Web Crawling
Introduction to Information Retrieval
INF 141
Donald J. Patterson

Content adapted from Hinrich Schütze
http://www.informationretrieval.org
Web Crawlers
A Robust Crawl Architecture

Robust Crawling

WWW

DNS

Fetch

Parse

Doc. Fingerprints

Seen?

Robots.txt

URL Filter

Duplicate Elimination

URL Index

URL Frontier Queue
When a fetched document is parsed

- some outlink URLs are relative

For example:

- http://en.wikipedia.org/wiki/Main_Page
- has a link to “/wiki/Special:Statistics”
- which is the same as

Parsing involves normalizing (expanding) relative URLs
• Duplication is widespread on the web
• If a page just fetched is already in the index, don’t process it any further
• This can be done by using document fingerprints/shingles
• A type of hashing scheme
A Robust Crawl Architecture

WWW -> DNS -> Fetch -> Parse -> Seen? -> URL Filter -> Duplicate Elimination -> URL Index

URL Frontier Queue
Compliance with webmasters wishes...

- Robots.txt
  - Filters is a regular expression for a URL to be excluded
  - How often do you check robots.txt?
    - Cache to avoid using bandwidth and loading web server
- Sitemaps
  - A mechanism to better manage the URL frontier
A Robust Crawl Architecture
Duplicate Elimination

- For a one-time crawl
  - Test to see if an extracted, parsed, filtered URL
    - has already been sent to the frontier.
    - has already been indexed.

- For a continuous crawl
  - See full frontier implementation:
    - Update the URL’s priority
      - Based on staleness
      - Based on quality
      - Based on politeness
Distributing the crawl

- The key goal for the architecture of a distributed crawl is **cache locality**
- We want multiple crawl threads in multiple processes at multiple nodes for robustness
  - Geographically distributed for speed
- Partition the hosts being crawled across nodes
  - Hash typically used for partition
- How do the nodes communicate?
Robust Crawling

The output of the URL Filter at each node is sent to the Duplicate Eliminator at all other nodes.
URL Frontier

• Freshness
  • Crawl some pages more often than others
    • Keep track of change rate of sites
    • Incorporate sitemap info

• Quality
  • High quality pages should be prioritized
  • Based on link-analysis, popularity, heuristics on content

• Politeness
  • When was the last time you hit a server?
• Freshness, Quality and Politeness
  • These goals will conflict with each other
  • A simple priority queue will fail because links are bursty
    • Many sites have lots of links pointing to themselves creating bursty references
  • Time influences the priority
• Politeness Challenges
  • Even if only one thread is assigned to hit a particular host it can hit it repeatedly
• Heuristic: insert a time gap between successive requests
Magnitude of the crawl

- To fetch 1,000,000,000 pages in one month...
- a small fraction of the web
- we need to fetch 400 pages per second!
- Since many fetches will be duplicates, unfetchable, filtered, etc. 400 pages per second isn’t fast enough
Overview

- Introduction
- URL Frontier
- Robust Crawling
  - DNS
  - Various parts of architecture
- URL Frontier
- Index
  - Distributed Indices
- Connectivity Servers
Robust Crawling

The output of the URL Filter at each node is sent to the Duplicate Eliminator at all other nodes.
URLs flow from top to bottom
• Front queues manage priority
• Back queue manage politeness
• Each queue is FIFO
URL Frontier Implementation - Mercator

Front queues

- Prioritizer takes URLs and assigns a priority
  - Integer between 1 and F
  - Appends URL to appropriate queue

- Priority
  - Based on rate of change
  - Based on quality (spam)
  - Based on application
Back queues

- Selection from front queues is initiated from back queues
- Pick a front queue, how?
  - Round robin
  - Randomly
  - Monte Carlo
  - Biased toward high priority
Back queues

- Each back queue is non-empty while crawling
- Each back queue has URLs from one host only
- Maintain a table of URL to back queues (mapping) to help
Back queues

- Timing Heap
  - One entry per queue
  - Has earliest time that a host can be hit again
- Earliest time based on
  - Last access to that host
  - Plus any appropriate heuristic

[Diagram of URL Frontier Implementation - Mercator]

- Back Queue Router
- Host to Back Queue Mapping Table
- B "Back" Queues
- Back Queue Selector
- Timing Heap
Back queues

- A crawler thread needs a URL
- It gets the timing heap root
- It gets the next eligible queue based on time, b.
- It gets a URL from b
- If b is empty
- Pull a URL v from front queue
- If back queue for v exists place it in that queue, repeat.
- Else add v to b - update heap.
URL Frontier Implementation - Mercator

Back queues

- How many queues?
- Keep all threads busy
- ~3 times as many back queues as crawler threads

Web-scale issues
- This won’t fit in memory

Solution
- Keep queues on disk and keep a portion in memory.