Introduction

- Crawling: gathering pages from the internet, in order to index them
- 2 main objectives:
  - fast gathering
  - efficient gathering (as many useful web pages as possible, and the links interconnecting them)
- Today’s focus: issues arising when developing a web crawler

Overview

Features of a crawler
Crawling process
Architecture of a crawler
Web crawling and distributed indexes
Connectivity servers

Features of a crawler

- **Robustness**: ability to handle spider-traps (cycles, dynamic web pages)
- **Politeness**: policies about the frequency of robot visits
- **Distribution**: crawling should be distributed within several machines
- **Scalability**: crawling should be extensible by adding machines, extending bandwidth, etc.
- **Efficiency**: clever use of the processor, memory, bandwidth (e.g. as few idle processes as possible)
Features of a crawler (continued)

- **Quality**: should detect the most useful pages, to be indexed first
- **Freshness**: should continuously crawl the web (visiting frequency of a page should be close to its modification frequency)
- **Extensibility**: should support new data formats (e.g. XML-based formats), new protocols (e.g. ftp), etc.

Overview

Crawling process

- (a) The crawler begins with a seed set of URLs to fetch
- (b) The crawler fetches and parses the corresponding webpages, and extracts both text and links
- (c) The text is fed to a text indexer, the links (URL) are added to a URL frontier (≈ crawling agenda)
- (d) (continuous crawling) Already fetched URLs are appended to the URL frontier for later re-processing
  \[\Rightarrow\text{traversal of the web graph}\]

Crawling process (continued)

- Reference point: fetching a billion pages in a month-long crawl requires fetching several hundred pages each second
- Some potential issues:
  - Links encountered during parsing may be relative paths
    \[\Rightarrow\] normalization needed
  - Pages of a given web site may contain several duplicated links
  - Some links may point to robot-free areas
Overview

Architectural overview of a crawler.

Features of a crawler

Crawling process

Architecture of a crawler

Web crawling and distributed indexes

Connectivity servers

Architecture of a crawler

Several modules interacting:

- **URL frontier** – managing the URLs to be fetched
- **DNS resolution** – determining the host (web server) from which to fetch a page defined by a URL
- **Fetching module** – downloading a remote webpage for processing
- **Parsing module** – extracting text and links
- **Duplicate elimination** – detecting URLs and contents that have been processed a short time ago

Some remarks:

- During parsing, the text (with HTML tags) is passed on to the indexer, so are the links contained in the page (links analysis)
- links normalization
- links are checked before being added to the frontier
- - URL duplicates
- - content duplicates and nearly-duplicates
- - robots policy checking (robots.txt file) (*)

User-agent: *
Disallow: /subsite/temp

- priority score assignment (most useful links)
- robustness via checkpoints

Architecture of a crawler (continued)

Figure from (Manning et al., 2008)
About distributed crawling

- A crawling operation can be performed by several dedicated threads.
- Parallel crawlings can be distributed over nodes of a distributed system (geographical distribution, link-based distribution, etc.).
- Distribution involves a host-splitter, which dispatches URLs to the corresponding crawling node.
  - The duplicate elimination module cannot use a cache for fingerprints/shingles since duplicates do not necessarily belong to the same domain.
  - Documents change over time, potential duplicates may have to be added to the frontier.

About distributed crawling (continued)

Figure from (Manning et al., 2008)

About DNS resolution

- DNS resolution: translation of a server name and domain into an IP address.
- Resolution done by contacting a DNS server (which may itself contact other DNS servers).
- DNS resolution is a bottleneck in crawling (due to its recursive nature).
  - Use of a DNS cache (recently asked names).
- DNS resolution difficulty: lookup is synchronous (new request processed only when the current request is completed).
  - Thread \( i \) sends a request to a DNS server, if the time-out is reached or a signal from another thread is received, then it resumes.
  - In case of time-out, 5 attempts (with increasing time-outs ranging from 1 to 90 sec. cf Mercator system).

About the URL frontier

- Considerations governing the order in which URLs are extracted from the frontier:
  - (a) High quality pages updated frequently should have higher priority for frequent crawling.
  - (b) Politeness should be obeyed (no repeated fetches sent to a given host).
- A frontier should:
  - (i) Give a priority score to URLs reflecting their quality.
  - (ii) Open only one connection at a time to any host.
  - (iii) Wait a few seconds between successive requests to a host.
About the URL frontier (continued)

- 2 types of queues:
  - front queues for prioritization
  - back queues for politeness

- Each back queue only contains URLs for a given host (mapping between hosts and back queue identifiers)

- When a back queue is empty, it is filled with URLs from priority queues

- A heap contains the earliest time $t_e$ to contact a given host again

URL frontier extraction process:
(a) extraction of the root of the heap → back queue $j$
(b) extraction of the URL $u$ at the head of back queue $j$
(c) fetching of the web page defined by $u$
(d) if the queue $j$ is empty, selection of a front queue using a biasing function (random priority)
(e) selection of the head URL $v$ of the selected front queue
(f) does host($v$) already have a back queue? if yes, $v$ goes there, otherwise, $v$ is stored in back queue $j$
(g) the heap is updated for back queue $j$

Figure from (Manning et al., 2008)

Remarks about the URL frontier:
- The number of front queues + the biasing function implement the priority properties of the crawler
- The number of back queues implement the capacity of the crawler to avoid wasting time (by keeping the threads busy)
Overview

Features of a crawler

Crawling process

Architecture of a crawler

Web crawling and distributed indexes

Connectivity servers

Web crawling and distributed indexes

- High cooperation between crawlers and indexers
- Indexes are distributed over a large computer cluster
- 2 main partitioning techniques:
  - term partitioning (multi word queries are harder to process)
  - document partitioning (inverse document frequencies are harder to compute → background processes)
- In document partitioned indexes, a hash function maps index URL to nodes so that crawlers know where to send the extracted text
- In document partitioned indexes, documents that are most likely to score highly (cf links) are gathered → low score partitions used only when there are too few results

Connectivity servers

- Quality of a page is a function of the links that points to it
- For link analysis, queries on the connectivity of the web graph are needed
- Connectivity servers store information such as:
  - which URLs point to a given URL?
  - which URLs are pointed by a given URL?
- Mappings stored:
  - URL ↔ out-links
  - URL ↔ in-links
Connectivity servers (continued)

- Size needed to store the connectivity underlying the web graph?
- Estimate: 4 billion pages, 10 links per page, 4 bytes to encode a link extremity (i.e. URL id):
  \[ 4 \times 10^9 \times 10 \times 8 = 3.2 \times 10^{11} \text{ mbytes} \]
- Graph compression is needed to ensure efficient connectivity queries processing
- Graph encoded as adjacency tables:
  - \textit{in-table}: row \(\equiv\) page \(p\), columns \(\equiv\) links to \(p\)
  - \textit{out-table}: row \(\equiv\) page \(p\), columns \(\equiv\) links from \(p\)
- Space saved by using tables instead of list of links: \(50\%\)

Connectivity servers (continued)

- Compression based on the following ideas:
  - There is a large similarity between rows (e.g. menus)
  - The links tend to point to “nearby” page
    \(\rightarrow\) gap between URL ids in sorted list can be use (i.e. offset rather than absolute id)

Connectivity servers (continued)

- (a): Each URL is associated with an integer \(i\), where \(i\) is the position of the URL in the sorted list of URLs
- (b): Contiguous rows are noticed to have similar links (cf locality via menus)
- (c): Each row of the tables is encoded in terms of the 7 preceding rows
  - offset expressed within 3 bits
  - 7 to avoid expansive search among preceding queries
  - gap encoding inside a row
    - 1: 2, 5, 8, 12, 18, 24
    - 2: 2, 4, 8, 12, 18, 24
    - 1: 2, 3, 3, 4, 6, 6
    - 2: row1 -5 +4

Connectivity servers (continued)

- (d): Querying:
  - index lookup for a URL \(\rightarrow\) row id
  - row reconstruction (threshold in the number of redirection within preceding rows)
Conclusion

- Crawler: robots fetching and parsing web pages via a traversal of the web graph
- Issues of crawling related to traps, politeness, dependency to DNS servers and quality scoring
- Core component of a crawler: URL frontier (FIFO architecture used to sort the next URLs to process)
- Next week: link analysis

References

- C. Manning, P. Raghavan and H. Schütze
  *Introduction to Information Retrieval*

- Allan Heydon and Marc Najork
  http://citeseer.ist.psu.edu/heydon99mercator.html

- Sergey Brin and Lawrence Page
  http://citeseer.ist.psu.edu/brin98anatomy.html

- http://www.robotstxt.org/