spider traps: need to integrate **spam detection**
- **Politeness**: we need to be “nice” and space out all requests for a site over a longer period (hours, days)
- **Freshness**: we need to recrawl periodically.
 - ▶ Because of the size of the web, we can do frequent recrawls only for a small subset.
 - ▶ Again, subselection problem or **prioritization**

Magnitude of the crawling problem

- To fetch 20,000,000,000 pages in one month . . .
- . . . we need to fetch almost 8000 pages per second!
- Actually: many more since many of the pages we attempt to crawl will be duplicates, unfetchable, spam etc.

What a crawler must do

Be polite

- Don't hit a site too often
- Only crawl pages you are allowed to crawl: robots.txt

Be robust

- Be immune to spider traps, duplicates, very large pages, very large websites, dynamic pages etc

- Protocol for giving crawlers (“robots”) limited access to a website, originally from 1994
- Examples:
 - ▶ User-agent: *
Disallow: /yoursite/temp/
 - ▶ User-agent: searchengine
Disallow: /
- Important: cache the robots.txt file of each site we are crawling

Example of a robots.txt (nih.gov)

```
User-agent: PicoSearch/1.0
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/

User-agent: *
Disallow: /news/information/knight/
Disallow: /nidcd/
...
Disallow: /news/research_matters/secure/
Disallow: /od/ocpl/wag/
Disallow: /ddir/
Disallow: /sdminutes/
```

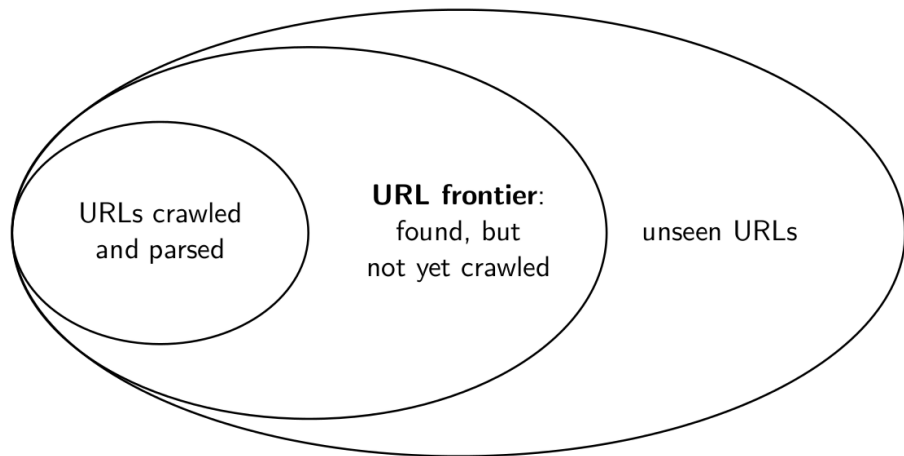
What any crawler should do

- Be capable of **distributed** operation
- Be scalable: need to be able to increase crawl rate by adding more machines
- Fetch pages of higher quality first
- Continuous operation: get fresh version of already crawled pages

Outline

- 1 A simple crawler
- 2 A real crawler

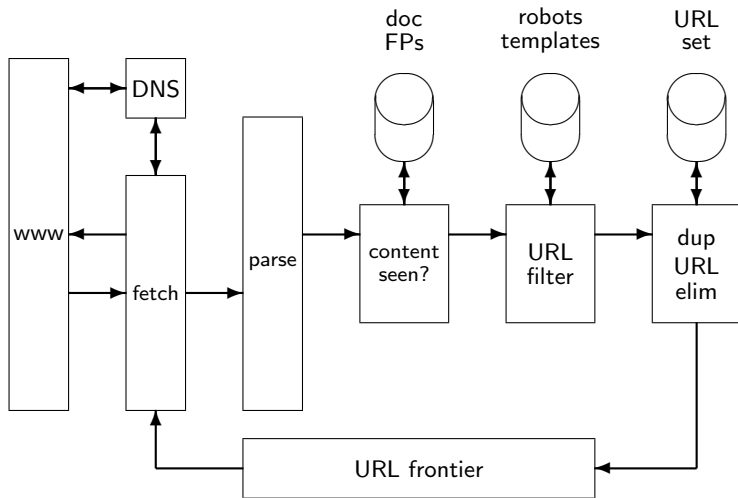
URL frontier



URL frontier

- The URL frontier is the data structure that holds and manages URLs we've seen, but that have not been crawled yet.
- Can include multiple pages from the same host
- Must avoid trying to fetch them all at the same time
- Must keep all crawling threads busy

Basic crawl architecture



URL normalization

- Some URLs extracted from a document are **relative** URLs.
- E.g., at `http://mit.edu`, we may have `aboutsitem.html`
 - ▶ This is the same as: `http://mit.edu/aboutsite.html`
- During parsing, we must normalize (expand) all relative URLs.

Content seen

- For each page fetched: check if the content is already in the index
- Check this using document fingerprints or [shingles](#)
- Skip documents whose content has already been indexed

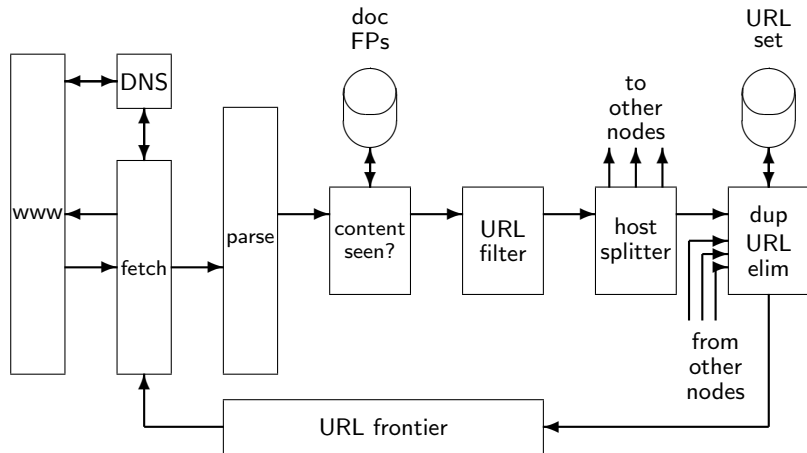
Distributing the crawler

- Run multiple crawl threads, potentially at different nodes
 - ▶ Usually geographically distributed nodes
- Partition hosts being crawled into nodes

Google data centers



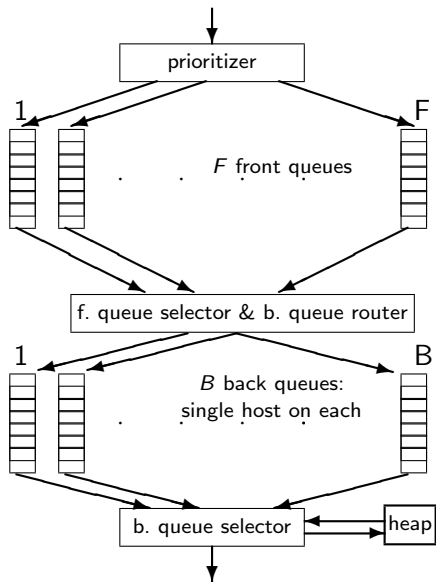
Distributed crawler



URL frontier: Two main considerations

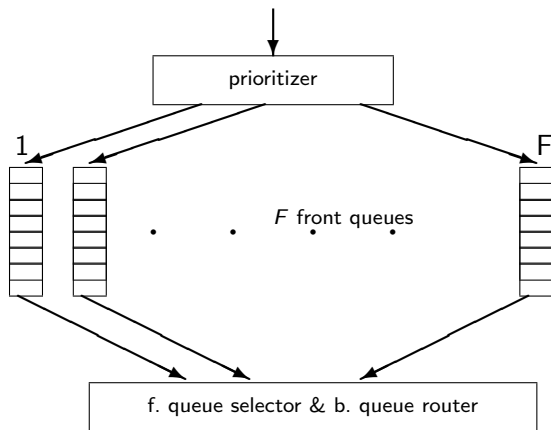
- Politeness: Don't hit a web server too frequently
 - ▶ E.g., insert a time gap between successive requests to the same server
- Freshness: Crawl some pages (e.g., news sites) more often than others
- Not an easy problem: simple priority queue fails.

Mercator URL frontier



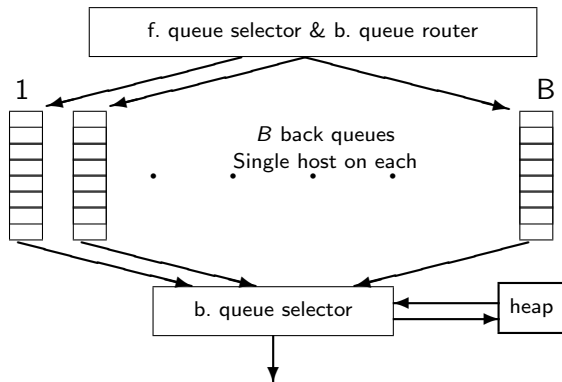
- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.
- Each queue is FIFO.

Mercator URL frontier: Front queues



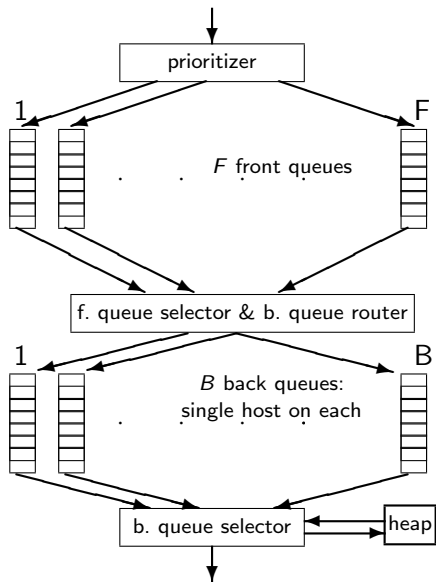
- Prioritizer assigns to URL an integer priority between 1 and F .
- Then appends URL to corresponding queue
- Heuristics for assigning priority: refresh rate, PageRank etc
- Selection from front queues is initiated by back queues

Mercator URL frontier: Back queues



- **Invariant 1.** Each back queue is kept non-empty while the crawl is in progress.
- **Invariant 2.** Each back queue only contains URLs from a single host.
- Maintain a table from hosts to back queues.
- In the heap:
- One entry for each back queue
- The entry is the earliest time t_e at

Mercator URL frontier



- URLs flow in from the top into the frontier.
- Front queues manage prioritization.
- Back queues enforce politeness.
- Each queue is FIFO.

Spider trap

- Malicious server that generates an infinite sequence of linked pages
- Sophisticated spider traps generate pages that are not easily identified as dynamic.

- Chapter 20 of IIR
- Resources at <http://ifnlp.org/ir>
 - ▶ Paper on Mercator by Heydon et al.
 - ▶ Robot exclusion standard