Content from the Web
Servers + Crawlers

Class Overview

Other Cool Stuff
- Query processing
- Content Analysis
- Indexing
- Crawling
- Document Layer
- Network Layer

A Closeup View

10/13 - Crawlers
10/15 - DL ∈ atrium
10/20 - No class
10/22 - IR, indexing
10/27 - Alta Vista
10/29 - No class

Group Meetings

Today

• Search Engine Overview
• HTTP
• Crawlers
• Server Architecture

Standard Web Search Engine Architecture

Crawling

user query

store documents, check for duplicates, extract links

create an inverted index

show results to user

search engine servers

inverted index

DocIds

store documents, check for duplicates, extract links

crawl the web

DocIds
Indexing

- What data is necessary?
- Format?
- Compression?
- Efficient Creation

Query Processing

- Efficient Processing
- Ranking

Precision and Recall

- **Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant)

<table>
<thead>
<tr>
<th></th>
<th>Relevant</th>
<th>Not Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>tp</td>
<td>fp</td>
</tr>
<tr>
<td>Not Retrieved</td>
<td>fn</td>
<td>tn</td>
</tr>
</tbody>
</table>

- Precision $P = \frac{tp}{tp + fp}$
- Recall $R = \frac{tp}{tp + fn}$

But Really

- Precision & Recall are too simple
- Evaluation is a very thorny problem

Precision & Recall

- **Precision**: Proportion of selected items that are correct
  \[ \frac{tp}{tp + fp} \]
- **Recall**: % of target items that were selected
  \[ \frac{tp}{tp + fn} \]

Precision-Recall curve

- Shows tradeoff

What Did I Forget?

- Little Change to UI
- Faceted Interfaces
- Personalization
- Revisiting
Your Project Architecture?

- Crawl the web
- Store documents, check for duplicates, extract links
- Classify?
- Extract
- Relational DB
- DocIds

Outline

- Search Engine Overview
- HTTP
- Crawlers
- Server Architecture

Connecting on the WWW

What happens when you click?

- Suppose
  - You are at www.yahoo.com/index.html
  - You click on www.grippy.org/mattmarg/
- Browser uses DNS => IP addr for www.grippy.org
- Opens TCP connection to that address
- Sends HTTP request:

```
GET /mattmarg/ HTTP/1.0
User-Agent: Mozilla/2.0 (Macintosh; i; PPC)
Accept: text/html, */*
Cookie: name = value
Referer: http://www.yahoo.com/index.html
Host: www.grippy.org
```

HTTP Response

```
HTTP/1.0 200 Found
Date: Mon, 10 Feb 1997 23:48:22 GMT
Server: Apache/1.1.1 HotWired/1.0
Content-type: text/html
Last-Modified: Tues, 11 Feb 1999 22:45:55 GMT
Image/jpeg, ...
```

- One click => several responses
- HTTP1.0: new TCP connection for each elt/page
- HTTP1.1: KeepAlive - several requests/connection
Response Status Lines

- **1xx** Informational
- **2xx** Success
  - 200 Ok
- **3xx** Redirection
  - 302 Moved Temporarily
- **4xx** Client Error
  - 404 Not Found
- **5xx** Server Error

HTTP Methods

- **GET**
  - Bring back a page
- **HEAD**
  - Like GET but just return headers
- **POST**
  - Used to send data to server to be processed (e.g. CGI)
    - Different from GET:
      - A block of data is sent with the request, in the body, usually with extra headers like **Content-Type** and **Content-Length**.
      - Request URL is not a resource to retrieve; it's a program to handle the data being sent
      - HTTP response is normally program output, not a static file.
  - **PUT, DELETE, ...**

HTTP Secure

- **HTTPS**
  - Secure connections
  - Encryption: SSL/TLS
  - Fairly straightforward:
    - Agree on crypto protocol
    - Exchange keys
    - Create a shared key
    - Use shared key to encrypt data
  - Certificates

Logging Web Activity

- Most servers support “common logfile format” or “extended logfile format”
- Apache lets you customize format
- Every HTTP event is recorded
  - Page requested
  - Remote host
  - Browser type
  - Referring page
  - Time of day
- Applications of data-mining logfiles ??

Cookies

- Small piece of info
  - Sent by server as part of response header
  - Stored on disk by browser; returned in request header
  - May have expiration date (deleted from disk)
- Associated with a specific domain & directory
  - Only given to site where originally made
  - Many sites have multiple cookies
  - Some have multiple cookies per page!
- Most Data stored as name=value pairs
- See
  - C:\Program Files\Netscape\Users\default\cookies.txt
  - C:\WINDOWS\Cookies

HTTPS

CRAWLERS...
Danger Will Robinson!!

- Consequences of a bug

Max 6 hits/server/minute

plus...

http://www.cs.washington.edu/lab/policies/crawlers.html

Open-Source Crawlers

- **GNU Wget**
  - Utility for downloading files from the Web.
  - Fine if you just need to fetch files from 2-3 sites.

- **Heritix**
  - Open-source, extensible, Web-scale crawler
  - Easy to get running.
  - Web-based UI

- **Nutch**
  - Featureful, industrial strength, Web search package.
  - Includes Lucene information retrieval part
    - TF/IDF and other document ranking
    - Optimized, inverted-index data store
  - You get complete control thru easy programming.

Search Engine Architecture

- **Crawler (Spider)**
  - Searches the web to find pages. Follows hyperlinks.
  - Never stops

- **Indexer**
  - Produces data structures for fast searching of all words in the pages

- **Retriever**
  - Query interface
  - Database lookup to find hits
    - 300 million documents
    - 300 GB RAM, terabytes of disk
  - Ranking, summaries

- **Front End**

Thinking about Efficiency

- **Clock cycle**: 2 GHz
  - Typically completes 2 instructions / cycle

- **Disk access**: 1-10ms
  - Depends on seek distance, published average is 5ms
  - Thus perform 200 seeks / sec
  - (And we are ignoring rotation and transfer times)

- **Disk is 20 Million times slower !!!**

- Store index in Oracle database?
- Store index using files and unix filesystem?

Spiders = Crawlers

- **1000s of spiders**
- **Various purposes:**
  - Search engines
  - Digital rights management
  - Advertising
  - Spam
  - Link checking – site validation

Spiders (Crawlers, Bots)

- **Queue := initial page URL**
- **Do forever**
  - Dequeue URL
  - Fetch P
  - Parse P for more URLs; add them to queue
  - Pass P to (specialized?) indexing program

- **Issues...**
  - Which page to look at next?
    - keywords, recency, focus, ???
  - Avoid overloading a site
  - How deep within a site to go?
  - How frequently to visit pages?
  - Traps!
Crawling Issues

- Storage efficiency
- Search strategy
  - Where to start
  - Link ordering
  - Circularities
  - Duplicates
  - Checking for changes
- Politeness
  - Forbidden zones: robots.txt
  - CGI & scripts
  - Load on remote servers
  - Bandwidth (download what need)
- Parsing pages for links
- Scalability
- Malicious servers: SEOs

Robot Exclusion

- Person may not want certain pages indexed.
- Crawlers should obey Robot Exclusion Protocol.
  - But some don’t
- Look for file robots.txt at highest directory level
  - If domain is www.ecom.cmu.edu, robots.txt goes in www.ecom.cmu.edu/robots.txt
- Specific document can be shielded from a crawler by adding the line:
  <META NAME="ROBOTS" CONTENT="NOINDEX">

Robots Exclusion Protocol

- Format of robots.txt
  - Two fields. User-agent to specify a robot
  - Disallow to tell the agent what to ignore
- To exclude all robots from a server:
  User-agent: *
  Disallow: /
- To exclude one robot from two directories:
  User-agent: WebCrawler
  Disallow: /news/
  Disallow: /tmp/

Danger, Danger

- Ensure that your crawler obeys robots.txt.
- Don’t make any of these specific gaffes.
- Provide contact info in user-agent field.
- Monitor the email address
- Notify the CS Lab Staff
- Honor all Do Not Scan requests
- Post any "stop-scanning" requests
- "The scanee is always right."
- Max 6 hits/server/minute

Outgoing Links?

- Parse HTML...
- Looking for…what?

Which tags / attributes hold URLs?

Anchor tag: <a href="URL" … > … </a>
Option tag: <option value="URL"…> … </option>
Map: <area href="URL" …>
Frame: <frame src="URL" …>
Link to an image: <img src="URL" …>
Relative path vs. absolute path: <base href= …>
Bonus problem: Javascript
  In our favor: Search Engine Optimization
Web Crawling Strategy

- Starting location(s)
- Traversal order
  - Depth first (LIFO)
  - Breadth first (FIFO)
  - Or ???
- Politeness
- Cycles?
- Coverage?

Structure of Mercator Spider

1. Remove URL from queue
2. Simulate network protocols & REP
3. Read w/ RewindInputStream (RIS)
4. Has document been seen before? (checksums and fingerprints)
5. Extract links
6. Download new URL?
7. Has URL been seen before?
8. Add URL to frontier

URL Frontier (priority queue)

- Most crawlers do breadth-first search from seeds.
- Politeness constraint: don’t hammer servers!
  - Obvious implementation: “live host table”
  - Will it fit in memory?
  - Is this efficient?
- Mercator’s politeness:
  - One FIFO subqueue per thread.
  - Choose subqueue by hashing host’s name.
  - Dequeue first URL whose host has NO outstanding requests.

Fetching Pages

- Need to support http, ftp, gopher, ....
  - Extensible!
- Need to fetch multiple pages at once.
- Need to cache as much as possible
  - DNS
  - robots.txt
  - Documents themselves (for later processing)
- Need to be defensive!
  - Need to time out http connections.
  - Watch for “crawler traps” (e.g., infinite URL names.)
  - See section 5 of Mercator paper.
  - Use URL filter module
  - Checkpointing!

Duplicate Detection

- URL-seen test: has URL been seen before?
  - To save space, store a hash
- Content-seen test: different URL, same doc.
  - Suppress link extraction from mirrored pages.
- What to save for each doc?
  - 64 bit “document fingerprint”
  - Minimize number of disk reads upon retrieval.

Nutch: A simple architecture

- Seed set
- Crawl
- Remove duplicates
- Extract URLs (minus those we’ve been to)
  - new frontier
- Crawl again
- Can do this with Map/Reduce architecture
  - How?
Mercator Statistics

<table>
<thead>
<tr>
<th>PAGE TYPE</th>
<th>PERCENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>text/html</td>
<td>69.2%</td>
</tr>
<tr>
<td>image/gif</td>
<td>17.9%</td>
</tr>
<tr>
<td>image/jpeg</td>
<td>8.1%</td>
</tr>
<tr>
<td>text/plain</td>
<td>1.5%</td>
</tr>
<tr>
<td>pdf</td>
<td>0.9%</td>
</tr>
<tr>
<td>audio</td>
<td>0.4%</td>
</tr>
<tr>
<td>zip</td>
<td>0.4%</td>
</tr>
<tr>
<td>postscript</td>
<td>0.3%</td>
</tr>
<tr>
<td>other</td>
<td>1.4%</td>
</tr>
</tbody>
</table>

Exponentially increasing size

Advanced Crawling Issues

- **Limited resources**
  - Fetch most *important* pages first
- **Topic specific search engines**
  - Only care about pages which are *relevant* to topic

  "Focused crawling"

- **Minimize stale pages**
  - Efficient re-fetch to keep index timely
  - How track the rate of change for pages?

Focused Crawling

- Priority queue instead of FIFO.
- How to determine priority?
  - Similarity of page to driving query
  - Use traditional IR measures
  - Exploration / exploitation problem
  - Backlink
    - How many links point to this page?
  - PageRank (Google)
    - Some links to this page count more than others
  - Forward link of a page
  - Location Heuristics
    - E.g., Is site in .edu?
    - E.g., Does URL contain ‘home’ in it?
  - Linear combination of above

Outline

- Search Engine Overview
- HTTP
- Crawlers
- Server Architecture

Server Architecture

Connecting on the WWW
Client-Side View
- Content rendering engine
  - Tags, positioning, movement
- Scripting language interpreter
  - Document object model
  - Events
  - Programming language itself
- Link to custom Java VM
- Security access mechanisms
- Plugin architecture + plugins

Server-Side View
- Database-driven content
- Lots of Users
- Scalability
- Load balancing
- Often implemented with cluster of PCs
- 24x7 Reliability
- Transparent upgrades

Trade-offs in Client/Server Arch.
- **Compute on clients?**
  - Complexity: Many different browsers
    - Firefox, IE, Safari, … × Version × OS
  - **Compute on servers?**
    - Peak load, reliability, capital investment.
      + Access anywhere, anytime, any device
      + Groupware support (shared calendar, …)
      + Lower overall cost (utilization & debugging)
      + Simpler to update service

Dynamic Content
- We want to do more via an http request
  - E.g. we’d like to invoke code to run on the server.
  - **Initial solution:** Common Gateway Interface (CGI) programs.
  - Example: web page contains form that needs to be processed on server.

CGI Code
- CGI scripts can be in any language.
- A new process is started (and terminated) with each script invocation (overhead!).
- **Improvement I:**
  - Run some code on the client’s machine
  - E.g., catch missing fields in the form.
- **Improvement II:**
  - Server APIs (but these are server-specific).

Java Servlets
- Servlets: applets that run on the server.
  - Java VM stays, servlets run as threads.
- Accept data from client + perform computation
- Platform-independent alternative to CGI.
  - Can handle multiple requests concurrently
    - Synchronize requests - use for online conferencing
  - Can forward requests to other servers
    - Use for load balancing
Java Server Pages (JSP)
Active Server Pages (ASP)
- Allows mixing static HTML w/ dynamically generated content
- JSP is more convenient than servlets for the above purpose
- More recently PHP (and Ruby on Rails, sort of) fall in this category

```html
<html>
<head>
<title>Example #3</title>
</head>
<? print(Date("m/j/y"); ?>
<body>
</body>
</html>
```

AJAX
- Getting the browser to behave like your applications (caveat: Asynchronous)
- Client → Rendering library (Javascript)
  - Widgets
- Talks to Server (XML)
- How do we keep state?
- Over the wire protocol: SOAP/XML-RPC/etc.

Connecting on the WWW

Tiered Architectures
1-tier = dumb terminal → smart server.
2-tier = client/server.
3-tier = client/application server/database.
Why decompose the server?

Two-Tier Architecture

Three-Tier Architecture
Getting to ‘Giant Scale’
• Only real option is cluster computing

Optional Backplane:
System-wide network for intra-server traffic:
Query redirect, coherence traffic for store, updates, ...

Typical Cluster
• Extreme symmetry
• Internal disks
• No monitors
• No visible cables
• No people!
• Offsite management
• Contracts limit
  – Power
  – Temperature

From: Brewer Lessons from Giant-Scale Services
Images from Zillow talk

High Availability
• Essential Objective
• Phone network, railways, water system
• Challenges
  – Component failures
  – Constantly evolving features
  – Unpredictable growth

From: Brewer Lessons from Giant-Scale Services

Architecture
• What do faults impact? Yield? Harvest?
• Replicated systems
  – Faults → reduced capacity (hence, yield @ high util)
• Partitioned systems
  – Faults → reduced harvest
  – Capacity (queries / sec) unchanged
  – DQ Principle
  – 3 physical bottleneck
  – Data/Query × Queries/Sec = Constant

From: Brewer Lessons from Giant-Scale Services

Graceful Degradation
• Too expensive to avoid saturation
• Peak/average ratio
  – 1.6x - 6x or more
  – Moviefone: 10x capacity for Phantom Menace
    • Not enough...
• Dependent faults (temperature, power)
  – Overall DQ drops way down
• Cutting harvest by 2 doubles capacity…

From: Brewer Lessons from Giant-Scale Services

Admission Control (AC) Techniques
• Cost-Based AC
  – Denying an expensive query allows 2 cheap ones
  – Inkomi
• Priority-Based (Value-Based) AC
  – Stock trades vs. quotes
  – Datek
• Reduced Data Freshness