Information Retrieval

Weike Pan

The slides are **adapted from those provided by Prof. Hinrich Schütze** at University of Munich (http://www.cis.lmu.de/~hs/teach/14s/ir/).

Chapter 20 Web crawling and indexes

- 20.1 Overview
- 20.2 Crawling
- 20.3 Distributing indexes
- 20.4 Connectivity servers
- 20.5 References and further reading

Chapter 20

- 20.1 Overview
- 20.2 Crawling
- 20.3 Distributing indexes
- 20.4 Connectivity servers
- 20.5 References and further reading

20.2 Crawling

- 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 (递归下降) on your file system
- For web IR, getting the content of the documents takes longer because of latency.

20.2 Crawling

- Basic operations of a simple crawler
  - 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**

20.2 Crawling

- What's wrong with the simple crawler (1/4)
  - Scale: we need to distribute.
    - need to be able to increase crawl rate by adding more machines
  - We can't index everything: we need to sub-select … How?
  - Duplicates: need to integrate duplicate detection
  - Spam and spider traps: need to integrate spam detection

20.2 Crawling

- What's wrong with the simple crawler (2/4)
  - Be polite: we need to be "nice" and space out (隔开) all requests for a site over a longer period (hours, days)
    - Don't hit a site too often
    - Only crawl pages you are allowed to crawl: robots.txt

```
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/
```

Example of a robots.txt (nih.gov)

Protocol for giving crawlers ("robots") limited access to a website, originally from 1994

Important: cache the robots.txt file of each site we are crawling

20.2 Crawling

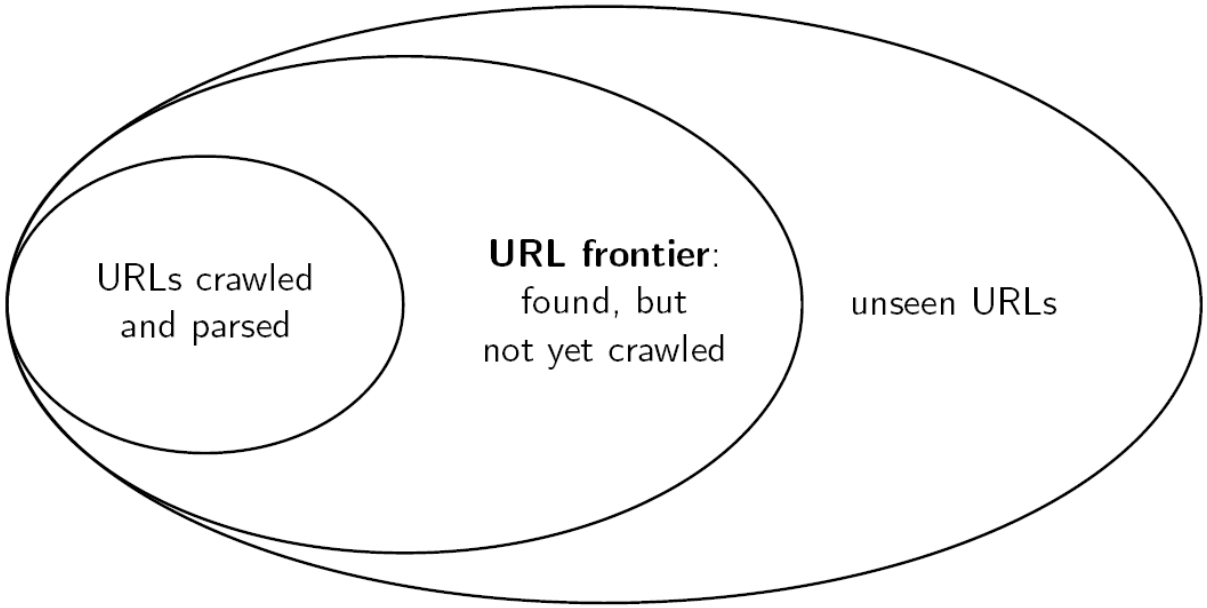
- What's wrong with the simple crawler (3/4)
  - Freshness: we need to re-crawl periodically
    - Because of the size of the web, we can do frequent re-crawls only for a small subset
    - Again, sub-selection (选择子集) or prioritization (优先次序)

20.2 Crawling

- What's wrong with the simple crawler (4/4)
  - Be robust
    - Be immune to spider traps, duplicates, very large pages, very large websites, dynamic pages, etc

20.2 Crawling

- A real crawler

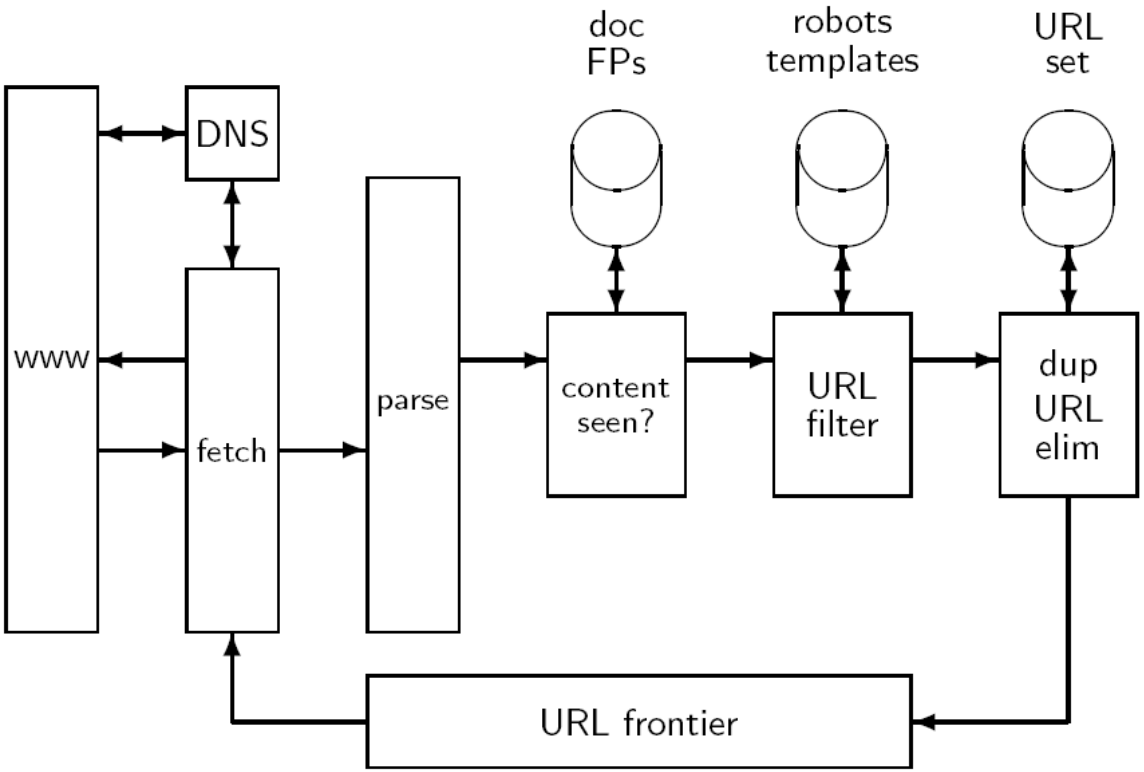


20.2 Crawling

- URL frontier
  - The URL frontier is the data structure that holds and manages URLs we have 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

20.2 Crawling

- Basic crawl architecture



20.2 Crawling

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

20.2 Crawling

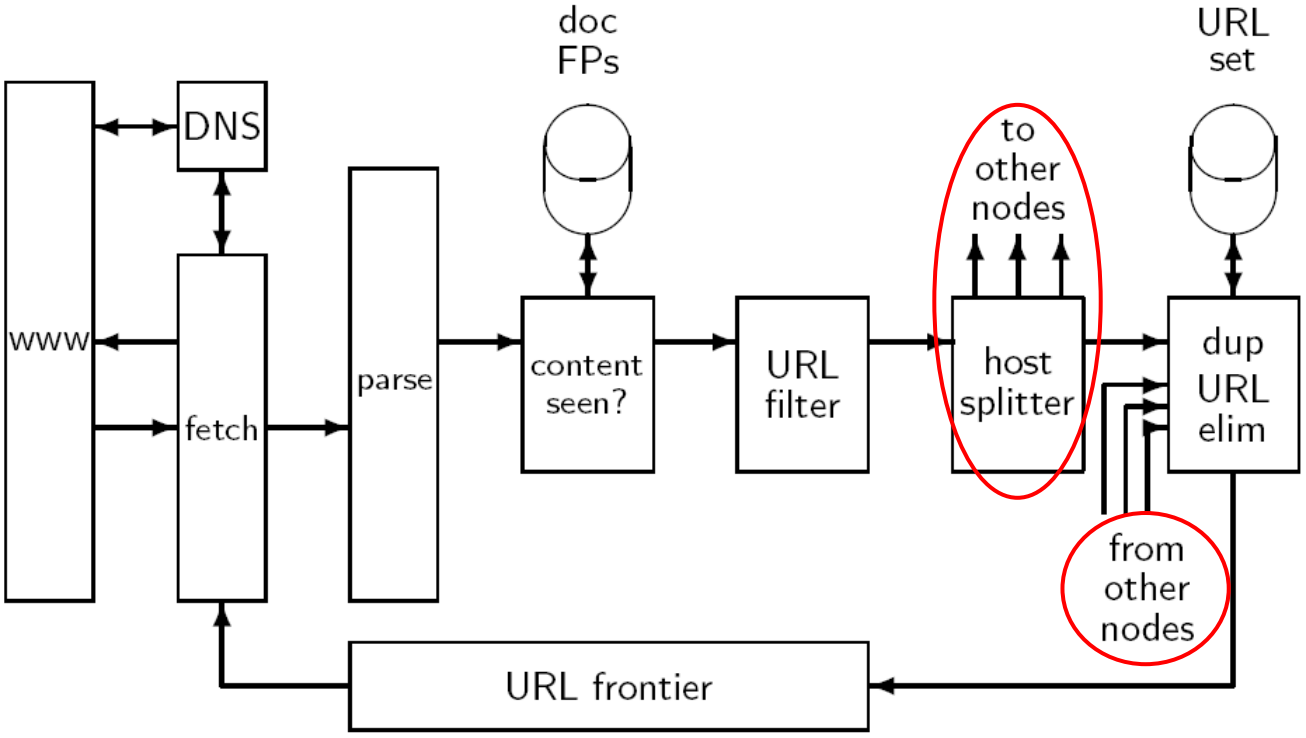
- Content seen
  - For each page fetched: check if the content is already in the index
  - Check this using document fingerprints or shingles (Chapter 19)
  - Skip documents whose content has already been indexed

20.2 Crawling

- Distributing the crawler
  - Run multiple crawl threads, potentially at different nodes
    - Usually geographically distributed nodes
  - Partition hosts being crawled into different nodes (将要爬取的站点分给不同的节点)

20.2 Crawling

- Distributed crawler

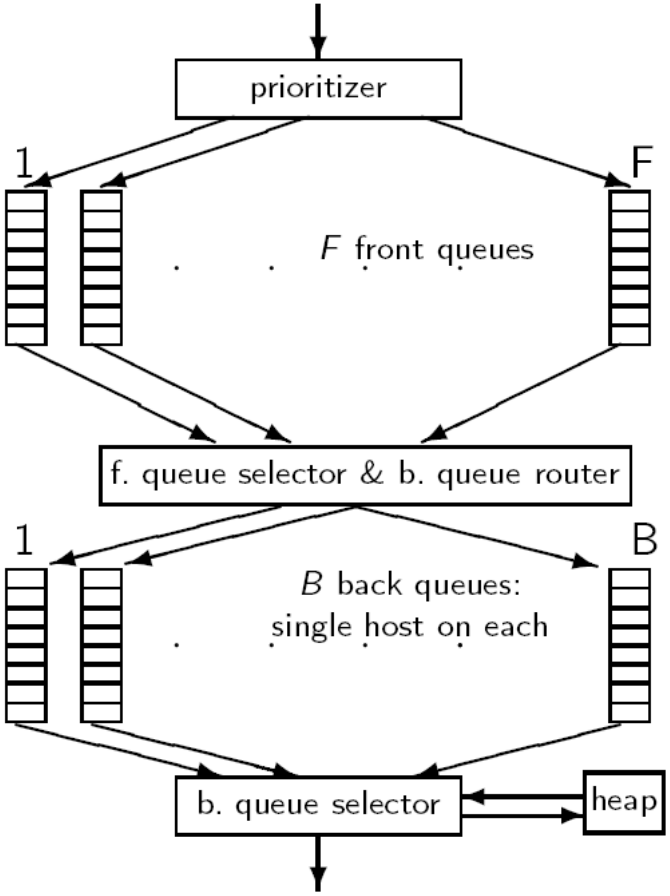


20.2 Crawling

- 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

20.2 Crawling

- The URL frontier



URLs flow in from the top into the frontier.

Front queues manage prioritization.

Back queues enforce politeness.

Each queue is FIFO.

20.2 Crawling

- Front queues
  - Prioritizer assigns to URL an integer priority between 1 and F
  - Then appends URL to the corresponding queue
  - Heuristics for assigning priority: refresh rate, PageRank, etc

  - Selection from front queues is initiated by back queues
  - Pick a front queue from which to select next URL: round robin (轮询), randomly, or a more sophisticated variant
  - But with a bias in favor of high-priority front queues

20.2 Crawling

- Back queues (1/4)
  - 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

20.2 Crawling

- Back queues (2/4)
  - In the heap: One entry for each back queue
  - The entry is the earliest time (used for politeness) t_e at which the host corresponding to the back queue can be hit again
    - The earliest time t_e is determined by (i) last access to that host, and (ii) time gap heuristic

20.2 Crawling

- Back queues (3/4)
  - How **fetcher** interacts with back queue
    - Repeat: (i) extract current root queue *q* of the heap (*q* is a back queue), and (ii) fetch URL *u* at the head of *q*
    - …
    - … until we empty the queue *q* we get (i.e., *u* was the last URL in *q*)

20.2 Crawling

- Back queues (4/4)
  - When we have **emptied** a back queue *q*
    - Repeat: (i) pull URLs *u* from front queues, and (ii) add *u* to its corresponding back queue … until we get a *u* whose host does not have a back queue. Then put *u* in *q* and create a heap entry for it.

20.2 Crawling

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

Summary

- 20.1 Overview
- 20.2 Crawling
- 20.3 Distributing indexes
- 20.4 Connectivity servers
- 20.5 References and further reading