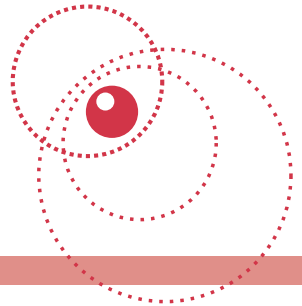


Web Mining

Web search: Web Crawling

5 October 2010

Pierre Senellart

TELECOM ParisTech

Discovering new URLs

Identifying duplicates

Crawling architecture

Recrawling URLs?

Conclusion

- crawlers, (Web) spiders, (Web) robots: autonomous user agents that retrieve pages from the Web
- Basics of crawling:
  1. Start from a given URL or set of URLs
  2. Retrieve and process the corresponding page
  3. Discover new URLs (cf. next slide)
  4. Repeat on each found URL
- No real termination condition (virtual unlimited number of Web pages!)
- Graph-browsing problem

  deep-first: not very adapted, possibility of being lost in robot traps

  breadth-first

  combination of both: breadth-first with limited-depth deep-first on each discovered website

- From HTML pages:
  - hyperlinks `<a href="...">...</a>`
  - media `<img src="...">` `<embed src="...">` `<object data="...">`
  - frames `<frame src="...">` `<iframe src="...">`
  - JavaScript links `window.open("...")`
  - etc.
- Other hyperlinked content (e.g., PDF files)
- Non-hyperlinked URLs that appear anywhere on the Web (in HTML text, text files, etc.): use regular expressions to extract them
- Referrer URLs
- Sitemaps [sitemaps.org, 2008]

Web-scale

- The Web is infinite! Avoid robot traps by putting depth or page number limits on each Web server
- Focus on important pages [Abiteboul et al., 2003] (cf. lecture on the Web graph)

- Web servers under a list of DNS domains: easy filtering of URLs
- A given topic: focused crawling techniques [Chakrabarti et al., 1999, Diligenti et al., 2000] based on classifiers of Web page content and predictors of the interest of a link.
- The national Web (cf. public deposit, national libraries): what is this? [Abiteboul et al., 2002]
- A given Web site: what is a Web site? [Senellart, 2005]

A hash function is a deterministic mathematical function transforming objects (numbers, character strings, binary...) into fixed-size, seemingly random, numbers. The more random the transformation is, the better.

Example

Java hash function for the `String` class:

$$\sum_{i=0}^{n-1} s_i \times 31^{n-i-1} \bmod 2^{32}$$

where s_i is the (Unicode) code of character i of a string s.

Problem

Identifying duplicates or near-duplicates on the Web to prevent multiple indexing

trivial duplicates: same resource at the same canonized URL:

```
http://example.com:80/toto
http://example.com/titi/../toto
```

exact duplicates: identification by hashing

near-duplicates: (timestamps, tip of the day, etc.) more complex!

Edit distance. Count the minimum number of basic modifications (additions or deletions of characters or words, etc.) to obtain a document from another one. Good measure of similarity, and can be computed in $O(mn)$ where m and n are the size of the documents. But: does not scale to a large collection of documents (unreasonable to compute the edit distance for every pair!).

Shingles. Idea: two documents similar if they mostly share the same succession of k-grams (succession of tokens of length k).

Example

I like to watch the sun set with my friend.
My friend and I like to watch the sun set.

S ={i like, like to, my friend, set with, sun set, the sun, to watch, watch the, with my}
T ={and i, friend and, i like, like to, my friend, sun set, the sun, to watch, watch the}

- Similarity: Jaccard coefficient on the set of shingles:

$$J(S,T) = \frac{|S \cap T|}{|S \cup T|}$$

- Still costly to compute! But can be approximated as follows:
  1. Choose N different hash functions
  2. For each hash function h_i and each set of shingles $S_k = \{s_{k1} \ldots s_{kn}\}$, store $\phi_{ik} = \min_j h_i(s_{kj})$
  3. Approximate $J(S_k, S_l)$ as the proportion of ϕ_{ik} and ϕ_{il} that are equal
- Possibly to repeat in a hierarchical way with super-shingles (we are only interested in very similar documents)

- Standard for robot exclusion: robots.txt at the root of a Web server [Koster, 1994].

```
User-agent: *
Allow: /searchhistory/
Disallow: /search
```

- Per-page exclusion (*de facto* standard).

```
<meta name="ROBOTS" content="NOINDEX,NOFOLLOW">
```

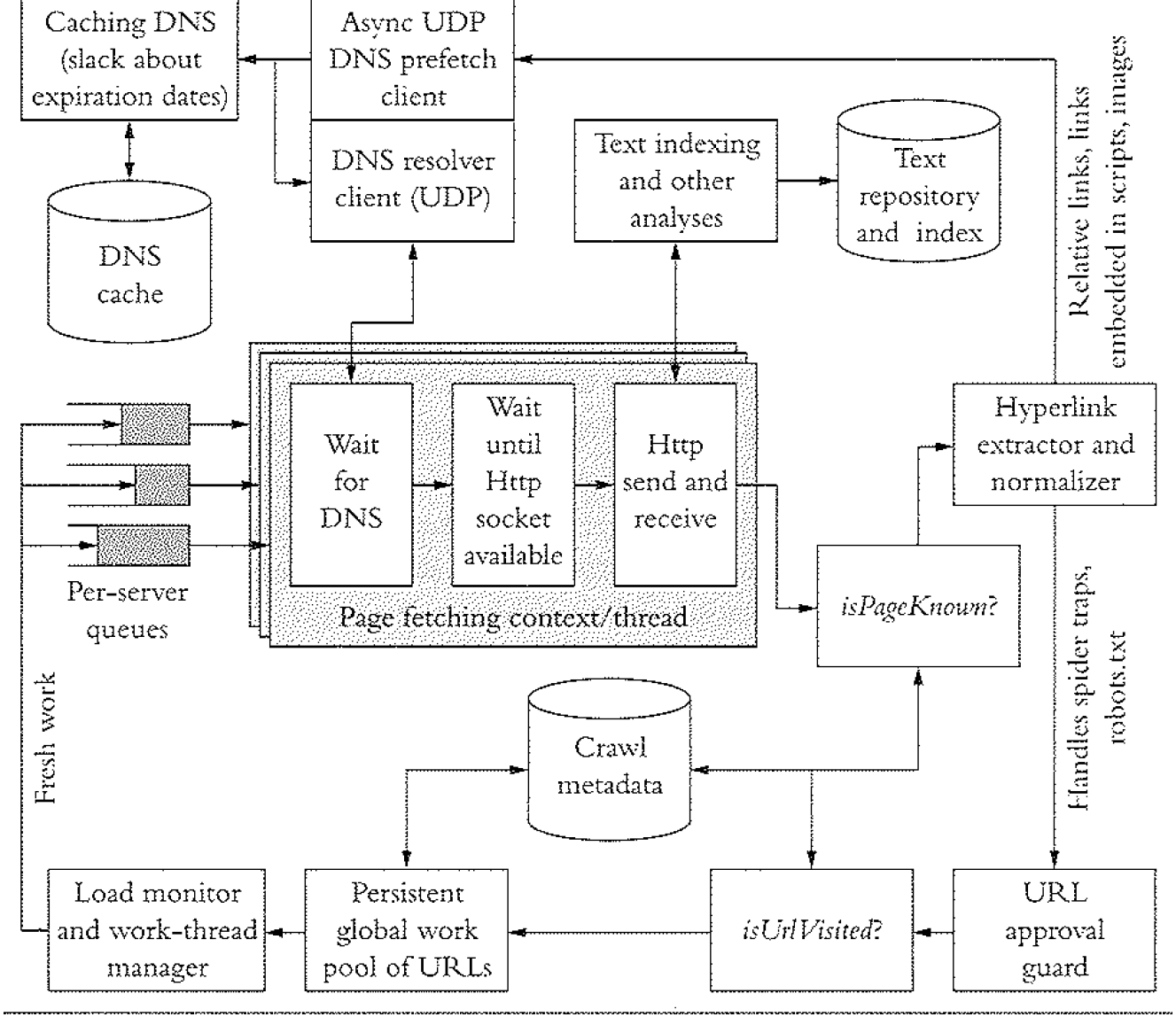
- Per-link exclusion (*de facto* standard).

```
<a href="toto.html" rel="nofollow">Toto</a>
```

- Avoid Denial Of Service (DOS), wait 100ms/1s between two repeated requests to the same Web server

Network delays, waits between requests:

- Per-server queue of URLs
- Parallel processing of requests to different hosts:
  - multi-threaded programming
  - asynchronous inputs and outputs (`select`, classes from `java.util.concurrent`): less overhead
- Use of keep-alive to reduce connexion overheads
- Local DNS cache to save on DNS lookup costs


Caching DNS (slack about expiration dates)
Async UDP DNS prefetch client
DNS resolver client (UDP)
DNS cache
Text indexing and other analyses
Text repository and index
Relative links, links embedded in scripts, images
Wait for DNS
Wait until Http socket available
Http send and receive
Page fetching context/thread
Per-server queues
Hyperlink extractor and normalizer
isPageKnown?
Handles spider traps, robots.txt
Fresh work
Crawl metadata
Load monitor and work-thread manager
Persistent global work pool of URLs
isUrlVisited?
URL approval guard

- Content on the Web changes
- Different change rates:

  online newspaper main page: every hour or so
  published article: virtually no change
- Continuous crawling, and identification of change rates for adaptive crawling: how to know the time of last modification of a Web page?

Two timestamping mechanism in HTTP: entity tags and modification dates. Potentially provided with all requests:

```
ETag: "497bef-1fcb-47f20645"
Last-Modified: Tue, 01 Apr 2008 09:54:13 GMT
```

Etag: unique identifier for the provided document, should change if the document changes; can be used in requests with `If-Match` and `If-None-Match`.

Lat-Modified: last modification time; can be used in requests with `If-Modified-Since` and `If-Unmodified-Since`.

- Information generally provided and very reliable for static content (often includes media files in CMS).
- Information hardly ever provided (or with dummy *now* date) for dynamic content, CMS, etc.

Two additional (redundant) items of freshness information, for caches and proxies:

```
Cache-Control: max-age=60, private
Expires: Tue, 01 Apr 2008 13:25:55 GMT
```

max-age: maximum time in second a document remains fresh.

Expires: time when a document stops being fresh.

- Often provided...
- ...but with 0 or very low expiration delay.
- $\Rightarrow$ does not give any interesting information.

Very frequent in CMS:

- either as a global timestamps (*Last modified:*);
- or on individual items: news stories, blogs, etc. (is the global timestamp the max of the individual ones?);
- possibly also in meta-data on the Web page: comments, Dublin Core `<meta>` tags.

- Quite easy to identify and extract from the Web page (keywords, date recognition).
- Informal: sometimes partial (no time indication), often without timezone.
- Might not always be trustworthy.

Files of other types than HTML may have semantic timestamping mechanism:

PDF, Office suite documents, etc.: both creation and modification date available in metadata. Quite reliable.

RSS feeds: reliable semantic timestamps.

Images, Sounds: EXIF (or similar) metadata. Not always reliable, and the capture date of a picture may have nothing in common with its publication date.

Semantic external content used for dating a HTML Web page:

- Possibility of mapping a RSS feed to Web page content
- Sitemaps provided by the Web site owner. Allows for providing both timestamps and change rate indications (*hourly*, *monthly*...), but these functionalities are not often used. A few CMS produce all of this: ideal case!

1. Check HTTP timestamp.
2. Check content timestamp.
3. Compare a hash of the page with a stored hash.
4. Non-significant differences (ads, fortunes, request timestamp):
   - only hash text content, or “useful” text content;
   - compare distribution of *n*-grams (shingling);
   - or even compute edit distance with previous version.

Adapting strategy to each different archived website?

Discovering new URLs

Identifying duplicates

Crawling architecture

Recrawling URLs?

Conclusion

What you should remember

- Crawling as a graph-browsing problem.
- Shingling for identifying duplicates.
- Numerous engineering issues in building a Web-scale crawler.

Software

- Wget, a simple yet effective Web spider (free software)
- Heritrix, a Web-scale highly configurable Web crawler, used by the Internet Archive (free software)
- HTML Parser, TagSoup: Java libraries for parsing real-world Web pages

A good textbook [Chakrabarti, 2003]

Serge Abiteboul, Grégory Cobena, Julien Masanès, and Gerald Sédrati. A first experience in archiving the French Web. In *Proc. ECDL*, Roma, Italie, September 2002.

Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive on-line page importance computation. In *Proc. WWW*, May 2003.

Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Fransisco, USA, 2003.

Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *Proc. VLDB*, Cairo, Egypt, September 2000.

Martijn Koster. A standard for robot exclusion. `http://www.robotstxt.org/orig.html`, June 1994.

Pierre Senellart. Identifying Websites with flow simulation. In *Proc. ICWE*, pages 124–129, Sydney, Australia, July 2005.

sitemaps.org. Sitemaps XML format. `http://www.sitemaps.org/protocol.php`, February 2008.