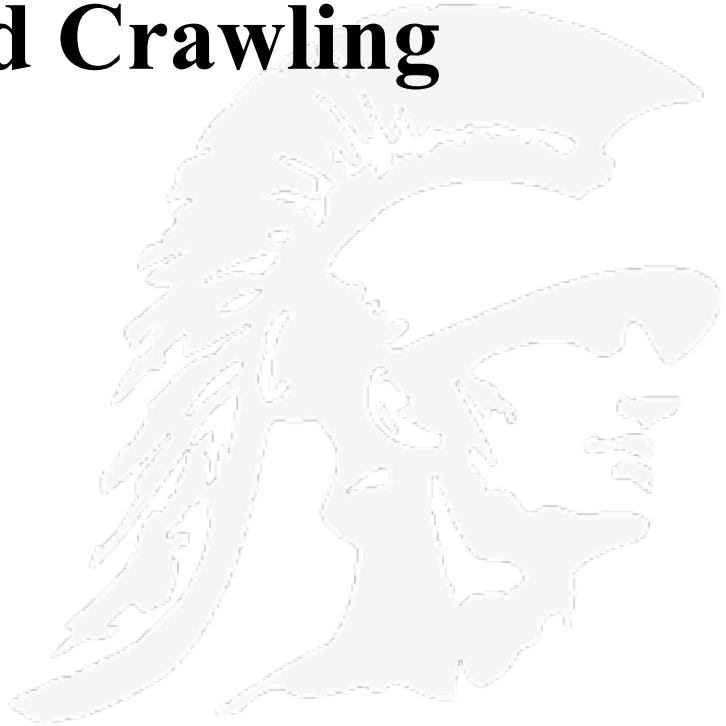
Crawlers and Crawling



There are Many Crawlers

- A web crawler is a computer program that visits web pages in an organized way
  - Sometimes called a spider or robot
- A list of web crawlers can be found at

  http://en.wikipedia.org/wiki/Web_crawler

  Google's crawler is called googlebot, see

http://support.google.com/webmasters/bin/answer.py?hl=en&answer=182072

- Yahoo's web crawler is called Yahoo! Slurp, see

http://en.wikipedia.org/wiki/Yahoo!_Slurp

- Bing uses four crawlers
  - Bingbot
  - Adidxbot
  - MSNbot
  - BingPreview
  - http://www.bing.com/webmaster/help/which-crawlers-does-bing-use-8c184ec0

Web Crawling Issues

- **How to crawl?**
  - ***Quality*: how to find the "Best" pages first**
  - ***Efficiency*: how to avoid duplication (or near duplication)**
  - ***Etiquette*: behave politely by not disturbing a website's performance**
- **How much to crawl? How much to index?**
  - ***Coverage*: What percentage of the web should be covered?**
  - ***Relative Coverage:* How much do competitors have?**
- **How often to crawl?**
  - ***Freshness*: How much has changed?**
  - **How much has <u>really</u> changed?**

Simplest Crawler Operation

- Begin with known "seed" pages
- Fetch and parse a page
  - Place the page in a database
  - Extract the URLs within the page
  - Place the extracted URLs on a queue
- Fetch each URL on the queue and repeat

Simple Picture – Complications

- Crawling the entire web isn't feasible with one machine
  - But all of the above steps can be distributed
- Even non-malicious pages pose challenges
  - Latency/bandwidth to remote servers can vary widely
  - Robots.txt stipulations can prevent web pages from being visited
    - How "deep" should you crawl a site's URL hierarchy?
  - How can one avoid site mirrors and duplicate pages
- How to handle malicious pages
  - Pages containing spam
  - Pages containing spider traps – especially dynamically generated pages
- Maintain politeness – don't hit a server too often

Robots.txt

- There is a protocol that defines the limitations for a web crawler as it visits a website; its definition is here
  - http://www.robotstxt.org/orig.html
- The website announces its request on what can(not) be crawled by placing a robots.txt file in the root directory
  - e.g. see

  http://www.ticketmaster.com/robots.txt

Robots.txt Example

- No robot visiting this domain should visit any URL starting with "/yoursite/temp/":

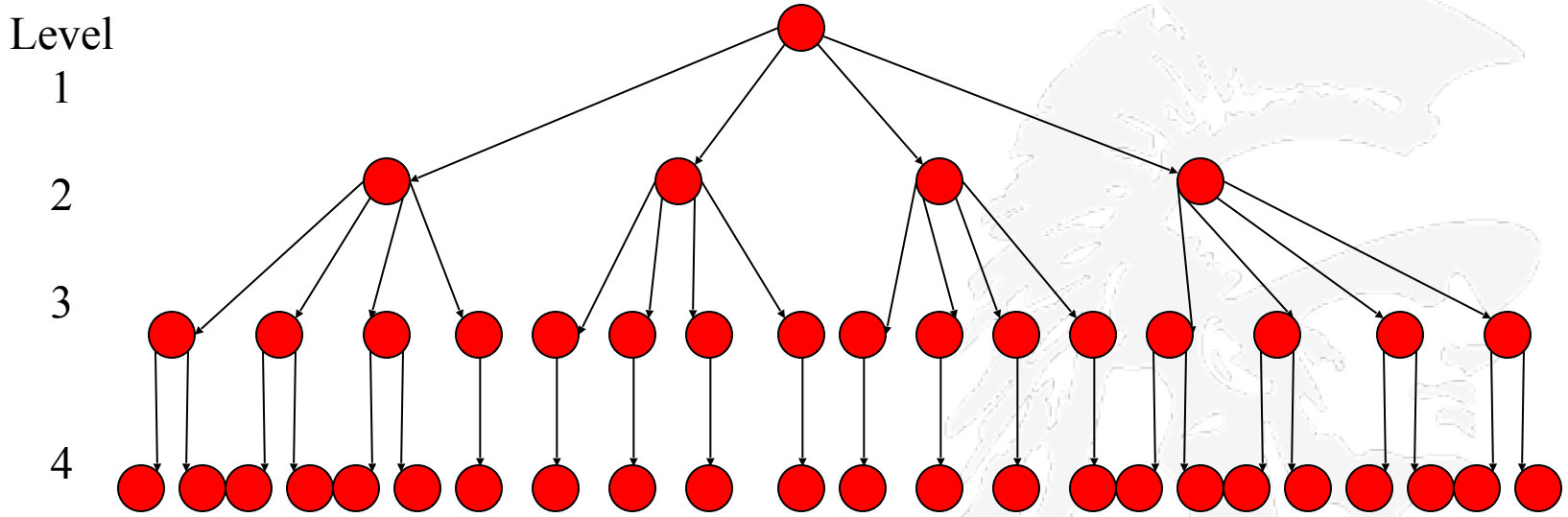
```
User-agent: *
Disallow: /yoursite/temp/
```

- Directives are case sensitive
- Additional symbols allowed in the robots.txt directives include:
  - '*' - matches a sequence of characters
  - '$' - anchors at the end of the URL string
- Example of '*':

```
User-agent: Slurp
Allow: /public*/
Disallow: /*_print*.html
Disallow: /*?sessionid
```

Basic Search Strategies

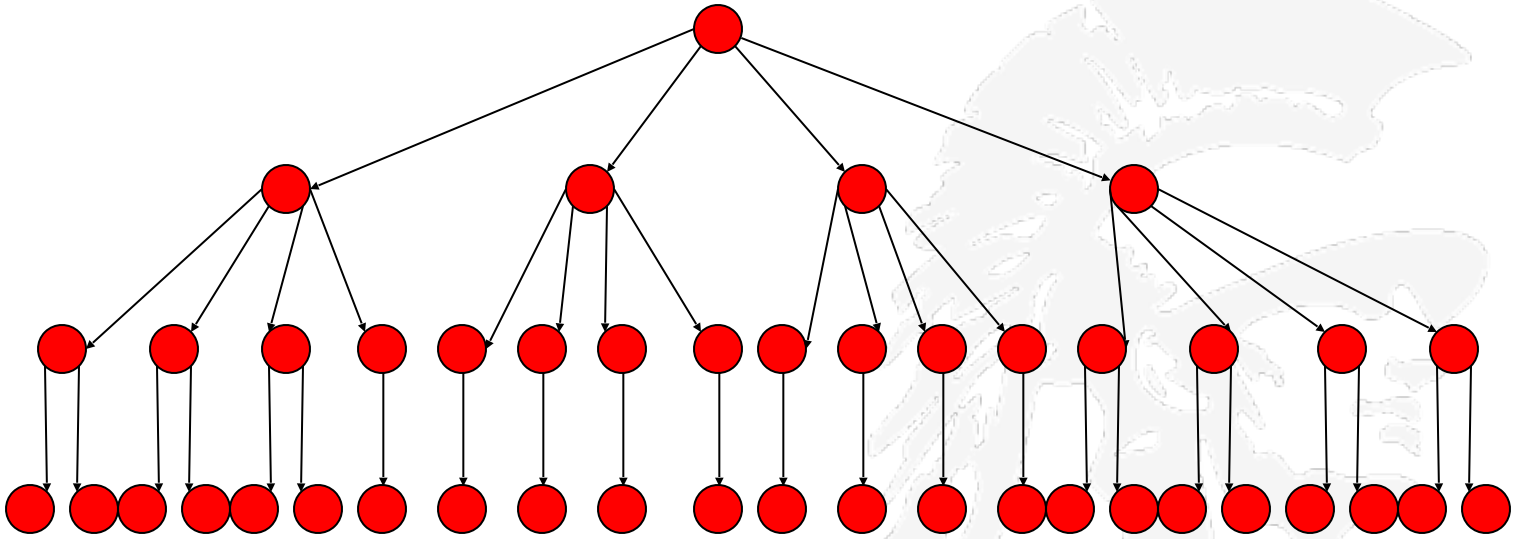
Breadth-first Search



Examine all pages at level i before examining pages at level i+1

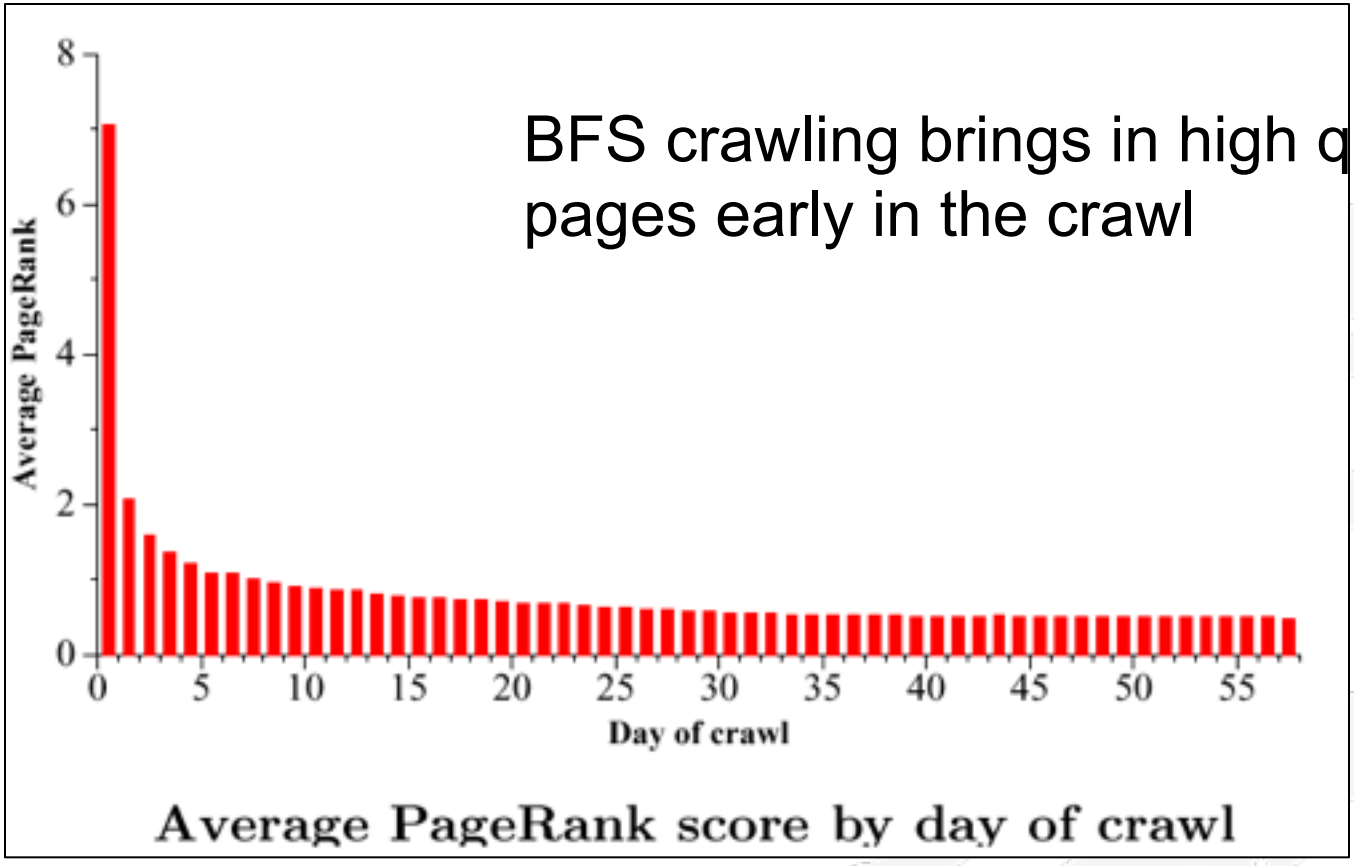
Basic Search Strategies (cont)

Depth-first Search



At each step move to a page down the tree

Web Wide Crawl (328M pages, 2000) [Najo01]



BFS crawling brings in high quality pages early in the crawl

Page Rank is an algorithm developed by Google for determining the value of a page

Crawling Algorithm – Version 2

Initialize queue (Q) with initial set of known URL's.
Loop until Q empty or page or time limit exhausted:
    Pop a URL, call it L, from the front of Q.
    If L is not an HTML page (e.g. .gif, .jpeg, ….)
        continue the loop.
    If L has already been visited, continue the loop.
    Download page, P, for L.
    If cannot download P (e.g. 404 error, robot excluded)
        continue loop.
    Index P (e.g. add to inverted index and store cached copy).
    Parse P to obtain list of new links N.
    Append N to the end of Q
End loop

Queueing Strategy

- How new links are added to the queue determines the search strategy.
- FIFO (append to end of Q) gives breadth-first search.
- LIFO (add to front of Q) gives depth-first search.
- Heuristically ordering the Q gives a "focused crawler" that directs its search towards "interesting" pages; e.g.
  - A document that has changed frequently could be moved forward
  - A document whose content appears relevant to some topic can be moved forward
- One way to re-order the URLs on the queue is to:
    - Move forward URLs whose In-degree is large
    - Move forward URLs whose PageRank is large
      - We will discuss the PageRank algorithm later

Avoiding Page Duplication

- A crawler must detect when revisiting a page that has already been crawled (web is a graph not a tree).
- Therefore, a crawler must efficiently index URLs as well as already visited pages
- To determine if a URL has already been seen,
  - Must store URLs in a standard format (discussed ahead)
  - Must develop a fast way to check if a URL has already been seen
- To determine if a new page has already been seen,
  - Must develop a fast way to determine if an *identical* page was already indexed
  - Must develop a fast way to determine if a *near-identical* page was already indexed

Link Extraction

- Must find all links in a page and extract URLs.
  - URLs occur in tags other than <a>, e.g.
  - <frame src="site-index.html">, <area, href="…">, <meta>, <link>, <script>
- Relative URL's must be completed, e.g. using current page URL or <base> tag
  - <a href="proj.html"> to http://www.myco.com/special/tools/proj.html
  - <a href="../outline/syllabus.html"> to http://www.myco.com/special/outline/syllabus.html
- Anomalies
  - Some anchors don't have links, e.g. <a name="banner">
  - <a href=http://www.mysite.com/search?x=arg1&y=arg2> invoke programs that produce dynamic pages – these links are avoided

Representing URLs

- URLs are rather long, 80 bytes on the average, implying 1 billion URLs will require 80 Gigabytes
    - Recently Google reported finding 1 trillion unique URLs, which by the above would require 80 terabytes to store

1. One Proposed Method: To determine if a new URL has already been seen
   – First hash on host/domain name, then
   – Use a trie data structure to determine if the path/resource is the same as one in the URL database
2. Another Proposed Method: URLs are sorted lexicographically and then stored as a delta-encoded text file
    - Each entry is stored as the difference (delta) between the current and previous URL; this substantially reduces storage
    - However, restoring the actual URL is slower, requiring all deltas to be applied to the initial URL
    - To improve speed, checkpointing (storing the full URL) is done periodically

- Two papers which address this problem are:
  – *The Connectivity Server: fast access to linkage information on the web*, Bharat et al
  – `http://cui.unige.ch/tcs/cours/algoweb/2002/articles/art_keller_vincent.pdf`
  – *URL Forwarding and Compression in Adaptive Web Caching* by B. Michel et al
  – `http://www.cs.ucla.edu/~lixia/papers/2000URL_forward.pdf`

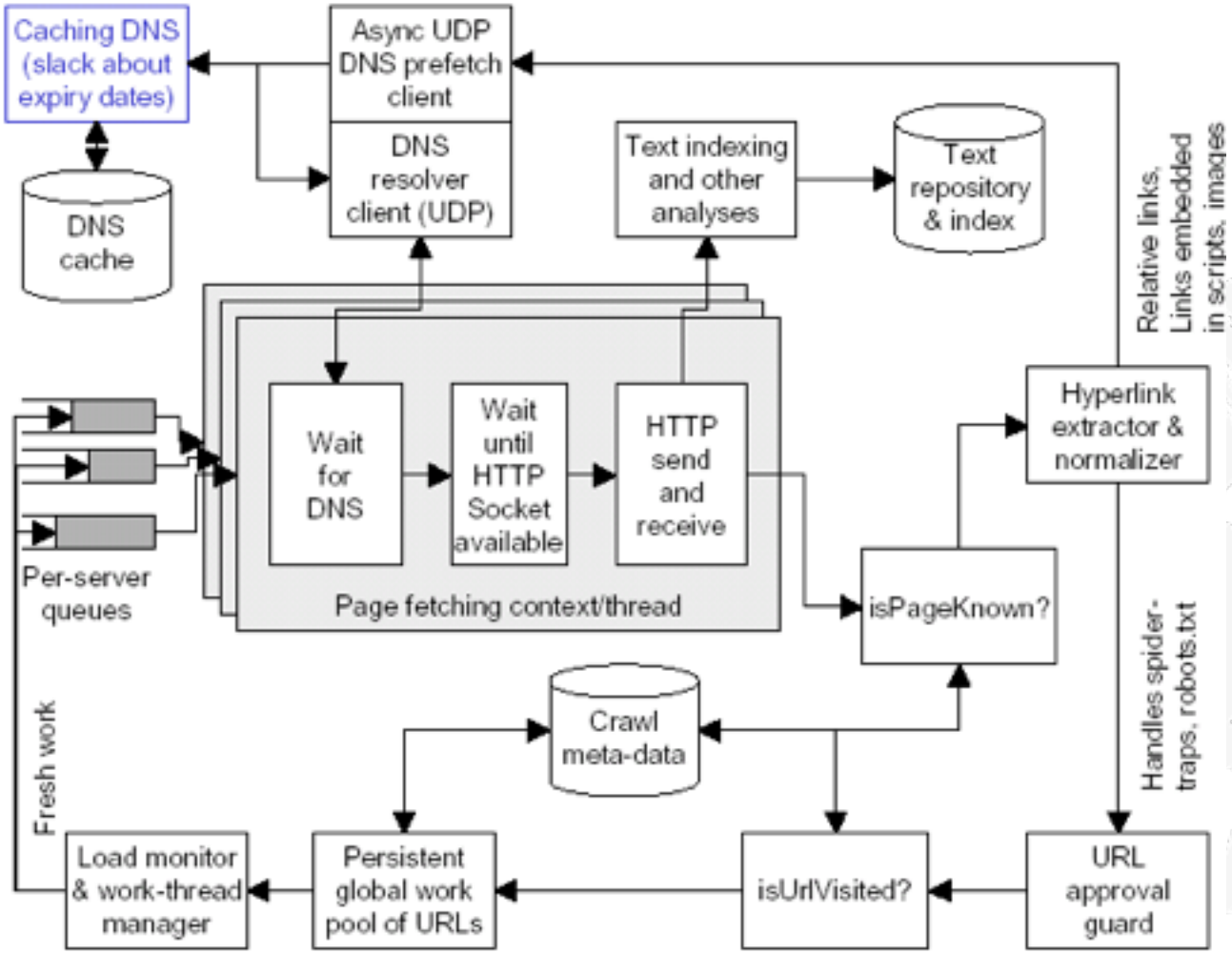
Avoiding Spider Traps

- A spider trap is when a crawler re-visits the same page over and over again
- The biggest problem with spider traps is often the amount of bandwidth and server load it puts on the site which is being crawled.
- The most well-known spider trap is the one created by the use of Session ID's
- A Session ID is often used to keep track of visitors, and some sites puts a unique ID in the URL:
  - An example is www.webmasterworld.com/page.php?id=264684413484654
    (Note this URL doesn't exist).
    Each user gets a unique ID and it's often requested from each page.
    The problem here is when Googlebot comes to the page, it spiders the page and then leaves, it comes back to another page and it finds a link to the same page but since it has been given a different session id now, the link shows up as another URL.
- One way to avoid such traps is for the crawler to be careful when the querystring "ID=" is present in the URL
- Most search engines have very advanced duplicate filters which removes the duplicates and selects one URL
- Another technique is to monitor the length of the URL, and stop if the length gets "too long"

Handling Spam

- The **first generation** of spam consisted of pages with a high number of repeated terms, so as to score high on search engines that ranked by word frequency
  - Words were typically rendered in the same color as the background, so as to not be visible, but still count
- The **second generation** of spam used a technique called *cloaking*;
  - When the web server detects a request from a crawler, it returns a different page than the page it returns from a user request
  - The page is mistakenly indexed
- A **third generation**, called a doorway page, contains text and metadata chosen to rank highly on certain search keywords, but when a browser requests the doorway page it instead gets a more "commercial" page
- Cloaking and doorway pages are not permitted according to Google's webmaster suggestions
  - See http://support.google.com/webmasters/bin/answer.py?hl=en&answer=66355
  - (show video)

The Mercator Web Crawler



The diagram points out all of the key elements of a crawler;
Notice
1.The DNS caching server
2. Use of UDP for DNS
3. Load and thread monitor
4. Parallel threads waiting for a page to download

Measuring and Tuning a Crawler

- Measuring and tuning a crawler for peak performance eventually reduces to
  - Improving parsing speed
  - Improving network bandwidth speed
  - Improving fault tolerance
- More Issues (which are discussed ahead)
  - Refresh Strategies: how often is the process re-started
  - Detecting duplicate pages
  - Detecting mirror sites
  - Speeding up DNS lookup
  - URL normalization
  - Handling malformed HTML

DNS caching, pre-fetching and resolution

1. DNS caching: build a caching server that retains IP-domain name mappings previously discovered
2. Pre-fetching client
   - once a page is parsed, immediately make DNS resolution requests to the caching server; and if unresolved, use UDP (User Datagram Protocol) to resolve from the DNS server
3. Customize the crawler so it allows issuing of many resolution requests simultaneously; there should be many DNS resolvers

Multi-Threaded Crawling

- One bottleneck is network delay in downloading individual pages.
- It is best to have multiple threads running in parallel each requesting a page from a different host.
- Distribute URL's to threads to guarantee equitable distribution of requests across different hosts to maximize through-put and avoid overloading any single server.
- Early Google spider had multiple coordinated crawlers with about 300 threads each, together able to download over 100 pages per second.

Distributed Crawling Approaches

- Once the crawler program itself has been optimized, the next issue to decide is how many crawlers will be running at any time
- Scenario 1: A *centralized crawler* controls a set of parallel crawlers all running on a LAN
- Scenario 2: A *distributed set of crawlers* running on widely distributed machines, without cross communication
- A *parallel crawler* consists of multiple crawling processes communicating via local network (sometimes called an intra-site parallel crawler), e.g. see
  - *Parallel Crawlers*, Cho and Garcia-Molina, WWW2002, May, 2002, available at
  - http://www2002.org/CDROM/refereed/108/index.html

Distributed Model

- Crawlers are running in diverse geographic locations
  - Organize crawlers by country, by region, by available bandwidth
  - Crawlers periodically update a master index
  - Incremental update is generally "cheap"
    - Why? Because compression, differential update, etc. can be used to limit communication

Issues and Benefits of Distributed Crawling

- **Benefits:**
  - **scalability: for large-scale web-crawls**
  - **costs: use of cheaper machines**
  - **network-load dispersion and reduction: by dividing the web into regions and crawling only the nearest pages**
- **Issues:**
  - **overlap: minimization of multiple downloaded pages**
  - **quality: depends on the crawl strategy**
  - **communication bandwidth: minimization**

Coordination of Distributed Crawling

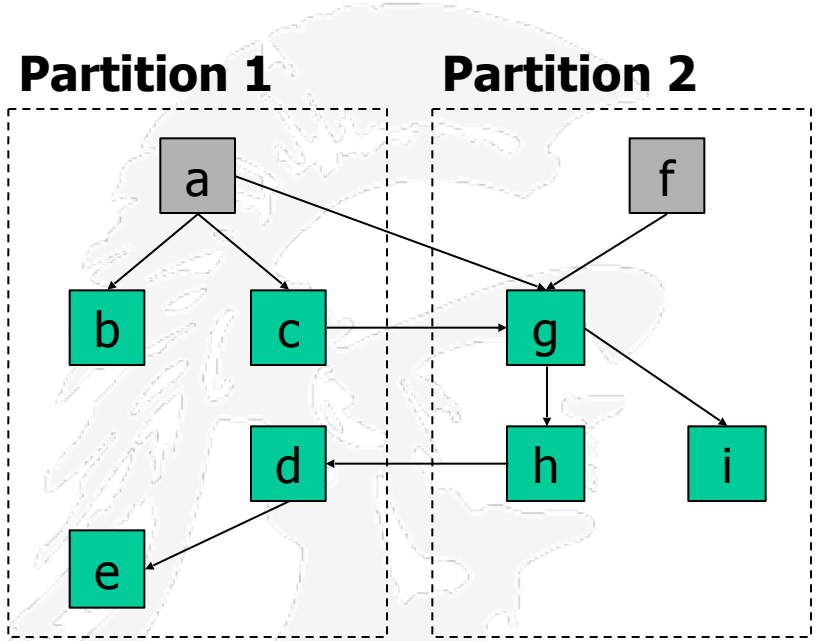
– Three strategies

1. Independent: no communication overhead
   - ► no coordination, every process follows its extracted links
2. Dynamic assignment:
   - ► a central coordinator dynamically divides the web into small partitions and assigns each partition to a process
3. Static assignment: no revisiting.
   - ► Web is partitioned and assigned without a central coordinator before the crawl starts

Static Assignment

Links from one partition to another (inter-partition links) can be handled in one of three ways:

1. Firewall mode:
   a process does not follow any inter-partition link
2. Cross-over mode:
   a process also follows inter-partition links and discovers also more pages in its partition
3. Exchange mode:
   processes exchange inter-partition URLs; this mode requires communication



Classification of Parallel Crawlers

- If exchange mode is used, communication can be limited by:
  - Batch communication: every process collects some URLs and sends them in a batch
  - Replication: the *k* most popular URLs are replicated at each process and are not exchanged (previous crawl or on the fly)
- Some ways to partition the Web:
  - URL-hash based: this yields many inter-partition links
  - Site-hash based: reduces the inter partition links
  - Hierarchical: by TLD, e.g. .com domain, .net domain …
- General Conclusions of Cho and Garcia-Molina
  - Firewall crawlers attain good, general coverage with low cost
  - Cross-over ensures 100% quality, but suffer from overlap
  - Replicating URLs and batch communication can reduce overhead

Freshness

- **The behavior of a Web crawler is the outcome of a combination of policies:**
  - **A *selection policy* that states which pages to download.**
  - **A *re-visit policy* that states when to check for changes to the pages.**
  - **A *politeness policy* that states how to avoid overloading websites.**
  - **A *parallelization policy* that states how to coordinate distributed web crawlers.**

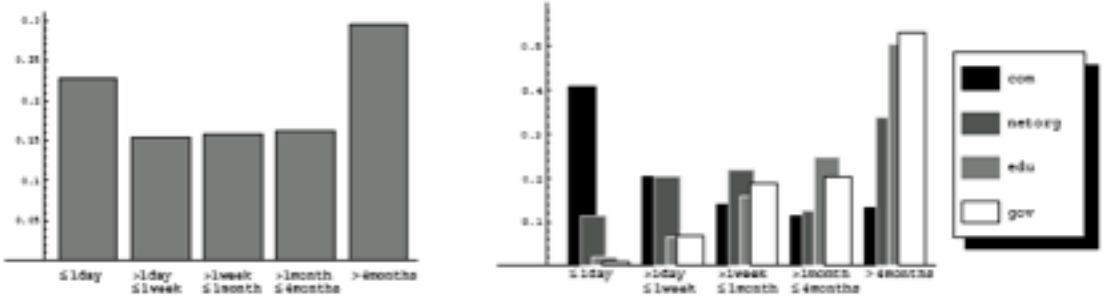
Keeping Spidered Pages Up to Date

- Web is very dynamic: many new pages, updated pages, deleted pages, etc.
- Periodically check crawled pages for updates and deletions:
  - Just look at LastModified indicator to determine if page has changed, only reload entire page if needed.
- Track how often each page is updated and preferentially return to pages which are historically more dynamic.
- Preferentially update pages that are accessed more often to optimize freshness of more popular pages.

Cho, Garcia-Molina Experiment

- Cho & Garcia-Molina made the first study of policies for crawling scheduling.
  - "The evolution of the web and implications for an incremental crawler," *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000.
- They selected 270 "popular" sites to crawl
- They computed the page rank for all pages in their study
- They limited their crawl of each site to the first 3,000 pages encountered in a breadth first search
- Their observations
  - More than 20% changed on each visit
  - More than 40% of the pages in .com change every day
  - .edu and .gov pages are

  mostly static

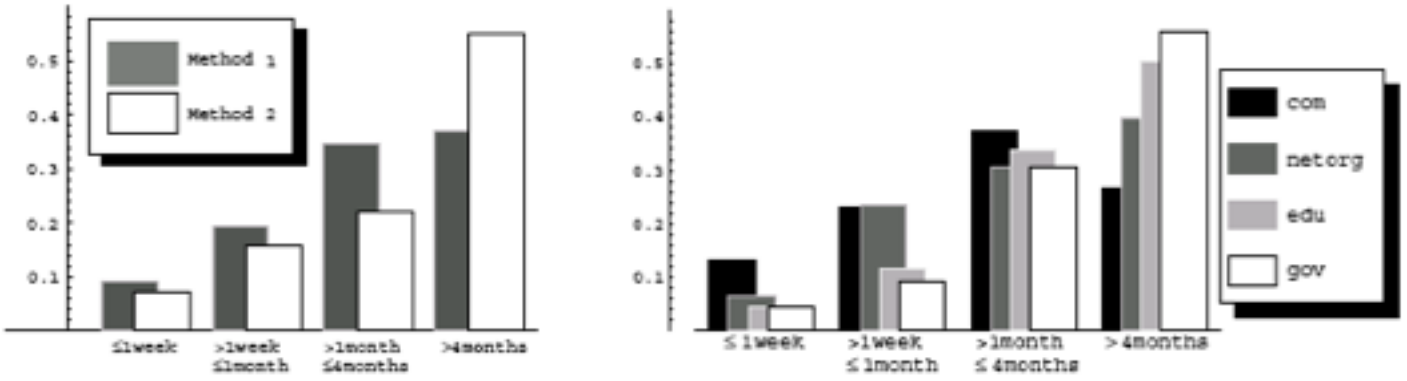


(a) Over all domains (b) For each domain

Figure 2: Fraction of pages with given average interval of change

Their Experiment (cont'd)

- Method 1 shows 19% of pages have lifespan longer than one week, but shorter than 1 month
- Method 2 shows a similar result
- More than 70% of the pages over all domains remained for more than one month
- More than 50% of pages in .edu and .gov stayed for more than 4 months



(a) Over all domains

(b) For each domain

Figure 4: Percentage of pages with given visible lifespan

Implications for a Web Crawler

- ***Batch-mode*** **crawler runs periodically updating all pages on each crawl**
- **A *steady crawler* runs continuously without pause**
- **When a crawler replaces an old version by a new page, does it do it "in-place" or "shadowing"**
  - **Shadowing implies a new set of pages are collected and stored separately and all are updated at the same time**
  - **But shadowing decreases freshness**
- **Conclusions: running multiple types of crawlers may be best**

Change Frequency vs. Optimal Re-visiting

- The horizontal axis represents the change frequency of a page, and the vertical axis shows the optimal revisit frequency for that page.
- For example, if a page in the collection changes at the frequency 1, the crawler should visit the page at the frequency *f*1.
  - Note, the *shape* of the graph is always the same regardless of the scenario.
- When a page changes at a low frequency (*lambda* < *lambda sub h*), the crawler should visit the page more often as it changes more often (*f* increases as lambda increases).
- However, when the page changes at a high frequency (lambda > *h*), the crawler should visit the page less often as it changes more often (f decreases as lambda increases)

after the threshold, if the frequency changes faster, revisit less.

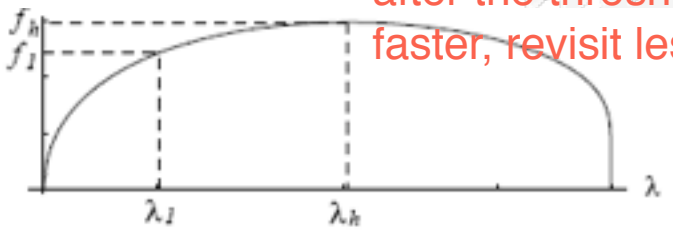


Figure 9: Change frequency of a page vs. optimal revisit frequency of the page

Cho and Garcia-Molina, 2000

- Two simple re-visiting policies
  - Uniform policy: This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.
  - Proportional policy: This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.
- Cho and Garcia-Molina proved the surprising result that, in terms of average freshness, the uniform policy outperforms the proportional policy in both a simulated Web and a real Web crawl.
- The explanation for this result comes from the fact that, when a page changes too often, the crawler will waste time by trying to re-crawl it too fast and still will not be able to keep its copy of the page fresh.
- To improve freshness, we should penalize the elements that change too often

Help the Search Engine Crawler Creating a SiteMap

- A sitemap is a list of pages of a web site accessible to crawlers
- This helps search engine crawlers find pages on the site
- XML is used as the standard for representing sitemaps
- Here is an example of an XML sitemap for a three page website

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
   <loc>http://www.example.com/?id=who</loc>
   <lastmod>2009-09-22</lastmod>
   <changefreq>monthly</changefreq>
   <priority>0.8</priority> </url>
<url>
   <loc>http://www.example.com/?id=what</loc>
   <lastmod>2009-09-22</lastmod>
   <changefreq>monthly</changefreq>
   <priority>0.5</priority> </url>
<url>
<loc>http://www.example.com/?id=how</loc>
<lastmod>2009-09-22</lastmod>
<changefreq>monthly</changefreq>
<priority>0.5</priority> </url>
</urlset>
```

Google originally introduced the sitemap format; now Bing, Yahoo, and Ask also support sitemaps

See the Google, Bing, Yahoo, Ask announcement: http://www.google.com/press/pressrel/sitemapsorg.html

Google Crawlers

- Google now uses multiple crawlers
  - Googlebot
  - Googlebot News
  - Googlebot Images
  - Googlebot Video
  - Google Mobile
  - Google Mobile Smartphone
  - Google Mobile AdSense
  - Google AdsBot
  - For details see https://support.google.com/webmasters/answer/1061943?hl=en

Crawler	User-agents	HTTP(S) requests user-agent
Googlebot (Google Web search)	Googlebot	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html) or (rarely used): Googlebot/2.1 (+http://www.google.com/bot.html)
Googlebot News	Googlebot-News (Googlebot)	Googlebot-News
Googlebot Images	Googlebot-Image (Googlebot)	Googlebot-Image/1.0
Googlebot Video	Googlebot-Video (Googlebot)	Googlebot-Video/1.0
Google Mobile	Googlebot-Mobile	[various mobile device types] (compatible; Googlebot-Mobile/2.1; +http://www.google.com/bot.html)
Google Mobile Smartphone	Googlebot-Mobile	Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X) AppleWebKit/536.26 (KHTML, like Gecko) Version/6.0 Mobile/10A5376e Safari/8536.25 (compatible; Googlebot-Mobile/2.1; +http://www.google.com/bot.html)
Google Mobile AdSense	Mediapartners-Google or Mediapartners (Googlebot)	[various mobile device types] (compatible; Mediapartners-Google/2.1; +http://www.google.com/bot.html)
Google AdSense	Mediapartners-Google Mediapartners (Googlebot)	Mediapartners-Google
Google AdsBot landing page quality check	AdsBot-Google	AdsBot-Google (+http://www.google.com/adsbot.html)

Google's Googlebot

- **Begins with a list of webpage URLs generated from previous crawls**
- **Uses Sitemap data provided by webmasters**
- **Many versions of Googlebot are run on multiple machines located near the site they are indexing**
- **Advice**
  - **To prevent "File not found" in a website's error log, create an empty robots.txt file**
  - **To prevent Googlebot from following any links on a page, use "nofollow" meta tag**
  - **To prevent Googlebot from following an individual link, add "rel='nofollow'" attribute to the link**
- **Assertion:** Chrome was in fact a repackaging of the Googlebot search crawler;
- **Why:** browsers don't just render the DOM Hierarchy of HTML, they include transformations via CSS and JavaScript, and for Googlebot to extract the most meaningful features from a web page it would be necessary to have access to these transformations
- **Conclusion:** Googlebot and Chrome share a great deal of code

Evidence that Googlebot Does More than Just Finding Links

- There is increasing evidence Googlebot can execute JavaScript and parse content generated by Ajax calls
- In the snapshot to the right you see the log file entries at Thumbtack Engineering; where Googlebot has issued a POST request with a successful result

Thumbtack has published their findings

Googlebot makes POST requests via AJAX

By steve • Tue 11 October 2011

Googlebot is constantly evolving to better capture the web's content. Over the past few years we've seen Googlebot submit GET forms and execute JavaScript. But we've always taken it for granted that Googlebot would never execute a POST request, [nor would any other well-behaved web crawler][].

We were wrong about that. Recently, we started observing Googlebot making POST requests to thumbtack.com. As far as we can tell, such requests have not been openly observed before. These Apache access log excerpts show a few examples:

```
66.249.71.47 - - [04/Sep/2011:04:53:52 +0000] "POST /act/site/clienterror
HTTP/1.1" 200 36 "http://www.thumbtack.com/ma/malden/dog-walking
/dog-walking-and-pet-care-services" "Mozilla/5.0 (compatible;
Googlebot/2.1; +http://www.google.com/bot.html)"

66.249.72.198 - - [25/Sep/2011:04:27:50 +0000] "POST /act/site/clienterror
HTTP/1.1" 200 36 "http://www.thumbtack.com/ca/solana-beach/wedding-
photographers/photography-cary-pennington-photography" "Mozilla/5.0
(compatible; Googlebot/2.1; +http://www.google.com/bot.html)"

66.249.72.207 - - [04/Oct/2011:09:53:08 +0000] "POST /act/site/clienterror
HTTP/1.1" 200 36 "http://www.thumbtack.com/tx/san-antonio/painting
/residential-commercial-construction-services" "Mozilla/5.0 (compatible;
Googlebot/2.1; +http://www.google.com/bot.html)"
```

We've verified the requests are coming from real Google crawler IP addresses:

```
$ dig -x 66.249.71.47 +short crawl-66-249-71-47.googlebot.com.
$ dig crawl-66-249-71-47.googlebot.com. +short 66.249.71.47
```

More on Googlebot

- In a public statement to Forbes magazine Google admitted that Googlebot does evaluate JavaScript and CS as it crawls web pages
- Fundamentally, a browser is just software that provides an implementation of the W3C DOM Specification via a Rendering Engine, and a scripting engine to enable any additional scripting resources.

- Evidence: Googlebot is now requesting links that don't appear directly in JavaScript — links that get put together on the fly

- What would the advantages be?
  - it emulates the user experience including page load time, final markup, positions of elements;
  - it allows detection of hidden links or hidden text

Read more: http://ipullrank.com/googlebot-is-chrome/#ixzz2qbmjVGDj

http://www.forbes.com/sites/velocity/2010/06/25/google-isnt-just-reading-your-links-its-now-running-your-code/print/

Google Isn't Just Reading Your Links, It's Now Running Your Code

Taylor Buley, Contributor

It's long been observed that Google's search indexer can read JavaScript code, the lingua franca of dynamic Web applications. But for years it's been unclear whether or not the Googlebot actually understood what it was looking at or whether it was merely doing "dumb" searches for well-understood data structures like hyperlinks.

On Friday, a Google spokesperson confirmed to *Forbes* that Google does indeed go beyond mere "parsing" of JavaScript. "Google can parse and understand some JavaScript," said the spokesperson.

Rather than just read a page for links, Google's acknowledgment suggests that it might be able to interact with applications like a human would — and crack open parts of the Web that search engines like Bing might not be able to see. That would mean that Google has redefined what it means to be a search engine.

Very few JavaScript files exist in Google's results and for its part, the company has kept any JavaScript interpreting abilities pretty close to its chest. Documentation for Google's Site Search product, for example, says that it cannot index content contained in JavaScript. A primer on indexing says it "cannot process the content of some rich media files or dynamic pages."

Yet those looking closely at their server logs may notice that Google is now requesting links that don't appear directly in JavaScript — links that get put together on the fly and Google could not possible know about unless it could execute at least part of that JavaScript code.

Mark Drummond, chief executive of Wowd, a unique search engine company

Above is a portion of the Forbes article