Web Crawling

- Web crawlers spend a lot of time waiting for responses to requests
- To reduce this inefficiency, web crawlers use threads and fetch hundreds of pages at once
- Crawlers could potentially flood sites with requests for pages
- To avoid this problem, web crawlers use *politeness policies*
  - e.g., delay between requests to same web server
URL frontier: two main considerations

- **Politeness**: do not hit a web server too frequently
- **Freshness**: crawl some pages more often than others
  - E.g., pages (such as News sites) whose content changes often

These goals may conflict with each other.
(E.g., simple priority queue fails – many links out of a page go to its own site, creating a burst of accesses to that site.)

Explicit and implicit politeness

- **Explicit politeness**: specifications from webmasters on what portions of site can be crawled
  - robots.txt
- **Implicit politeness**: even with no specification, avoid hitting any site too often
Controlling Crawling

- Even crawling a site slowly will anger some web server administrators, who object to any copying of their data
- Robots.txt file can be used to control crawlers

```
User-agent: *
Disallow: /private/
Disallow: /confidential/
Disallow: /other/
Allow: /other/public/

User-agent: FavoredCrawler
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
```

Robots.txt example

- No robot should visit any URL starting with "/yoursite/temp/", except the robot called "searchengine":

```
User-agent: *
Disallow: /yoursite/temp/

User-agent: searchengine
Disallow:
```
Politeness – challenges

• Even if we restrict only one thread to fetch from a host, can hit it repeatedly
• Common heuristic: insert time gap between successive requests to a host that is >> time for most recent fetch from that host

Freshness

• Web pages are constantly being added, deleted, and modified
• Web crawler must continually revisit pages it has already crawled to see if they have changed in order to maintain the freshness of the document collection
  – stale copies no longer reflect the real contents of the web pages
Freshness

• HTTP protocol has a special request type called HEAD that makes it easy to check for page changes
  – returns information about page, not page itself

Client request: HEAD /csinfo/people.html HTTP/1.1
Host: www.cs.umass.edu

HTTP/1.1 200 OK
Date: Thu, 03 Apr 2008 05:17:54 GMT
Server: Apache/2.0.52 (CentOS)
Last-Modified: Fri, 04 Jan 2008 15:28:39 GMT

Server response: ETag: "239c33-2576-2a2837c0"
Accept-Ranges: bytes
Content-Length: 9590
Connection: close
Content-Type: text/html; charset=ISO-8859-1

Freshness

• Not possible to constantly check all pages
  – must check important pages and pages that change frequently
• Freshness is the proportion of pages that are fresh
• Optimizing for this metric can lead to bad decisions, such as not crawling popular sites
• Age is a better metric
**Age**

- Expected age of a page $t$ days after it was last crawled:
  \[
  \text{Age}(\lambda, t) = \int_0^t P(\text{page changed at time } x)(t - x)\,dx
  \]

- Web page updates follow the Poisson distribution on average
  - time until the next update is governed by an exponential distribution
  \[
  \text{Age}(\lambda, t) = \int_0^t \lambda e^{-\lambda x}(t - x)\,dx
  \]
Age

- The older a page gets, the more it costs not to crawl it
  - e.g., expected age with mean change frequency $\lambda = 1/7$ (one change per week)

![Graph showing age over time]

URL frontier: Mercator scheme

- URLs
  - Prioritizer
    - $K$ front queues
      - Biased front queue selector
        - Back queue router
          - $B$ back queues
            - Single host on each
              - Back queue selector
                - Crawl thread requesting URL
Mercator URL frontier

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

- Prioritizer assigns to URL an integer priority between 1 and $K$
  - Appends URL to corresponding queue
- Heuristics for assigning priority
  - Refresh rate sampled from previous crawls
  - Application-specific (e.g., “crawl news sites more often”)

Biased front queue selector

- When a back queue requests a URL (in a sequence to be described): picks a front queue from which to pull a URL
- This choice can be round robin biased to queues of higher priority, or some more sophisticated variant
  - Can be randomized
Back queues

- Biased front queue selector
- Back queue router

Back queue heap

- One entry for each back queue
- The entry is the earliest time $t_e$ at which the host corresponding to the back queue can be hit again
- This earliest time is determined from
  - Last access to that host
  - Any time buffer heuristic we choose
Back queue processing

- A crawler thread seeking a URL to crawl:
  - Extracts the root of the heap
  - Fetches URL at head of corresponding back queue $q$ (look up from table)
  - Checks if queue $q$ is now empty – if so, pulls a URL $v$ from front queues
    - If there’s already a back queue for $v$’s host, append $v$ to it and pull another URL from front queues, repeat
    - Else add $v$ to $q$
  - When $q$ is non-empty, create heap entry for it

Focused Crawling

- Attempts to download only those pages that are about a particular topic
  - used by vertical search applications
- Rely on the fact that pages about a topic tend to have links to other pages on the same topic
  - popular pages for a topic are typically used as seeds
- Crawler uses text classifier to decide whether a page is on topic