

Web Search Engines and Search Technology

techniques, tools, architectures, and current trends

CATT Short Course - July 12, 2002

Instructor: Prof. Torsten Suel

Department of Computer and Information Science

Polytechnic University

suel@poly.edu

<http://cis.poly.edu/suel>

This material was developed for a one-day short course on search engines and search technology, given at Polytechnic University in Brooklyn on July 12, 2002. The course was organized by the Center for Advanced Technology in Telecommunication (CATT) at Polytechnic University; see <http://catt.poly.edu> for more information.

An electronic version of these slides (except for handwritten additions) is available at

<http://cis.poly.edu/suel/webshort.ppt>

which allows you to follow the hyperlinks in the presentation. A detailed list of additional pointers and bibliographic references is available online at

<http://cis.poly.edu/suel/webshort/>

and will be updated periodically.

Purpose of this course:

- **learn how search engines work**
- **learn about other types of web search tools and applications**
- **learn how to build and deploy such tools**
 - basic information retrieval techniques
 - what software tools to use
 - system architectures and performance

Target Audience:

- **technically oriented people interested in how it works**
- **developers who may need to build or deploy tools**

Overview:

- **I - Introduction: (30 minutes)**
 - *Motivation (web and web search tools)*
 - *Examples of search tools*
- **II - Basic Techniques (75 minutes)**
 - *How the web works*
 - *Basic search engine architecture*
 - *Crawling basics: following links, robot exclusion, ..*
 - *Storage*
 - *Indexing*
 - *Querying and term-based ranking*
 - *Text classification*

Overview: (cont.)

• **III - Search Applications & Tools** (45 minutes)

- *Types of search tools*
- *Available software systems and tools*
- *Example: Major search engine*
- *Example: Focused Data Collection and Analysis*
- *Example: Browsing/Search Assistants*
- *Example: Site and Enterprise Search*
- *Demonstration of search tools*
- *Using search engines*
- *Search engine optimization and manipulation*

Lunch (12:30 -1:30)

Overview: (cont.)

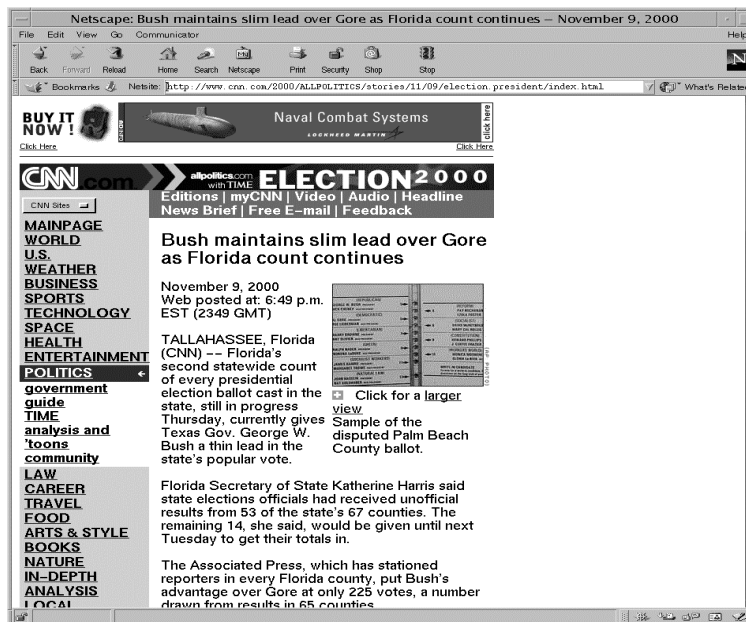
• **IV - Advanced Techniques** (afternoon)

- *High-performance crawling*
- *Recrawling and focused crawling*
- *Link-based ranking (Pagerank, HITS)*
- *Vector-space models and term-based ranking*
- *Integration of link- and term-based methods*
- *Meta search engines*
- *Parallel search engines and scaling*
- *Structural analysis of the web graph*
- *Document clustering and duplicate detection*

Not Covered:

- **Semi-structured data and XML**
- **Web accessible databases**
 - crawling the hidden web
 - efficient query processing on remote data sources
 - wrapper construction
- **Extracting relational data from the web**
- **Shopping bots**
- **Image and multimedia search**
- **Peer-to-peer search technologies**
- **advanced IR: categorization, clustering, ...**
- **natural language processing (NLP)**

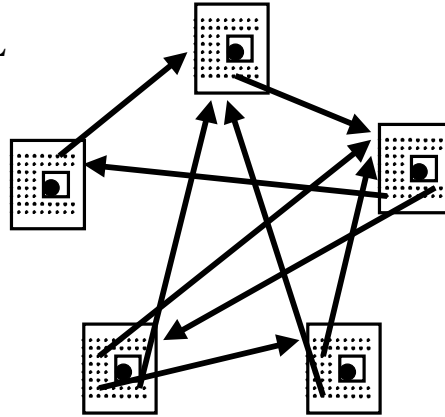
I - Introduction: What is the Web?



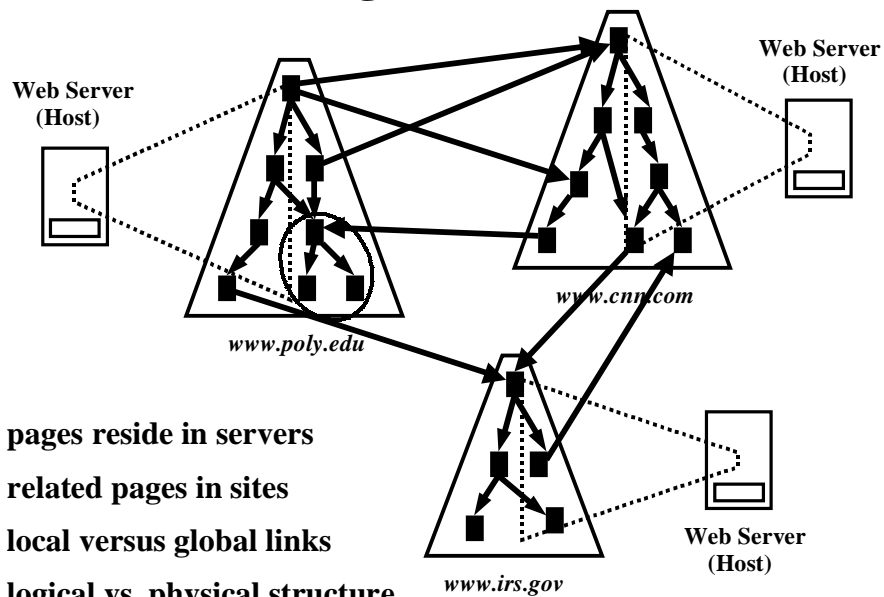
What is the web? (another view)

- pages containing (fairly unstructured) text
- images, audio, etc. embedded in pages
- structure defined using HTML
(*Hypertext Markup Language*)
- hyperlinks between pages!
- over 2 billion pages
- over 10 billion hyperlinks

➔ *a giant graph!*



How is the web organized?

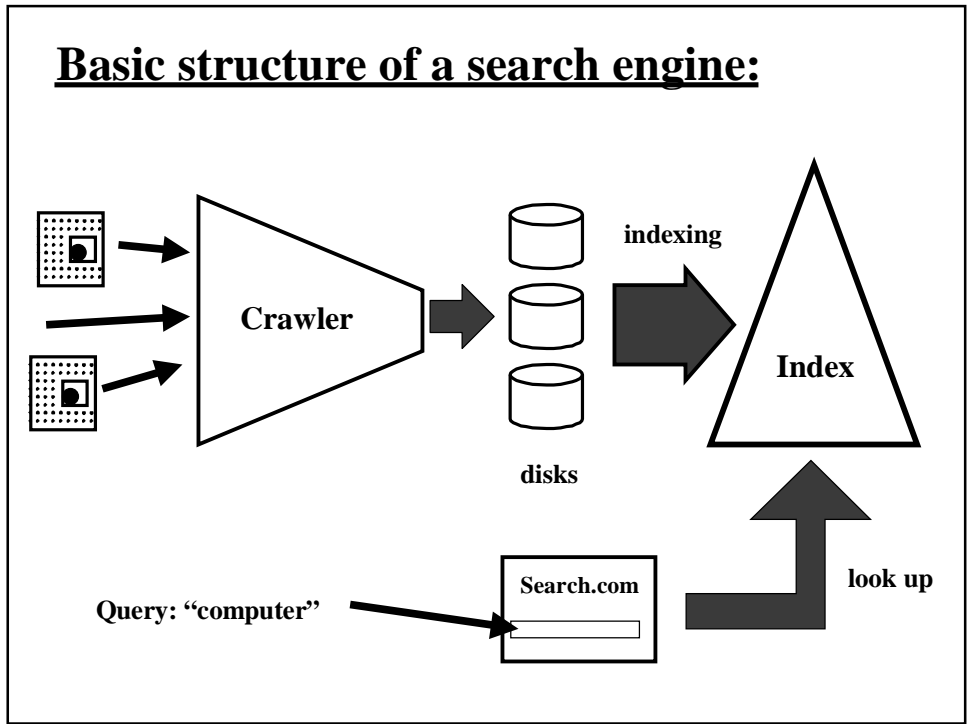


- pages reside in servers
- related pages in sites
- local versus global links
- logical vs. physical structure

Major search engines:



Basic structure of a search engine:



Ranking:

- return best pages first
- term- vs. link-based approaches

Advanced Search Preferences Search Tips

Google

Searched the web for **computer**. Results 1 - 10 of about 35,400,000. Search took 0.

COMPUTERS - Lowest Prices on all Computers! Spo
www.BizRate.com [Click Here to Stop Searching and Start SHOPPING!](#)

Category: [Computers](#) > [Computer Science](#) > [Journals](#)

Dell Computer - Laptop, Desktop, Workstation, Server
...Support.Dell.com Copyright 1999-2000 Dell Computer Corporation. For...
Description: Custom-build and order computers using the Dell online store. Design desktop computers, laptops, and...
Category: [Computers](#) > [Hardware](#) > [Notebooks and Laptops](#) > [By Manufacturer](#) > [Dell](#)
www.dell.com/ - 23k - [Cached](#) - [Similar pages](#)

IEEE Computer Society
...September 2000 Vote for Computer Society officers! (members only) Members...
...nonmembers alike: Explore the Computer Society's magazines, journals,...
Description: The IEEE Computer Society is the original and largest worldwide organization serving as the leading...
Category: [Science](#) > [Institutions](#) > [Associations](#) > [Computer Science](#)
www.computer.org/ - 27k - [Cached](#) - [Similar pages](#)

Computer und Internet - Excite Deutschland
...hier! Ihr Standort: Homepage > [Computer](#) [Internet](#) [Computer](#)...
...Inhalt: [Computer](#) & [Internet](#) [COMPUTER](#) [INTERNET](#) [Download](#)...
excite.de.netscape.com/computer/ - 28k - [Cached](#) - [Similar pages](#)

Challenges for search engines:

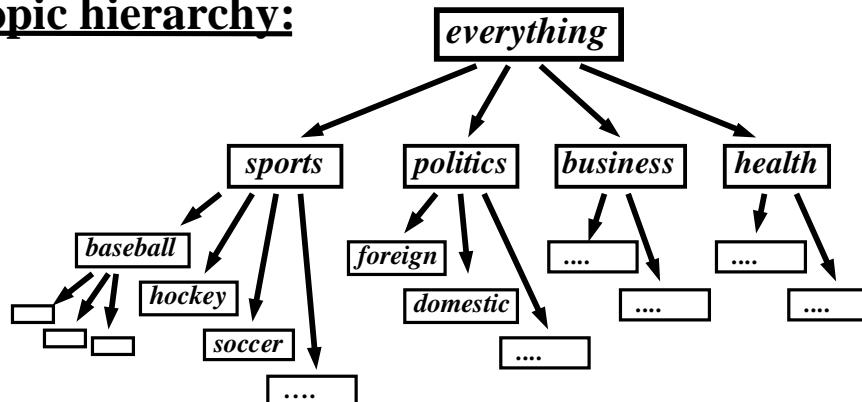
- coverage (need to cover large part of the web)
➔ *need to crawl and store massive data sets*
- good ranking (in the case of broad queries)
➔ *smart information retrieval techniques*
- freshness (need to update content)
➔ *frequent recrawling of content*
- user load (up to 10000 queries/sec - Google)
➔ *many queries on massive data*
- manipulation (sites want to be listed first)
➔ *naïve techniques will be exploited quickly*

Web directories: (*Yahoo, Open Directory Project*)

The screenshot shows the Yahoo! homepage with a search bar and various navigation links. Below the search bar, there are several categorized directory listings:

- Yahoo! Shopping** – Thousands of stores. Millions of products.
 - Departments:** Apparel, Beauty, Books, Computers
 - Electronics:** Music, Sports, Video/DVD
 - Stores:** Coach, Eddie Bauer, Patagonia, Victoria's Secret
 - Features:** Halloween, Free Shipping, PS2 Games, Razor Scooters
- Arts & Humanities:** Literature, Photography...
- Business & Economy:** B2B, Finance, Shopping, Jobs...
- Computers & Internet:** Internet, WWW, Software, Games...
- Education:** College and University, K-12...
- Entertainment:** Cool Links, Movies, Humor, Music...
- Government:** Elections, Military, Law, Taxes...
- Health:** Medicine, Diseases, Drugs, Fitness...
- News & Media:** Full Coverage, Newspapers, TV...
- Recreation & Sports:** Sports, Travel, Autos, Outdoors...
- Reference:** Libraries, Dictionaries, Quotations...
- Regional:** Countries, Regions, US States...
- Science:** Animals, Astronomy, Engineering...
- Social Science:** Archaeology, Economics, Languages...
- Society & Culture:** People, Environment, Religion...
- In the News:**
 - Violence escalates in West Bank, Gaza
 - Space station crew readies for tomorrow's launch
 - Gore, Bush enter last week of campaign battle
 - NFL, NHL, NBA
- Marketplace:**
 - PlayStation 2 – auctions, games, clubs, news
 - Y! Travel – plan your holiday travel
 - The X-Files official auction
 - Y! Store – build an online store in 10 minutes
- Broadcast Events:**
 - 5pm ET: Earnings – Barnes & Noble.com, Expedia
 - 7pm: Vertical Horizon audio chat
 - 9pm: Titans vs. Redskins
- Inside Yahoo!:**
 - new! Play free Fantasy NBA
 - Yahoo! Radio – tune in to your favorite station
 - Y! Politics – Election 2000

Topic hierarchy:



Challenges:

- designing topic hierarchy
- automatic classification: “what is this page about?”
- Yahoo and Open Directory mostly human-based

Specialized search engines: (*achoo, findlaw*)

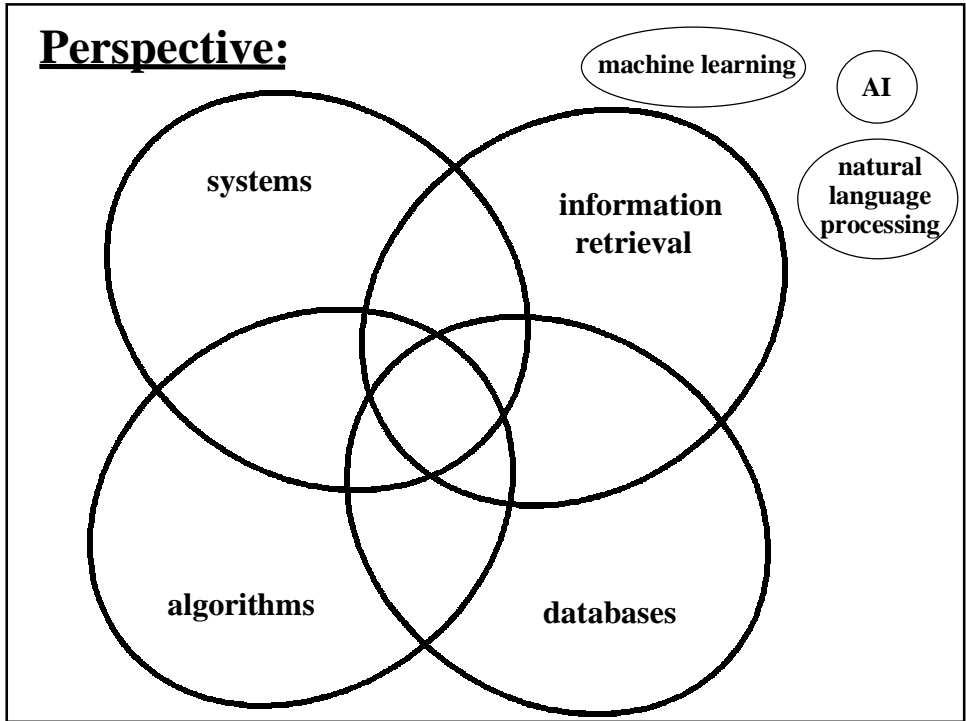
- be the best on one particular topic
- use domain-specific knowledge
- limited resources ➡ do not crawl the entire web!
- focused crawling techniques (or Meta search)

Meta search engines: (*dogpile, search.com, mamma*)

- uses other search engines to answer questions
- ask the right specialized search engine, or
- combine results from several large engines
- may need to be “familiar” with thousands of engines

Personal Search Assistants: (*Alexa, Google Toolbar*)

- embedded into browser
- can suggest “related pages”
- search by “highlighting text” ➡ can use context
- may exploit individual browsing behavior
- may collect and aggregate browsing information
➡ privacy issues
- architectures:
 - on top of crawler-based search engine (alex, google), or
 - based on meta search (*MIT Powerscout*)
 - based on limited crawls by client or proxy
(*MIT Letizia, Stanford Powerbrowser*)



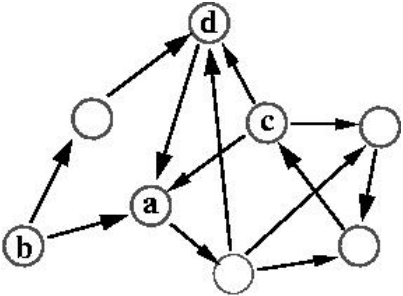
Example #1: Link-based ranking techniques

- **Ragerank (Brin&Page/Google)**

“significance of a page depends on significance of those referencing it”

- **HITS (Kleinberg/IBM)**

“Hubs and Authorities”



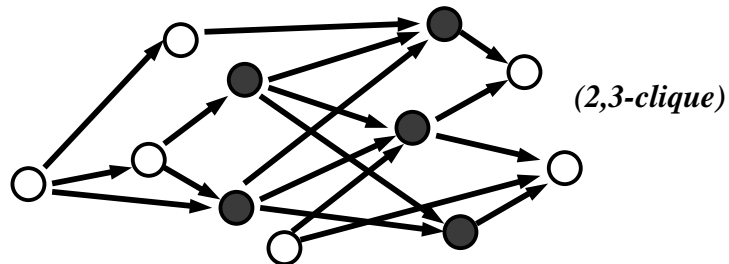
$s(a) \sim s(b) + s(c) + s(d) ?$

Example #2: Crawling 100 million pages

- crawler architecture
- networking requirements
- data structures: size and robustness
- crawling etiquette
- concerns for webmasters

Example #3: Analysis of the web graph

- What does the web look like?
(*diameter, connectivity, in-degree*)
- Why are there so many bipartite cliques? (IBM)
(*and why do we care?*)



- How do you compute with a 500 million node graph?

Example #4: Finding duplicates on the web

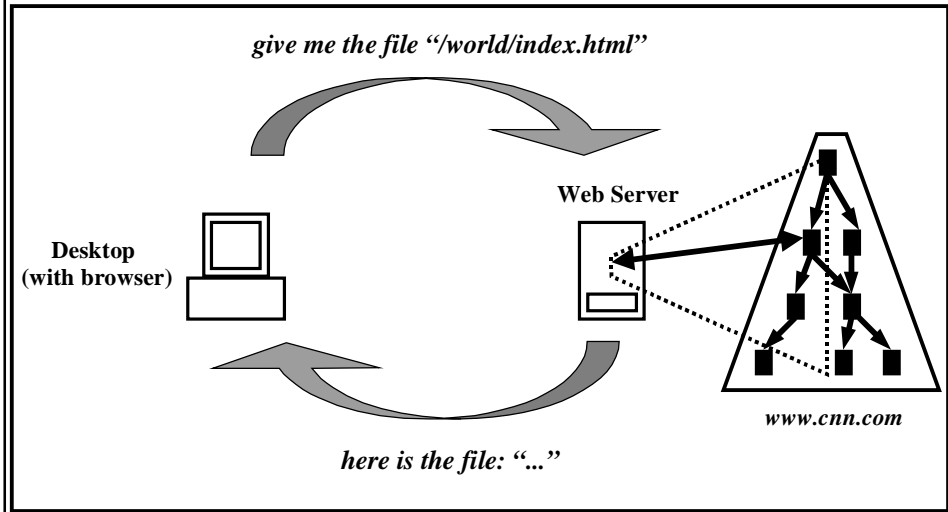
- **given 100 million pages, find duplicates (1.5 TB)**
- **more difficult: find similar pages (clustering)**
- **find mirror sites and replicated collections**
- **can you find them without crawling completely?**

II - Basic Techniques:

- **How the web works:** (*HTML, HTTP, DNS, web servers, ..*)
- **Basic search engine architecture** (*google, inktomi*)
- **Crawling:** (*following links, robot exclusion, black holes, ..*)
- **Storage**
- **Indexing:** (*inverted files, index compression, ..*)
- **Boolean querying and term-based ranking**
- **Text classification**

3 - How the web works *(more details)*

Fetching “www.cnn.com/world/index.html”



Three Main Ingredients:

- **Naming: URL (uniform resource locators)**
(used to identify and locate objects)
- **Communication: HTTP (hypertext transfer protocol)**
(used to request and transfer objects)
- **Rendering: HTML (hypertext markup language)**
(used to defined how object should be presented to user)

Client Server Paradigm:

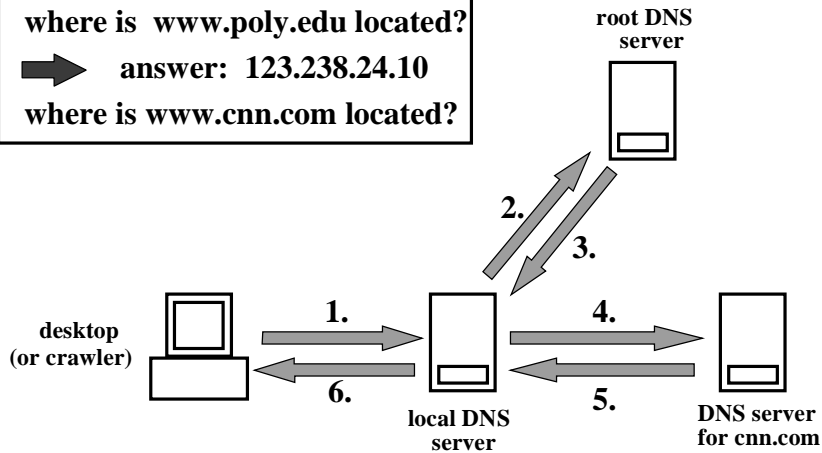
- Client (browser) used HTTP to ask server (web server) for object identified by URI, and renders this object according to rules defined by HTML

Domain Name Service:

where is `www.poly.edu` located?

➔ answer: `123.238.24.10`

where is `www.cnn.com` located?



Names, addresses, hosts, and sites

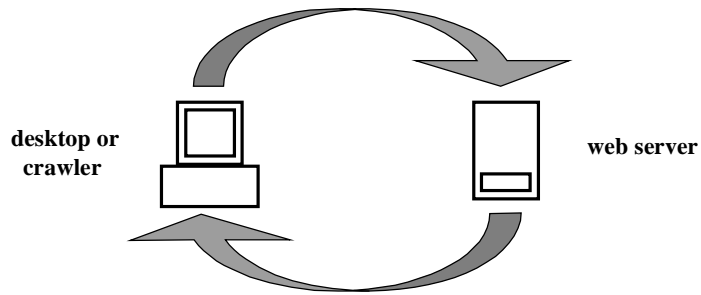
- one machine can have several host names and IP addresses
- one host name may correspond to several machines
- one host can have several “sites” (what is a site?)
- one “site” on several hosts
- issues: detecting duplicates, crawling, local vs. global links

```
weasel% nslookup www.cnn.com
Server: photon.poly.edu
Address: 128.238.32.22

Non-authoritative answer:
Name:   cnn.com
Addresses: 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28
          207.25.71.29, 207.25.71.30, 207.25.71.5, 207.25.71.6, 207.25.71.20
          207.25.71.22, 207.25.71.23, 207.25.71.24
Aliases: www.cnn.com
```

HTTP:

GET /world/index.html HTTP/1.0
User-Agent: Mozilla/3.0 (Windows 95/NT)
Host: www.cnn.com
From: ...
Referer: ...
If-Modified-Since: ...



HTTP/1.0 200 OK
Server: Netscape-Communications/1.1
Date: Tuesday, 8-Feb-99 01:22:04 GMT
Last-modified: Thursday, 3-Feb-99 10:44:11 GMT
Content-length: 5462
Content-type: text/html

<the html file>

HTML:



```
<HTML>
<HEAD>
  <TITLE> Some interesting links </TITLE>
</HEAD>
<body BGCOLOR="#FFFFFF" LINK="#0000FF" VLINK="000099">
  <H1>
Some interesting links
</H1>
  <p>
<strong>Search Engines:</strong><p>
<a href="http://www.google.com">Google Search Engine</a><br>
<a href="http://www.altavista.com">AltaVista Search</a>
  </BODY>
</HTML>
```

HTTP & HTML issues:

- “dynamic” URLs:

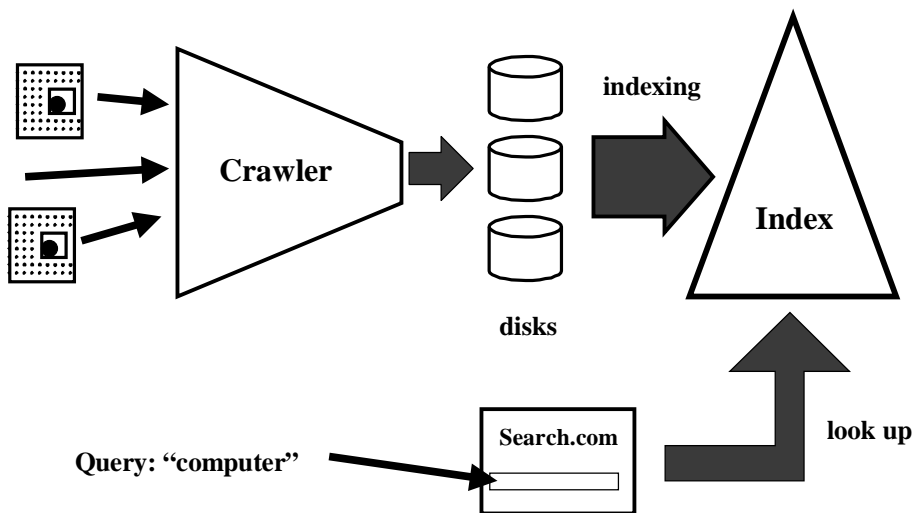
<http://www.google.com/search?q=brooklyn>

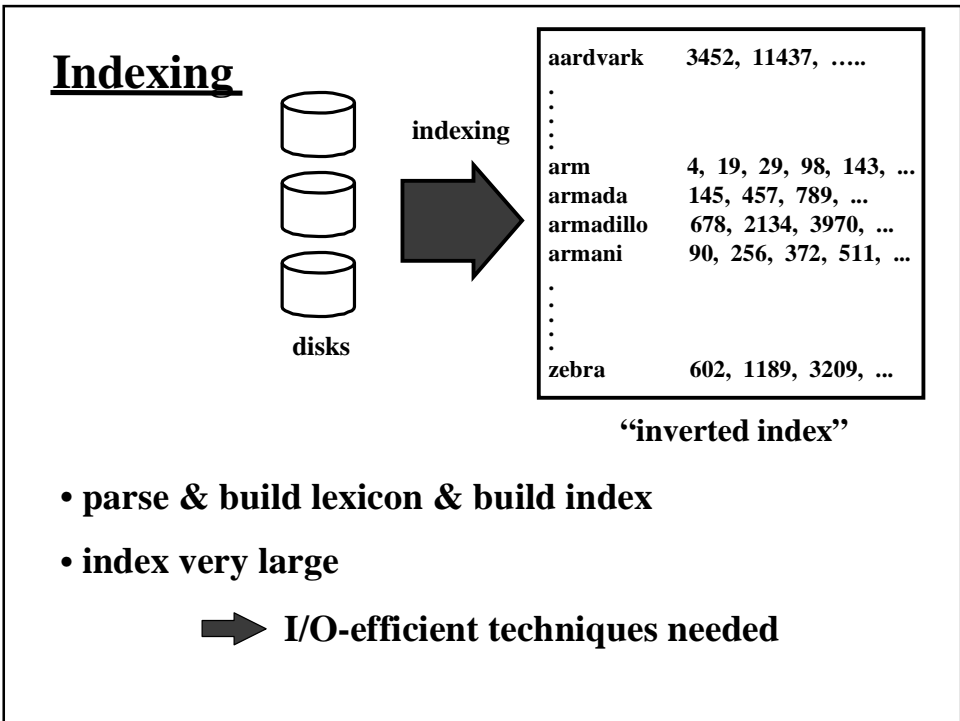
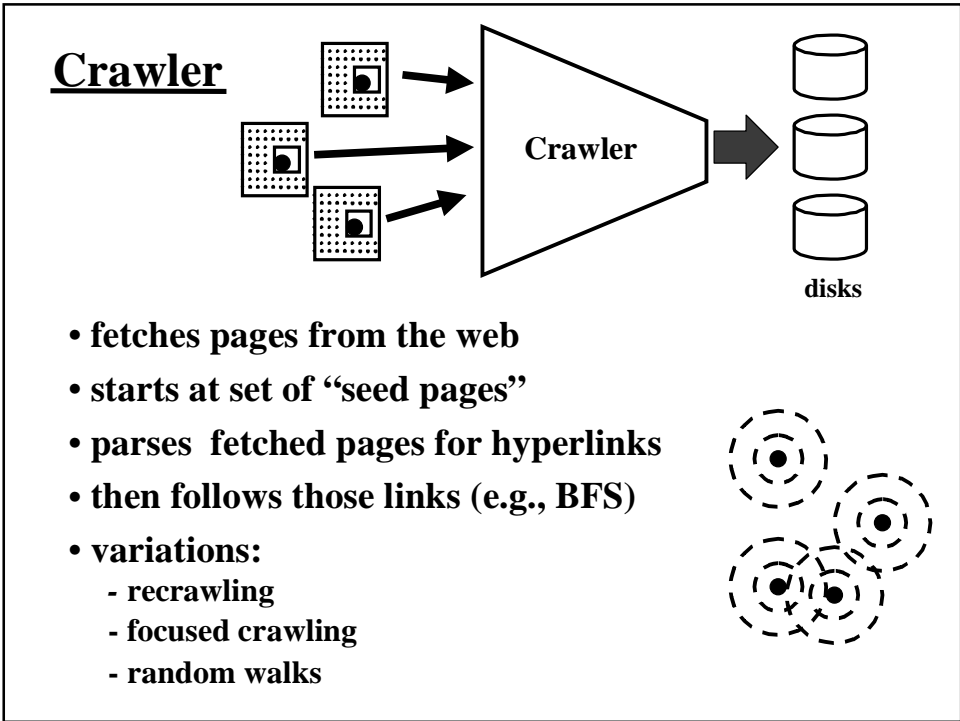
<http://www.amazon.com/exec/obidos/ASIN/1558605703/qid%3D9...>

<http://cis.poly.edu/search/search.cgi>

- result file can be computed by server in arbitrary manner!
- persistent connections in HTTP/1.1
- mime types and extensions
- frames
- redirects
- javascript/java/JEB/flash/activeX ????????

Search Engine Architecture:





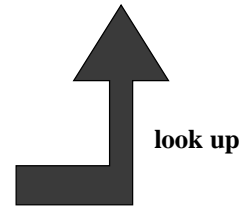
Querying

Boolean queries:

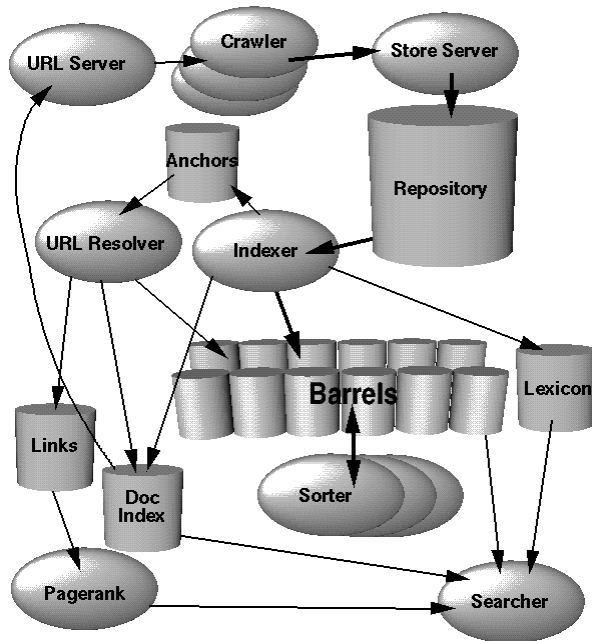
(zebra AND armadillo) OR armani

➔ unions/intersections of lists

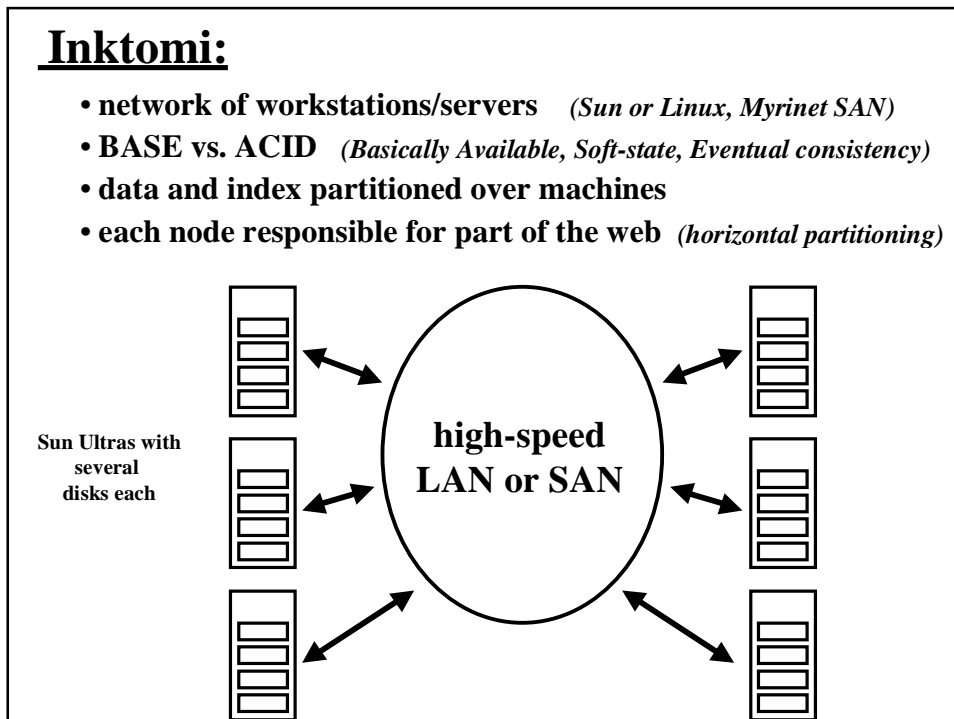
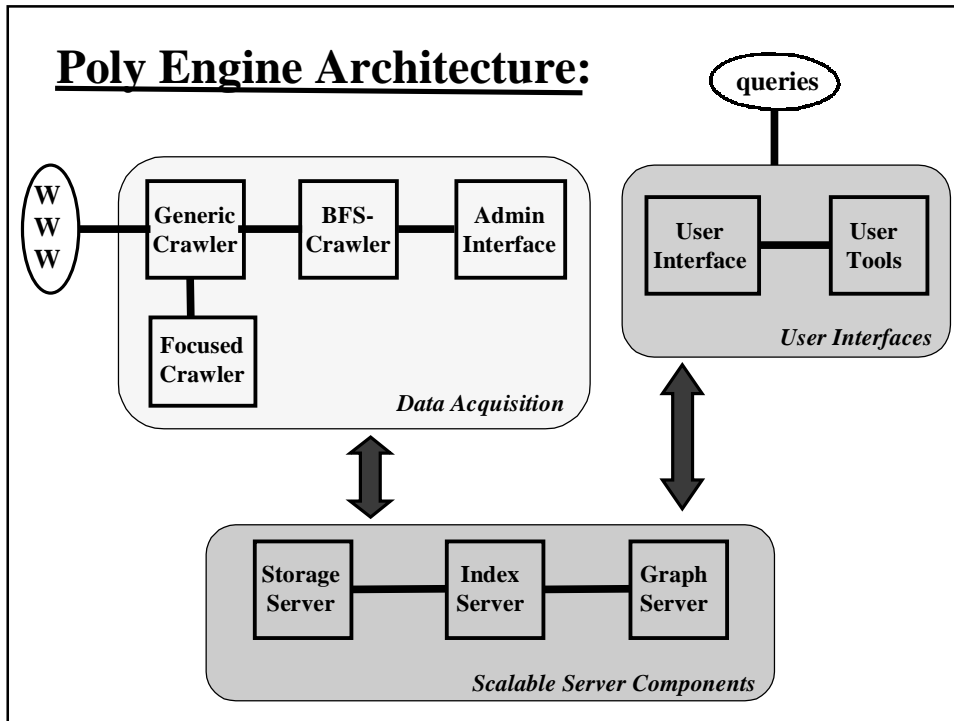
aardvark	3452, 11437,
...	...
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
...	...
zebra	602, 1189, 3209, ...



Google:



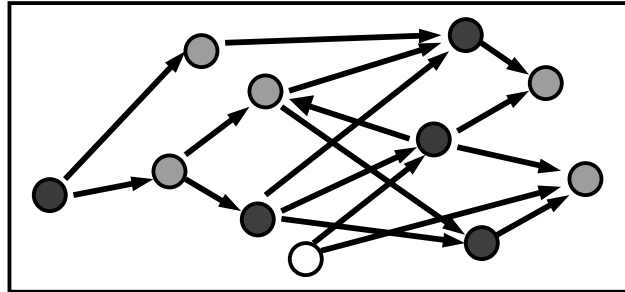
[Source: Brin/Page, WWW Conf., 1998]



Crawling the Web:

- **Basic idea:**

- start at a set of known URLs
- explore the web in “concentric circles” around these URLs



- start pages
- distance-one pages
- distance-two pages

Simple Breadth-First Search Crawler:

```
insert set of initial URLs into a queue Q
while Q is not empty
  currentURL = dequeue(Q)
  download page from currentURL
  for any hyperlink found in the page
    if hyperlink is to a new page
      enqueue hyperlink URL into Q
```

*this will eventually download all pages reachable from the start set
(also, need to remember pages that have already been downloaded)*

Traversal strategies: (why BFS?)

- **crawl will quickly spread all over the web**
- **load-balancing between servers**
- **in reality, more refined strategies** (*but still BFSish*)
- **many other strategies** (*focused crawls, recrawls, site crawls*)

Tools/languages for implementation:

- **Scripting languages** (*Python, Perl*)
- **Java** (*performance tuning tricky*)
- **C/C++ with sockets** (*low-level*)
- **available crawling tools** (*usually not completely scalable*)

Details: (lots of 'em) (*see this paper for details*)

- **handling filetypes**
(*exclude some extensions, and use mime types*)
- **URL extensions and CGI scripts**
(*to strip or not to strip? Ignore?*)
- **frames, imagemaps, base tags**
- **black holes (robot traps)**
(*limit maximum depth of a site*)
- **different names for same site?**
(*could check IP address, but no perfect solution*)

Performance considerations: later!

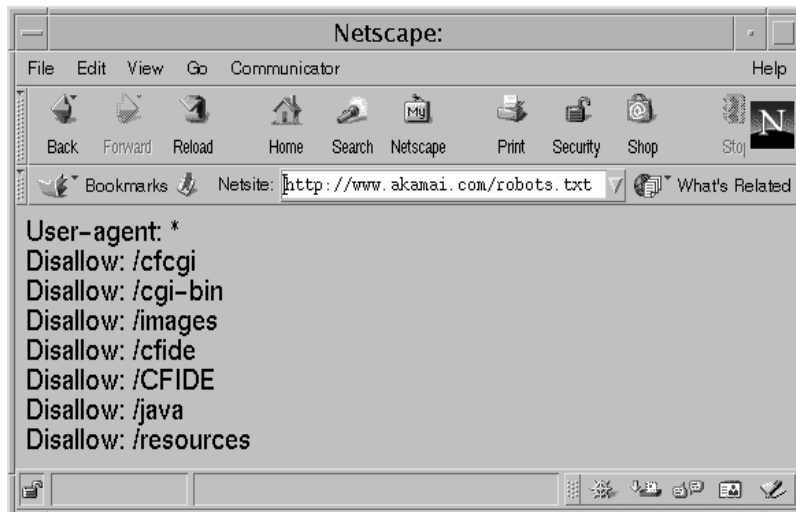
Robot Exclusion Protocol

(see *Web Robots Pages*)

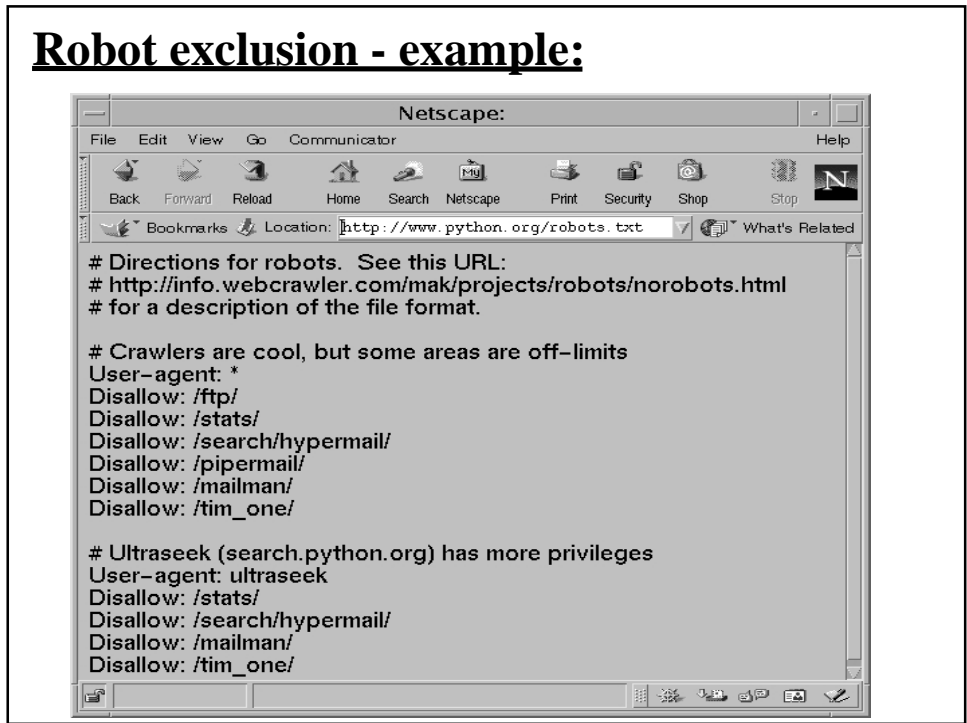


- file **robots.txt** in root directory
- allows webmaster to “exclude” crawlers (*crawlers do not have to obey*)
- may exclude only certain robots or certain parts of the site
 - to “protect proprietary data” (e.g., eBay case)
 - to prevent crawlers from getting lost
 - to avoid load due to crawling
 - to avoid crashes (protect CGI bin)
- if at all possible, follow robot exclusion protocol!

Robot exclusion - example:



Robot exclusion - example:



Robot META Tags

(see [Web Robots Pages](#))

- **allow page owners to restrict access to pages**
- **does not require access to root directory**
- **excludes all robots**
- **not yet supported by all crawlers**
- **“noindex” and “nofollow”**

Crawling courtesy

- **minimize load on crawled server**
- **no more than one outstanding request per site**
- **better: wait 30 seconds between accesses to site**
(this number is not fixed)
- **problems:**
 - one server may have many sites *(use domain-based load-balancing)*
 - one site may have many pages *(3 years to crawl 3-million page site)*
 - intervals between requests should depend on site
- **give contact info for large crawls (email or URL)**
- **expect to be contacted ...**

Crawling challenges

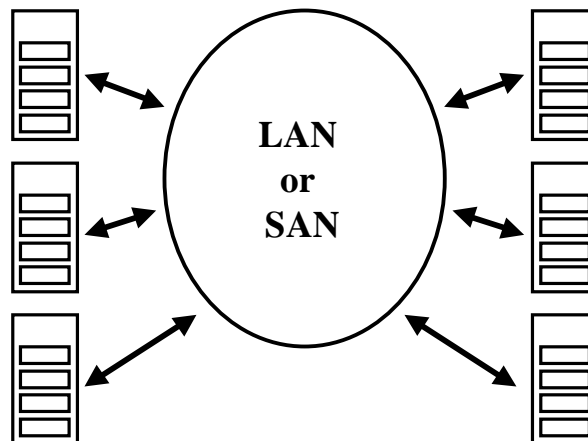
- **crawler may have to run for several weeks or months**
- **will interact with millions of web server**
- **some of them will be odd:**
 - noncompliant server responses
 - unfamiliarity with robot exclusion protocol
 - robot traps
 - CGI and unintended consequences
 - network security tools
 - weird webmasters
- **unclear legal situation**

Storage:

- average HTML page size: ~ 14KB (plus ~ 40KB images)
- 2 billion pages = 28 TB of HTML
- compression with gzip/zlib: 7-8 TB (3-4 KB per page)
- or about 3 KB text per page after stripping tags
(according to Stanford WebBase group)
- 1 KB per page if stripping and compressing
- 1-4 KB compressed index size per page
(depends on whether we store position in document)
- 2-8 TB index size for 2 billion pages
- page and index compression important

Low cost storage:

- Linux PCs connected by Ethernet or Myrinet SAN (system area network)
- several disks per node (160GB IDE for \$230)
- Stanford WebBase, Internet Archive (and here at Poly)
- parallel processing, active/intelligent disks paradigm
- separate data and index, or not?



Storage system options: *(for pages)*

- **store pages in standard DBMS**
(Oracle, DB2, mySQL)
- **use file system**
 - many pages per file *(due to file system limits and bottlenecks)*
 - done by Internet Archive
- **use specialized storage system**
 - **hash-partitioned:** Stanford WebBase, Berkeley DDS
 - **range-partitioned:** Polytechnic *(Alex Okulov 2002)*
 - **option:** use Berkeley DB or Wisconsin Shore as storage manager on nodes
- **operations: write, read, and scan range of pages**

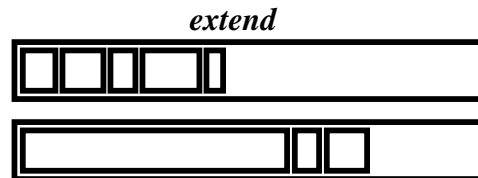
System at Poly: *(Alex Okulov 2002)*

- **Storage system supporting delta compression**
- **E.g.: Internet Archive:**
 - 10 billion pages, 100TB
 - many versions of each page:
Wayback Machine (at www.archive.org)
 - **does not currently employ delta compression**
[Kahle 2002]
- **How to build a TB storage system that**
 - employs delta compression
 - has good insertion and random read performance
 - has good streaming performance
 - is resilient to crashes

Basic Approach:

- similar pages tend to have similar URLs
- *Extends:*
 - put similar pages in same place (based on URLs)
 - use “extends” of 128/256KB to hold similar pages
 - delta-compress within an extend

- add in gzipped form
- compact, then continue
- eventually split

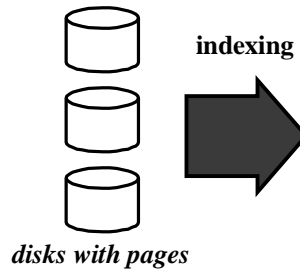


- Can use only very basic heuristics within extend
- CPU becomes bottleneck (*uncompression cost*)

Some numbers:

- hardware: 16-node cluster with fast Ethernet
- each node P4, 512 MB, 160 GB (2.56 TB total)
- data set: 140 million pages, 1.8TB
- 120-150 per sec per disk insertion (elevator)
 - ➡ one day to insert onto 4 nodes (6 hours on 16)
- currently studying compression schemes
- how much do pages change?
- page versions: 400000 random pages crawled for 2 months every night
 - significant delta compression (little change)
 - how much space needed to store daily changes?

Indexing *(see book for details)*



aardvark	3452, 11437,
.	
.	
.	
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
.	
.	
.	
zebra	602, 1189, 3209, ...

inverted index

- **how to build an index**
 - in I/O-efficient manner
 - in-place (no extra space)
 - in parallel (later)
- **closely related to I/O-efficient sorting**
- **how to compress an index** (*while building it in-place*)
- **goal: intermediate size not much larger than final size**

Basic concepts and choices:

- **lexicon: set of all “words” encountered**
millions in the case of the web, mostly non-words
- **for each word occurrence:**
store index of document where it occurs
- **also store position in document?** (*probably yes*)
 - increases space for index significantly!
 - allows efficient search for phrases
 - relative positions of words may be important for ranking
- **also store additional context?** (*in title, bold, in anchortext*)
- **stop words: common words such as “is”, “a”, “the”**
- **ignore stop words?** (*maybe better not*)
 - saves space in index
 - cannot search for “to be or not to be”
- **stemming: “runs = run = running”** (*depends on language*)

Indexing: (simplified approach)

(see Witten/Moffat/Bell for details)

- (1) scan through all documents
- (2) for every work encountered generate entry (word, doc#, pos)
- (3) sort entries by (word, doc#, pos)
- (4) now transform into final form

doc1: "Bob reads a book"

doc2: "Alice likes Bob"

doc3: "book"



(bob, 1, 1), (reads, 1, 2), (a, 1, 3)
(book,1, 4), (alice, 2, 1), (likes, 2, 2)
(bob, 2, 3), (book, 3, 1)



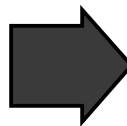
(a, 1, 3), (alice, 2, 1), (bob, 1, 1),
(bob, 2, 3), (book, 1, 4), (book, 3, 1),
(likes, 2, 2), (reads, 1, 2)



a: (1,3)
Alice: (2, 1)
Bob: (1, 1), (2, 3)
book: (1, 4), (3, 1)
likes: (2, 2)
reads: (1, 2)

Improvements

:	
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
:	
:	



:	
arm	4, 15, 10, 69, 45, ...
armada	145, 312, 332, ...
armadillo	678, 1456, 1836, ...
armani	90, 166, 116, 139, ...
:	
:	

- encode sorted runs by their gaps
 ➔ significant compression for frequent words!
- less effective if we also store position
 (adds incompressible lower order bits)
- many highly optimized schemes have been studied
 (see book)

Additional issues:

- **keep data compressed during index construction**
- **try to keep index in main memory?** (*altaVista*)
- **keep important parts in memory?** (*fancy hits in google*)
- **use database to store lists?** (*e.g., Berkeley DB*)
use BLOBs for compressed lists; rely on DB for caching
- **or use text indexes provided by databases?**

Alternative to inverted index:

- **signature files (Bloom filters): false positives**
- **bitmaps**
- **better to stick with inverted files!**

Updating index structures:

- **have only discussed bulk-building of indexes**
- **updates can be challenging**
 - **assume you are adding one new document (new page)**
 - **document consists of 500 words**
 - **requires 500 insertions into index on disk !!!!**
- **many indexers do not support updates (efficiently)**
- **solutions:**
 - **semi-dynamic solution: build separate index, and merge**
 - **buffer insertions in memory**
 - **use Zipf distribution of word occurrences**
 - **or buy lots of fast disks ...**
- **need to decide if update performance is important**

Some indexing numbers: (Long/Suel 2002)

- 140 million pages, 1.8 TB
- 7 nodes: 800Mhz P-III with 512MB and 2*80GB
- 130 GB uncompressed, 35GB compressed per disk
- build one index structure per disk
- indexing performance: 4 MB/s per node
- 9 hours per disk, 18 hours for parallel index run
- index size: 1.6 KB per page = 12% of original size (including position in document)

Boolean querying and term-based ranking:

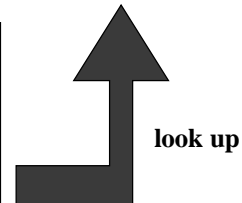
(see *Managing Gigabytes book*)

Recall Boolean queries:

(zebra AND armadillo) OR armani

➔ unions/intersections of lists

aardvark	3452, 11437,
.	.
.	.
.	.
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
.	.
.	.
.	.
zebra	602, 1189, 3209, ...



Boolean queries vs. ranking

- most web queries involve one or two common words
 - ➔ Boolean querying returns thousands of hits
- would like to rank results by ...
 - importance?
 - relevance?
 - accuracy?
- in general, arbitrary *score function*:
 - “return pages with highest score relative to query”
- use inverted index as access path for pages
 - start with (possibly expanded) Boolean query
 - only rank Boolean results
 - in fact, try to avoid computing complete Boolean results

Vector space model

- a document is a set (or bag) of words
- thus, a document corresponds to a vector in $[0,1]^k$, where k is the number of words in the lexicon
- two documents are similar if
 - their sets have a large intersection? (*inner product*)
 - their vectors go into a similar direction! (*cosine measure*)
- weigh words by inverse frequency (*rare = meaningful*)
- assume query is also a set of words (*no AND, OR*)
 - score = similarity between query and document
 - remember: most queries are only 1 or 2 words on the web
 - one approach: expand query with additional related words

Ranking continued:

- **vast amount of vector space work in IR**
(see *Witten/Moffat/Bell* and *Baeza-Yates/Ribeiro-Neto* for intro & pointers)
- **not all results directly applicable to search engines**
- **additional factors in ranking:**
 - distance between terms in text
 - titles and headings and font size
 - use of meta tags?
 - user feedback or browsing behavior?
 - link structure - later!
- **efficiency extremely important!** (*Google: 10000 queries/sec*)

Text classification:

- **given a set of documents and a set of topics, assign documents to topics**
- **classical problem in IR and machine learning**
- **chicken & egg: “how to define topics”**
- **learning approach**
 - take a small subset of documents, called *training set*
 - classify training set by hand
 - now have program learn by example
- **... imagine learning to classify documents in an alien language based on statistical observations**

Discussion:

- **Bayesian classifier:**

“the frequency of a term depends on the topic”

- assume that document on a topic has a certain likelihood of using a given term

- thus, given a document, which topic is most likely to produce its set of terms

- **many approaches**

- **software tools:** *rainbow* (CMU)
WECA (U. of Waikato)

- **hierarchical topic structure**

- **use of link structure for categorization**

III - Search Applications and Tools:

- **Types of search tools**

- **Available software systems and tools**

- **Example: Major search engine**

- **Example: Focused Data Collection and Analysis**

- **Example: Browsing/Search Assistants**

- **Example: Site and Enterprise Search**

- **Demonstration of search tools**

- **Using search engines**

- **Search engine optimization and manipulation**

Types of web search tools

- Major search engines
(*google, fast, altavista, inktomi, teoma, wisenut, openfind*)
- Web directories (yaho, *open directory project*)
- Specialized search engines (cora, *citeseer, achoo, findlaw*)
- Local search engines (for one site)
- Meta search engines (dogpile, *mamma, search.com*)
- Personal search assistants (*alexa, google toolbar*)
- Comparison shopping agents (*mysimon, pricewatch*)
- Image search (*ditto, visoo*)
- Natural language questions (*askjeeves?*)
- Database search (*completeplanet, brightplanet*)

Types of web search tools (another view)

massive data vs. *more moderate amounts of data*
crawl-based vs. *meta techniques*
server-based vs. *client-based* (vs. *proxy-based*)
general vs. *specialized*
single-node vs. *parallel cluster* (vs. *highly distributed*)

- **google, inktomi:** *massive data, crawl, server, general, cluster*
- **citeseer:** (*massive data*), *crawl, server, specialized, (cluster)*
- **findlaw:** *moderate data, meta, proxy/server, specialized, single-node*
- **dogpile:** *moderate data, meta, proxy/server, general, cluster*
- **letizia (MIT):** *moderate data, crawl, client, general, single-node*
- **alexa/google toolbar:** *moderate data, meta, client, general, single-node*

Useful software tools

- **major search engines based on proprietary software**
 - must scale to very large data and large clusters
 - must be very efficient
 - low cost hardware, no expensive Oracle licenses
- **site search based on standard software**
 - appliance; set up via browser (e.g., google)
 - or software with limited APIs (e.g., altaVista, fast, ...)
 - or part of web server or application server
 - or offered as remote services (fast, atomz, ...)
- **enterprise search: different ballgame**
 - security/confidentiality issues
 - data can be extremely large
 - established vendors (e.g., Verity)
- **other cases: what to do?**

Useful software tools (ctd.)

- **database text extensions (Oracle, IBM, Informix, Taxis)**
 - e.g., Oracle9i text, extensions (formerly interMedia text)
 - offer inverted indexes, querying, crawling
 - support for IR operations such as categorization, clustering
 - support for languages and file types
 - integrated with database, ACID properties
 - simple search almost out of the box
- **when to use DBMS?**
 - transaction properties needed?
 - which features are needed? which ones are really provided?
 - efficiency (DBMS overhead, index updates?)
 - how far do you need to scale? (Oracle: scaling == \$\$\$)
- **DBMS / IR gap:**
 - getting smaller (DBMS are getting there)
 - but be aware of differences: not everything is a relation, and a standard DB index is not a good text index

Useful software tools (ctd.)

- ***lucene*** (part of Apache Jakarta)
 - free search software in Java: crawling, indexing, querying
 - inverted index with efficient updates
 - documents are stored outside
 - good free foundation, not as feature-rich as Oracle text
- ***IBM intelligent miner for text***
 - similar features as lucene, plus extra IR operations
 - categorization, clustering, languages, feature extraction
 - uses DB2 to store documents, but not fully integrated
(different from DB2 text extensions)
- **MS indexserver/siteserver**
 - provides indexing and crawling on NT
- **many other tools ...** (*see here for list*)

Conclusions: software tools

- **evolving market: vendors from several directions moving in**
- **massive-data engines: proprietary code, made from scratch**
- **site search: out of the box**
- **many other applications in between should try to utilize existing tools, but cannot expect complete solutions**

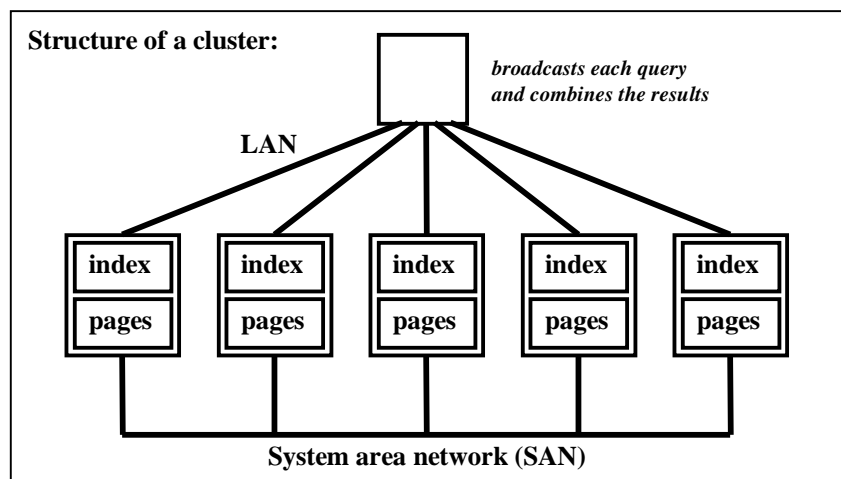
Questions:

- **how much do you need to scale, and how much can you pay?**
- **transaction properties needed?**
- **mixture of text and relational data?**
- **support for different languages and data types needed?**
- **advanced IR and data mining, or simple queries?**

Example 1: major search engines (*google, inktomi*)

- 2 billion pages, up to 10000 queries per second (google)
- very large, scalable clusters of rackmounted servers
- google: linux with proprietary extensions
- inktomi: Solaris on Sun, plus Intel-based
- large-scale parallel processing
- parallel crawler for data acquisition: 1000 pages per second
- pages and index are partitioned over cluster in redundant way
- inktomi: horizontal partitioning
 - each node contains subset of pages, and an inverted index for this subset only
 - simpler to manage, but not completely scalable

Example 1: major search engine (*ctd.*)



- several replicated clusters, with load-balancer in front
- or several leader nodes and more complicated replication
- can use SAN to maintain replication, or for query processing in vertical index partitioning

Example 1: major search engine (ctd.)

- **great paper by Eric Brewer (Inktomi):**
 - “*Lessons from Giant Scale Services*” (*paper, video of talk*)
- **index updates: no, but crawl some subset daily and index separately, then combine into query results**
- **lots of painful details omitted**
 - how to use links for ranking
 - how to use advanced IR techniques
 - languages, filetypes
 - crawling is an art
- **getting all the details right takes years**

Example 2: focused data collection & analysis

- **NEC Citeseer:** *specialized engine for Computer Science research papers*
- **crawls the web for CS papers and indexes them**
- **analyzes citations between papers and allows browsing via links**
- **challenges:**
 - **focused crawling:** learn where to find papers without crawling entire web
 - recognizing CS papers (and convert from PS and PDF to text)
 - identify references between papers
- **Whizbang job database:** *collect job announcements on company web sites*
- **also based on focused crawling**
- **needs to categorize job announcements by type, locations, etc.**

Example 2: focused data collection & analysis *(ctd.)*

Other applications:

- trademark and copyright enforcement
 - track down mp3 and video files
 - track down images with logos (*Cobion*)
- comparison shopping and auction bots
- competitive intelligence
- national security: monitoring extremist websites

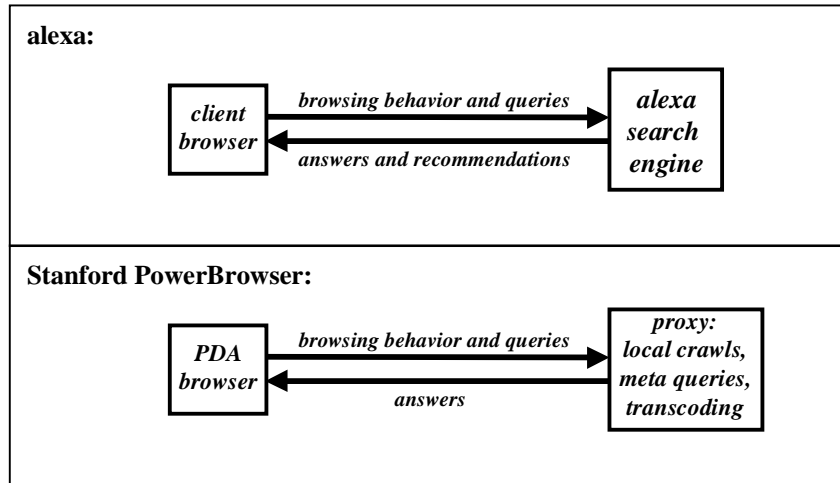
- applications may involve significant amounts of data
- a lot of proprietary code
- a lot of activities in the shadows ...

Example 3: browsing/search assistants

- tied into browser (plugin, or browser API)
- can suggest “related pages”
- search by “highlighting text” ➡ can use context
- may exploit individual browsing behavior
- may collect and aggregate browsing information
privacy issues (alexas case)
- architectures:
 - on top of crawler-based search engine (*alexas, google*), or
 - based on meta search (*MIT Powerscout*)
 - based on limited crawls by client or proxy

(MIT Letizia, Stanford Powerbrowser)

Example 3: browsing/search assistants (ctd.)



- Letizia (MIT): high-bandwidth client without proxy

Example 4: site and enterprise search

- **site search: out-of-the-box software or appliance**
- **simple interface:**
 - what should be crawled (domain, start pages)
 - when/how often to crawl
 - can also get data from databases and mail servers
 - can customize query results
- **often scaled-down versions of search engines, with pay per amount of data** (*fast, altaVista, google, inktomi*)
- **limited customization, usually no powerful API**
- **alternative: remote services**
 - service crawls site
 - results returned as web service

Example 4: site and enterprise search

- **enterprise search: search all company data**
- **challenges:**
 - many data sources and data sites
 - many locations around the globe
 - not all data should be accessible to everybody
 - data types: mix of relational and text data
- **huge market, attracting many players**
(DBMS, search companies, document management & warehousing)
- **single company may have more data than entire web**
- **data sources (e.g., databases) cannot be completely crawled**
(querying remote data sources using whatever interfaces they provide)
- **ranking for site and enterprise search is different**
(*not clear Pagerank works here*)

Using search engines: (some features)

- many engines allow limited Boolean operators
- many engines assume AND between terms as default
- terms in titles, bold face, or anchor text score higher
- distance between terms matters
- various advanced operations:
 - link query: *which pages link to page X?* (*google, altaVista*)
 - site query: return results only from site Y (*altaVista, altaVista*)
 - dates for age of page
- link and site queries wonderful for research, but limited “supply”
- also: internet archive “wayback machine”
- google Pagerank issues
 - google toolbar shows Pagerank
 - stopwords also show something about Pagerank (*google*)
 - anchor text may influence scores of pages that are linked
(*Bush google story*)

Search engine optimization and manipulation:

- large industry of consultants and software tools that promise to improve ranking of sites on certain queries
- example: books, CDs, computers
- important difference between web search and traditional IR
- *keyword optimization*: finding good combinations of keywords
- *keyword spamming*: adding lots of unrelated keywords
- *spoofing*: giving crawler a different page than surfer
- *link optimization*:
 - ask other sites to link to your site
 - or create your own network of fake sites
 - existence of large cliques and link farms
- one reason search engines do not publish their ranking schemes
(and why most engines only return top few hundred results)

Search engine optimization and manipulation: (ctd.)

- search engines are fighting back
 - punish for keyword spamming
 - may blacklist or not even crawl some sites
 - detection of “nepotistic” links between unrelated pages
(data cleaning step before running pagerank)
 - no winner expected soon (compare to security)
- optimization consultants: “pay us and get ranked much higher”
- search engines: “just build a good site and leave ranking to us”
- reality: a good site is important for ranking over time, but you have to be careful about links and keywords and avoid mistakes
- more info: (*searchenginewatch*, *searchengineworld*, *uneveninternet*)

IV - Advanced Techniques:

- **High-performance crawling**
- **Recrawling and focused crawling**
- **Link-based ranking (Pagerank, HITS)**
- **Vector-space models and term-based ranking**
- **Integration of link- and term-based methods**
- **Meta search engines**
- **Parallel search engines and scaling**
- **Structural analysis of the web graph**
- **Document clustering and duplicate detection**

High-performance crawling: *100 million pages or more*

- **experience from research project at Poly**
- **need high-performance crawler: >100 pages/sec**
- **robust operation over several weeks**
 - **crawler will crash**
 - **system will have to be modified**
- **controlled operation**
 - **other users on campus**
 - **remember the 30-second rule**
 - **things will happen**

Networking performance

- **server/DNS/network latency** (*0.2-1 seconds minimum*)
- **must open hundreds of connections simultaneously**
- **150 pages = 2 MB/s = 40% of max. T-3 capacity**
- **DNS can become bottleneck**
- **10-20% additional accesses for robots.txt**
- **data must be streamed to disk**
- **OS limits and overheads: networking, files**

Crawler Architectures

- **Google crawler** “*backRub*” (*see WWW’98 paper*)
 - **python downloaders on several machines**
 - **up to 100 pages per second**
- **Mercator (DEC/Altavista)** (*see Mercator paper*)
 - **2 GB machine with RAID**
 - **implemented in Java** (many performance issues)
 - **up to 200 pages/sec**
 - **detailed discussion of data structure size**
 - **optimized DNS caching**
- **PolyBot** (Shkapenyuk/Suel ICDE 2002)

Polybot crawler: (ICDE 2002)

- distributed implementation in C++
- manager handles several request streams with priorities
- manager handles DNS, exclusion, and frontier
- 300 pages/sec (and more)

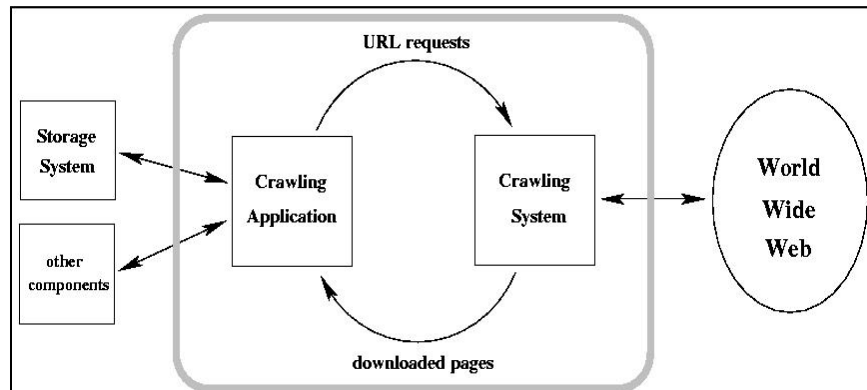


Figure 1: Basic two components of the crawler

Crawling Strategy and Download Rate:

- crawling strategy: “What page to download next?”
- download rate: “How many pages per second?”
- different scenarios require different strategies
- lots of recent work on crawling strategy
- little published work on optimizing download rate
(main exception: Mercator from DEC/Compaq/HP?)
- somewhat separate issues
- building a slow crawler is (fairly) easy ...

System Requirements:

- **flexibility** (different crawling strategies)
- **scalability** (high performance at low cost)
- **robustness**
(odd server content/behavior, crashes)
- **crawling etiquette and speed control**
(robot exclusion, 30 second intervals, domain level throttling, speed control for other users)
- **manageable and reconfigurable**
(interface for statistics and control, system setup)

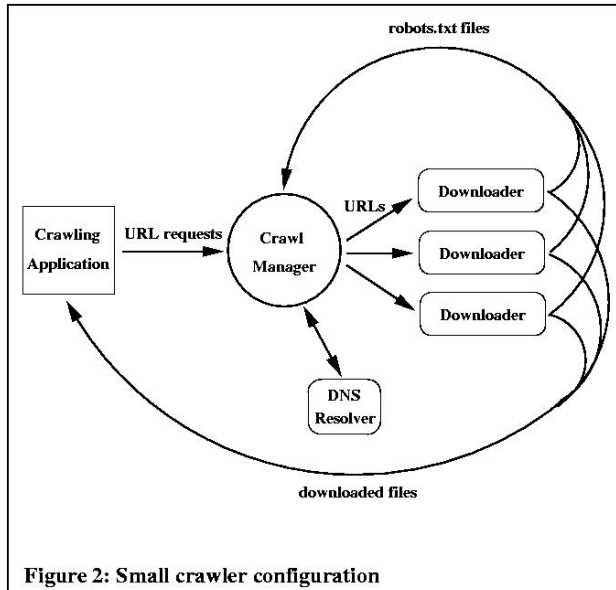
2. PolyBot System Architecture

Structure:

- **separation of crawling strategy and basic system**
- **collection of scalable distributed services**
(*DNS, downloading, scheduling, strategy*)
- **for clusters and wide-area distributed**
- **optimized per-node performance**
- **no random disk accesses** (no per-page access)

Basic Architecture, revisited:

- application issues requests to manager
- manager does DNS and robot exclusion
- manager schedules URL on downloader
- downloader gets file and puts it on disk
- application is notified of new files
- application parses new files for hyperlinks
- application sends data to storage component (indexing done later)



System components:

- **downloader:** optimized HTTP client written in Python (everything else in C++)
- **DNS resolver:** uses asynchronous DNS library
- **manager** uses Berkeley DB and STL for external and internal data structures
- **manager** does robot exclusion by generating requests to downloaders and parsing files
- **application** does parsing and handling of URLs (has this page already been downloaded?)

Scaling the system:

- small system on previous pages:
 - 3-5 workstations and 250-400 pages/sec peak
- can scale up by adding downloaders and DNS resolvers
- at 400-600 pages/sec, application becomes bottleneck
- at 8 downloaders manager becomes bottleneck
 - ➔ need to replicate application and manager
- hash-based technique (Internet Archive crawler)
 - partitions URLs and hosts among application parts
- data transfer via file system (NFS)

Scaling up:

- 20 machines
- 1500 pages/s?
- depends on crawl strategy
- hash to nodes based on site (b/c robot ex)

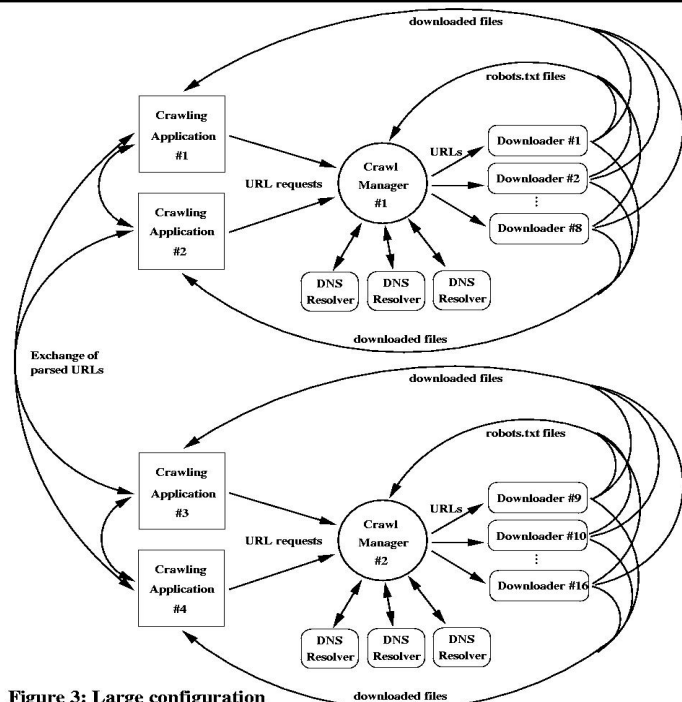


Figure 3: Large configuration

3. Data Structures and Techniques

Crawling Application

- parsing using *pcre* library
- NFS eventually bottleneck
- URL-seen problem:
 - need to check if file has been parsed or downloaded before
 - after 20 million pages, we have “seen” over 100 million URLs
 - each URL is 50 to 75 bytes on average
- Options: compress URLs in main memory, or use disk
 - prefix+huffman coding (DEC, JY01) or Bloom Filter (Archive)
 - disk access with caching (Mercator)
 - we use lazy/bulk operations on disk

- Implementation of URL-seen check:
 - while less than a few million URLs seen, keep in main memory
 - then write URLs to file in alphabetic, prefix-compressed order
 - collect new URLs in memory and periodically reform bulk check by merging new URLs into the file on disk
- When is a newly a parsed URL downloaded?
- Reordering request stream
 - want to space ot requests from same subdomain
 - needed due to load on small domains and due to security tools
 - sort URLs with hostname reversed (e.g., com.amazon.www), and then “unshuffle” the stream ➡ provable load balance

Challenges and Techniques: Manager

- large stream of incoming URL request files
- goal: schedule URLs roughly in the order that they come, while observing time-out rule (30 seconds) and maintaining high speed
- must do DNS and robot excl. “right before” download
- keep requests on disk as long as possible!
 - otherwise, structures grow too large after few million pages (performance killer)

Manager Data Structures:

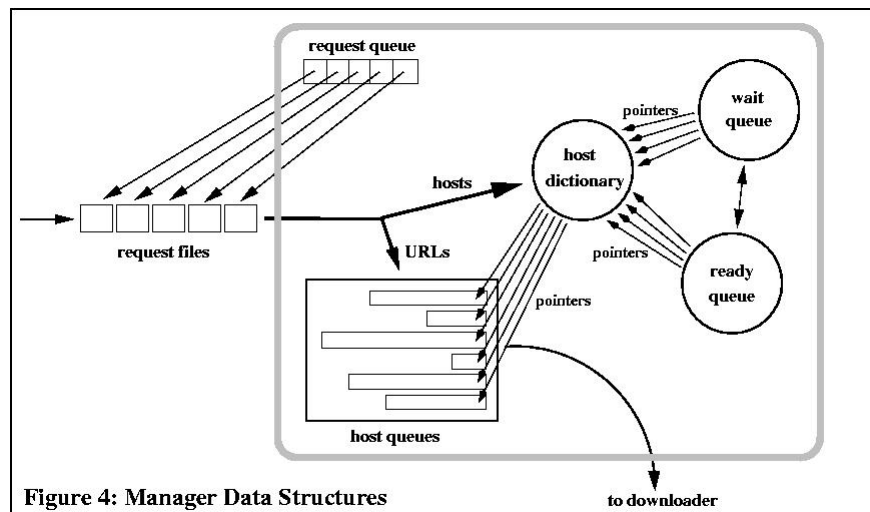


Figure 4: Manager Data Structures

- when to insert new URLs into internal structures?

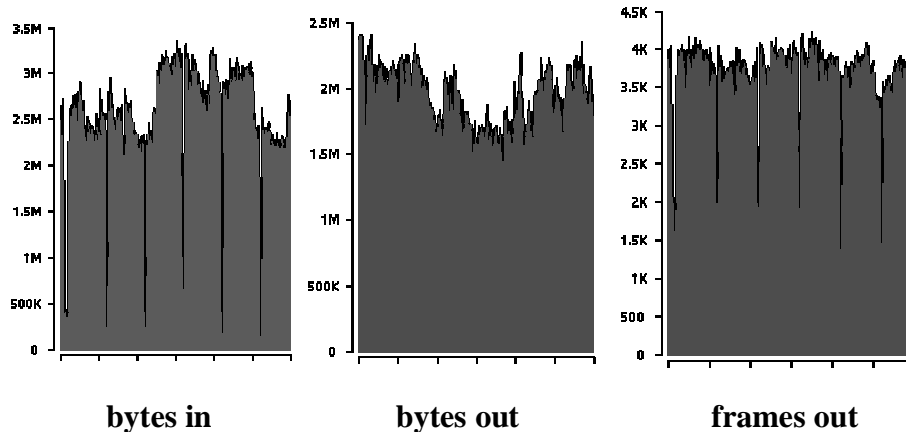
URL Loading Policy

- read new request file from disk whenever less than x hosts in ready queue
- choose $x > \text{speed} * \text{timeout}$ (e.g., 100 pages/sec * 30 sec)
- # of current host data structures is
 $x + \text{speed} * \text{timeout} + n_{\text{down}} + n_{\text{transit}}$
which is usually $< 2x$
- nice behavior for BDB caching policy
- performs reordering only when necessary!

Experimental Results

- crawl of 120 million pages over 19 days
 - 161 million HTTP request
 - 16 million robots.txt requests
 - 138 million successful non-robots requests
 - 17 million HTTP errors (401, 403, 404 etc)
 - 121 million pages retrieved
- slow during day, fast at night
- many downtimes due to attacks, crashes, revisions
- “slow tail” of requests at the end (4 days)

Experimental Results ctd.



Poly T3 connection over 24 hours of 5/28/01
(courtesy of AppliedTheory)

Experimental Results ctd.

- **sustaining performance:**
 - will find out when data structures hit disk
 - I/O-efficiency vital
- **speed control tricky**
 - vary number of connections based on feedback
 - also upper bound on connections
 - complicated interactions in system
 - not clear what we should want
- **other configuration: 140 pages/sec sustained**
on 2 Ultra10 with 60GB EIDE and 1GB/768MB
- **similar for Linux on Intel**

More Detailed Evaluation (to be done)

- **Problems**
 - cannot get commercial crawlers
 - need simulation system to find system bottlenecks
 - often not much of a tradeoff (get it right!)
- **Example: manager data structures**
 - with our loading policy, manager can feed several downloaders
 - naïve policy: disk access per page
- **parallel communication overhead**
 - low for limited number of nodes (URL exchange)
 - wide-area distributed: where do you want the data?
 - more relevant for highly distributed systems

Contributions:

- **distributed architecture based on collection of services**
 - separation of concerns
 - efficient interfaces
- **I/O efficient techniques for URL handling**
 - lazy URL -seen structure
 - manager data structures
- **scheduling policies**
 - manager scheduling and shuffling
- **resulting system limited by network and parsing performance**
- **detailed description and how-to (limited experiments)**

Other Work on Parallel Crawlers:

- **Atrax: recent distributed extension to Mercator**
 - combines several Mercators
 - URL hashing, and off-line URL check (as we do)
- **P2P crawlers (grub.org and others)**
- **Cho/Garcia-Molina (WWW 2002)**
 - study of overhead/quality tradeoff in paral. crawlers
 - difference: we scale services separately, and focus on single-node performance
 - in our experience, parallel overhead low