CS 572: Information Retrieval

Web Crawling

Acknowledgements
Some slides in this lecture are adapted from Chris Manning (Stanford) and Soumen Chakrabarti (IIT Bombay)
Status

• Project 1 results sent
• Final project: feedback on initial proposals sent
• Final project: updated proposal due Monday
Final Project Guidelines

- Groups of 2-3 students
- Evaluation: Proposal: 10%. Progress: 20%. Final Prez: 20%. Results/Report: 50%

Steps:

✓ Initial project proposal: **Monday March 21st**
  - Feedback Wednesday, March 23
  - Short 5-minute “idea pitch” to class (can still adjust/re-organize)

1. Final project proposal: **March 28**
   - ~ 1.5 page
   - The project will be evaluated against the final proposal

2. Intermediate progress presentation (short): **April 20**

3. Presentation & Report: **First week of May**
   - Short in-class presentation, in-class feedback
   - Report (5 pages maximum): few days after presentation.
Final Project Proposals: Monday 3/28

• Project name, team member(s)
1. 1 paragraph: problem and goal
   Main project goal: problem addressed, why important?
2. 1-2 paragraphs: approach
   Sketch of planned approach to the problem
3. 1 paragraph: evaluation
   Sketch of planned experiments, metrics, datasets
   Milestones to achieve by April 20th
   What will be finished by then? Think carefully what you will work on first.
5. 1 paragraph: potential issues and backup plan
   What if approach in #2 does not work? Backup plan?
6. Special requirements?
   Any special hardware/software (virtual web server)? Datasets?
Web Search Engine

- Web
- Crawl: Which pages to crawl
- Build Index: Efficient index order
- Answer Queries: Informs dynamic ranking
Crawling Goals

- Quality
  - General Search Engine Crawlers
  - Research and Archive Crawlers
  - Mirroring Systems
- Freshness
  - Focused and Personal Crawlers
- Quantity

Basic crawler operation

• Begin with known “seed” URLs
• Fetch and parse them
  – Extract URLs they point to
  – Place the extracted URLs on a queue
• Fetch each URL on the queue and repeat
Crawling picture

URLs crawled and parsed

Seed pages

Web

Unseen Web

URLs frontier
Simple picture – complications

- Web crawling isn’t feasible with one machine
  - All of the above steps distributed
- Malicious pages
  - Spam pages
  - Spider traps – incl dynamically generated
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers vary
  - Webmasters’ stipulations
    - How “deep” should you crawl a site’s URL hierarchy?
    - Site mirrors and duplicate pages
- Politeness – don’t hit a server too often
What any crawler *must* do

- **Be Polite**: Respect implicit and explicit politeness considerations
  - Only crawl allowed pages
  - Respect *robots.txt* (more on this shortly)

- **Be Robust**: Be immune to spider traps and other malicious behavior from web servers
What any crawler *should* do

- Be capable of **distributed** operation: designed to run on multiple distributed machines
- Be **scalable**: designed to increase the crawl rate by adding more machines
- Performance/efficiency: permit full use of available processing and network resources
What any crawler *should* do

- Fetch pages of “higher **quality**” first
- **Continuous** operation: Continue fetching fresh copies of a previously fetched page
- **Extensible**: Adapt to new data formats, protocols
Manager
Long term scheduling

Harvester
Short-term sched.
Network transfers

Gatherer
Parse pages and extract links

Seeder
Resolve links

Pages

Tasks

URLs

Documents
Queue of Web sites
*(long-term scheduling)*

Queue of Web pages for each site
*(short-term scheduling)*
Formal Problem

• Find a sequence of page requests \((p,t)\) that:
  
  – Optimizes a function of the volume, quality and freshness of the pages
  – Has a bounded crawling time
  – Fulfils politenessness
  – Maximizes the use of local bandwidth

• Must be on-line: how much knowledge?
Updated crawling picture

URLs crawled and parsed

Seed Pages

Crawling thread

Unseen Web

URL frontier

Sec. 20.1.1
URL frontier

• Can include multiple pages from the same host
• Must avoid trying to fetch them all at the same time
• Must try to keep all crawling threads busy
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
Robots.txt

- Protocol for giving spiders ("robots") limited access to a website, originally from 1994
  - [www.robotstxt.org/wc/norobots.html](http://www.robotstxt.org/wc/norobots.html)
- Website announces its request on what can(not) be crawled
  - For a URL, create a file `URL/robots.txt`
  - This file specifies access restrictions
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:

• Real Examples:
  – http://www.emory.edu/robots.txt
Processing steps in crawling

• Pick a URL from the frontier
• Fetch the document at the URL
• Parse the URL
  – Extract links from it to other docs (URLs)
• Check if URL has content already seen
  – If not, add to indexes
• For each extracted URL
  – Ensure it passes certain URL filter tests
  – Check if it is already in the frontier (duplicate URL elimination)

Which one?

E.g., only crawl .edu, obey robots.txt, etc.
Basic crawl architecture

WWW

DNS

Fetch

Parse

Content seen?

Doc FP’s

robots filters

URL set

URL Frontier

Fetch

URL Frontier

Sec. 20.2.1
DNS (Domain Name Server)

- A lookup service on the internet
  - Given a URL, retrieve its IP address
  - Service provided by a distributed set of servers – thus, lookup latencies can be high (even seconds)

- Common OS implementations of DNS lookup are **blocking**: only one outstanding request at a time

- Solutions
  - DNS caching
  - Batch DNS resolver – collects requests and sends them out together
Large-scale crawlers: performance and reliability considerations

• Need to fetch many pages at same time
  – utilize the network bandwidth
  – single page fetch may involve several seconds of network latency

• Highly concurrent and parallelized DNS lookups

• Use of asynchronous sockets
  – Explicit encoding of the state of a fetch context in a data structure
  – Polling socket to check for completion of network transfers
  – Multi-processing or multi-threading: Impractical

• Care in URL extraction
  – Eliminating duplicates to reduce redundant fetches
  – Avoiding “spider traps”
DNS caching, pre-fetching and resolution

- A customized DNS component with…..
  1. Custom client for address resolution
  2. Caching server
  3. Prefetching client
Custom client for address resolution

- Tailored for concurrent handling of multiple outstanding requests
- Allows issuing of many resolution requests together
  - polling at a later time for completion of individual requests
- Facilitates load distribution among many DNS servers.
Caching server

- With a large cache, persistent across DNS restarts
- Residing largely in memory if possible.
Prefetching client

• Steps
  1. Parse a page that has just been fetched
  2. extract host names from HREF targets
  3. Make DNS resolution requests to the caching server

• Usually implemented using UDP
  – User Datagram Protocol
  – connectionless, packet-based communication protocol
  – does not guarantee packet delivery

• Does not wait for resolution to be completed.
Multiple concurrent fetches

• Managing multiple concurrent connections
  – A single download may take several seconds
  – Open many socket connections to different HTTP servers simultaneously

• Multi-CPU machines not useful
  – crawling performance limited by network and disk

• Two approaches
  1. using multi-threading
  2. using non-blocking sockets with event handlers
When a fetched document is parsed, some of the extracted links are *relative* URLs.

E.g., at `http://en.wikipedia.org/wiki/Main_Page` we have a relative link to `/wiki/Wikipedia:General_disclaimer` which is the same as the absolute URL `http://en.wikipedia.org/wiki/Wikipedia:General_disclaimer`.

During parsing, we must normalize (expand) such relative URLs.
Link extraction and normalization

- **Goal**: Obtaining a canonical form of URL
- **URL processing and filtering**
  - Avoid multiple fetches of pages known by different URLs
  - many IP addresses
    - For load balancing on large sites
      - Mirrored contents/contents on same file system
    - “Proxy pass“
      - Mapping of different host names to a single IP address
      - need to publish many logical sites
  - **Relative URLs**
    - need to be interpreted w.r.t to a base URL.
Canonical URL

Formed by

• Using a standard string for the protocol
• Canonicalizing the host name
• Adding an explicit port number
• Normalizing and cleaning up the path
Avoiding repeated expansion of links on duplicate pages

• Reduce redundancy in crawls
• Duplicate detection
  – Mirrored Web pages and sites
• Detecting exact duplicates
  – Checking against MD5 digests of stored URLs
  – Representing a relative link \( v \) (relative to aliases \( u_1 \) and \( u_2 \)) as tuples \( (h(u_1); v) \) and \( (h(u_2); v) \)
• Detecting near-duplicates
  – Even a single altered character will completely change the digest!
    • E.g.: date of update/ name and email of the site administrator
  – Solution: Shingling
Spider traps

- Protecting from crashing on
  - Ill-formed HTML
    • E.g.: page with 68 kB of null characters
  - Misleading sites
    • indefinite number of pages dynamically generated by CGI scripts
    • paths of arbitrary depth created using soft directory links and path remapping features in HTTP server
Spider Traps: Solutions

- No automatic technique can be foolproof
- Check for URL length
- Guards
  - Preparing regular crawl statistics
  - Adding dominating sites to guard module
  - Disable crawling active content such as CGI form queries
  - Eliminate URLs with non-textual data types
Distributing the crawler

- Run multiple crawl threads, under different processes – potentially at different nodes
  - Geographically distributed nodes
- Partition hosts being crawled into nodes
  - Hash used for partition
- How do these nodes communicate?
Communication between nodes

- The output of the URL filter at each node is sent to the Duplicate URL Eliminator at all nodes.
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 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.)
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
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

Biased front queue selector
Back queue router
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

Diagram:
- 1
- B
- Heap
- Back queue selector
Back queue invariants

• Each back queue is kept non-empty while the crawl is in progress

• Each back queue only contains URLs from a single host
  – Maintain a table from hosts to back queues

<table>
<thead>
<tr>
<th>Host name</th>
<th>Back queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>
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 \( q \) and pull another URL from front queues, repeat
    – Else add \( v \) to \( q \)
  • When \( q \) is non-empty, create heap entry for it
Number of back queues $B$

- Keep all threads busy while respecting politeness
- Mercator recommendation: three times as many back queues as crawler threads
Number of back queues $B$

• Keep all threads busy while respecting politeness
• Mercator recommendation: three times as many back queues as crawler threads
Basic crawl architecture

WWW → DNS → Fetch → Parse → Content seen? → Doc FP’s → robots filters → URL set → Dup URL elim

URL Frontier
Text repository

- Crawler’s last task
  - Dumping fetched pages into a repository
- Decoupling crawler from other functions for efficiency and reliability preferred
- Page-related information stored in two parts
  - meta-data
  - page contents.
Storage of page-related information

- Meta-data
  - relational in nature
    - usually managed by custom software to avoid relation database system overheads
    - text index involves bulk updates
  - includes fields like content-type, last-modified date, content-length, HTTP status code, etc.
Page contents storage

- Typical HTML Web page compresses to 2-4 kB (using zlib)
- File systems have a 4-8 kB file block size
  - Too large !!
- Page storage managed by custom storage manager
  - simple access methods for
    - crawler to add pages
    - Subsequent programs (Indexer etc) to retrieve documents
Page Storage

- Small-scale systems
  - Repository fitting within the disks of a single machine
  - Use of storage manager (E.g.: Berkeley DB)
    - Manage disk-based databases within a single file
    - configuration as a hash-table/B-tree for URL access key
      - To handle ordered access of pages
    - configuration as a sequential log of page records.
      - Since Indexer can handle pages in any order
Large-scale crawlers often use multiple ISPs and a bank of local storage servers to store the pages crawled.
Page Storage

• Large Scale systems
  ▪ Repository distributed over a number of storage servers
  ▪ Storage servers
    ▪ Connected to the crawler through a fast local network (E.g.: Ethernet)
    ▪ Hashed by URLs
  ▪ ‘T3’ grade leased lines.
    ▪ To handle 10 million pages (40 GB) per hour
The Big Picture

User:
- Search queries
- Query selection
- Needy queries

Crawler:
- Crawled pages
- Uncrawled pages
- Link extractor

WWW:

Diagram:
- User sends search queries to the crawler.
- The crawler processes the search queries and extracts links from crawled pages.
- Uncrawled pages are added to the crawler's queue for further processing.
- The crawler incrementally crawls the web, adding pages to the crawled and uncrawled databases.
- Needy queries are generated based on the crawled and uncrawled pages.
Refreshing crawled pages

• Search engine's index should be fresh
• Web-scale crawler never `completes' its job
• High variance of rate of page changes
• “If-modified-since” request header with HTTP protocol
  ▪ Impractical for a crawler
• Solution
  ▪ At commencement of new crawling round estimate which pages have changed
Page Refreshing Schemes

• PageRank:
  – Crawl/Refresh according to “importance”

• Staleness-based:
  – Minimize # “stale” pages

• EBR: Embarrassment
  – Page clicked not relevant

• User-centric:
  – Priority optimized according to repository “quality”, measured by likelihood stale/fresh page viewed (est. past query workload)
Determining page changes

• “Expires” HTTP response header
  ▪ For page that come with an expiry date

• Otherwise need to guess if revisiting that page will yield a modified version.
  ▪ Score reflecting probability of page being modified
  ▪ Crawler fetches URLs in decreasing order of score.
  ▪ Assumption : recent past predicts the future
Estimating page change rates

• Brewington and Cybenko & Cho
  ▪ Algorithms for maintaining a crawl in which most pages are fresher than a specified epoch.

• Prerequisite
  ▪ average interval at which crawler checks for changes is smaller than the inter-modification times of a page

• Small scale intermediate crawler runs
  ▪ to monitor fast changing sites
    ▪ E.g.: current news, weather, etc.
  ▪ Patched intermediate indices into master index
Page quality

- Page-rank / in-degree
- Spam score
- ...

- At ranking time: given two documents with equal TF-IDF scores, prefer higher quality document
Page Change Frequency

• Recrawl pages which change often
  – News updates
  – Blogs

• Predict change frequency
Crawl Ordering by Search Impact
WSDM 2008

Sandeep Pandey
Christopher Olston
Yahoo! Research
Selecting pages to crawl next

**Goal:** Crawl discovered pages

**Challenges:**
- Huge number of pages
- Varying quality
- Quality is hard to judge beforehand
Crawling Objective

- acquire pages that show up in query results (impact)

Query result lists:

Objective: acquire the top part

US election  Super Bowl  Britney  Yahoo!
Impact of Crawling Page $p$

- $\text{Impact}(p) = \sum_{\text{queries } q} \text{freq}(q) \times \text{top-}K(p,q)$

- $\text{top-}K(p,q) = \begin{cases} 1 & \text{if } p \text{ is in top-}K \text{ results of } q, \\ 0 & \text{otherwise} \end{cases}$

- **Ideal approach:** Crawl high impact pages

- **Standard approach:** Crawl high prestige pages
  - e.g., Pagerank or approximation thereof [Najork et. al. WWW’01; Abiteboul et. al. WWW’03]
prestige ≠ impact

prestige-based priority list

impact-based priority list

URL: silverscape.com/.../Product_Positioning

bottom 20% of prestige; top 1% of impact ("product positioning")
prestige ≠ impact

prestige-based priority list

impact-based priority list

URL: pc2sms.eu

top 1% of prestige; low impact (relevant for “send free SMS”, but not in top-10)
Poor Correlation Between Prestige and Impact

![Graph showing poor correlation between prestige and impact]
Ranking Crawled Pages

Query → Page

Content-dependent features

Content-independent features

score $S(p,q)$

score dist. $S'(p,q)$
Ranking Crawled & Uncrawled Pages

• “Query sketch” for query q:

Crawled pages

Uncrawled pages

S(p,q)

P1

S′(p,q)

P2

S′(p,q)

P3

S′(p,q)

P4

S′(p,q)

P5

S′(p,q)

P6

P7

P8

P9
Selecting Pages to Crawl

- Objective: maximize total impact of crawled pages
- Constraint: crawl C pages only

\[
\text{total impact} = \sum C \sum_{\text{queries } q} \text{freq}(q) \times \text{top-K}(p,q)
\]

\[
\text{top-K}(p,q) = \begin{cases} 
1 & \text{if } p \text{ is in top-k of } q \\
0 & \text{otherwise}
\end{cases}
\]
Selecting Pages to Crawl

$q_1$

$q_2$

$q_3$

$q_4$
Solution: hybrid impact estimation

• 2 ways to estimate impact
  – Using past workload
  – Using prestige

• Combine their estimations
  – linear weighted combination
  – Impact-based = 0.9 ; prestige-based = 0.1
Example 1

1. YotaTech
   YotaTech is a Toyota truck and SUV discussion forum powered by vBulletin. YotaTech Knowledge Base \* Forums > Toyota SUV & Truck Tech > Offroad Tech & Fab Shop > ...
   www.yotatech.com - 49k - Cached - More from this site

2. Toyota Forum - Home
   Mambo - the dynamic portal engine and content management system ... Download SFSP/Iriskit nyt udsendte til Toyota Hiace - Stærkere dieselmotorer - Forbedret ...
   www.toyota-forum.dk - 39k - Cached - More from this site

3. 4x4Wire.com's TrailTalk Forums. Viewing forum. Early Toyota Trucks
   Toyota Forums: Early Toyota Trucks | 4Runner & SUV | T100 & Tundra | Tacoma ... 3 registered and 33 anonymous users are browsing this forum ...
   www.4x4wire.com/forums/postlist.php?Cat=4&Board=UBE11 - 39k - Cached - More from this site

4. Toyota Forums - Topix
   Toyota Forum. Forums and message boards for Toyota. Toyota News. Forum. Wire ... happening on all Topix forums. Toyota News. Fee considered for 'editar' ...
   www.topix.net/forum/autos/toyota - 83k - Cached - More from this site

hybrid policy

prestige-based policy
Example 2

1. **Texhoma Schools' Home Page**
   Information and Activities of Texhoma Oklahoma's Schools ... Texhoma Times, Volume 3, Issue 14. Information on Hanta Virus. Contact Web Page Author ...
   www.texhoma54.net - 12k - Cached - More from this site

2. **Texhoma OK/TX Cemetery**
   Texhoma Menu and Front Page. Baker Cemetery, Bethel Cemetery, Goodwill ... Oslo Church Cemetery. Texhoma Panhandle Pioneers, in rootsweb.com, by Bob Fleming ...
   www.texhoma.us/cemetery/cemetery.htm - 10k - Cached - More from this site

3. **Texhoma, Texas, Elementary School**
   ... rating given to Texas Schools, and places Texhoma in the top group of over 1000 ... of Interest. Oklahoma Side, 9th-12th. Town of Texhoma ...
   www.texhomaisd.net - 5k - Cached - More from this site

4. **Texhoma's Location and History**
   Texhoma, Texas, Renews "1948" Fiesta Day's Old Timers, 1958, 1960, 1974. Annual ... Ten Decades of Texhoma, Centennial Book. Local History Books Available ...
   www.texhoma.us/history.htm - 12k - Cached - More from this site

---

**Centennial Year 1901-2001**
Now 104 Years Old
Our Location and Our History

**So Big It Takes Two States To Hold Us!**
We're on the Oklahoma-Texas State line

---

**hybrid policy**

**prestige-based policy**
Resources

• MRS Ch. 20
• Mercator: A scalable, extensible web crawler (Heydon et al. 1999)
• A standard for robot exclusion
• Open-Source web crawlers:
  – Heritrix: http://crawler.archive.org/
  – Nutch: http://lucene.apache.org/nutch/