Web Crawling

Web Crawling

- Web crawling is the process by which we gather pages from the Web, in order to index them and support a search engine.
- The objective of crawling is to quickly and efficiently gather as many useful web pages as possible, together with the link structure that interconnects them.

Web Crawler

- WebCrawler is a Web service that assists users in their Web navigation by automating the task of link traversal, creating a searchable index of the web, and fulfilling searchers' queries from the index.
- Conceptually, WebCrawler is a node in the Web graph that contains links to many sites on the net, shortening the path between users and their destinations.

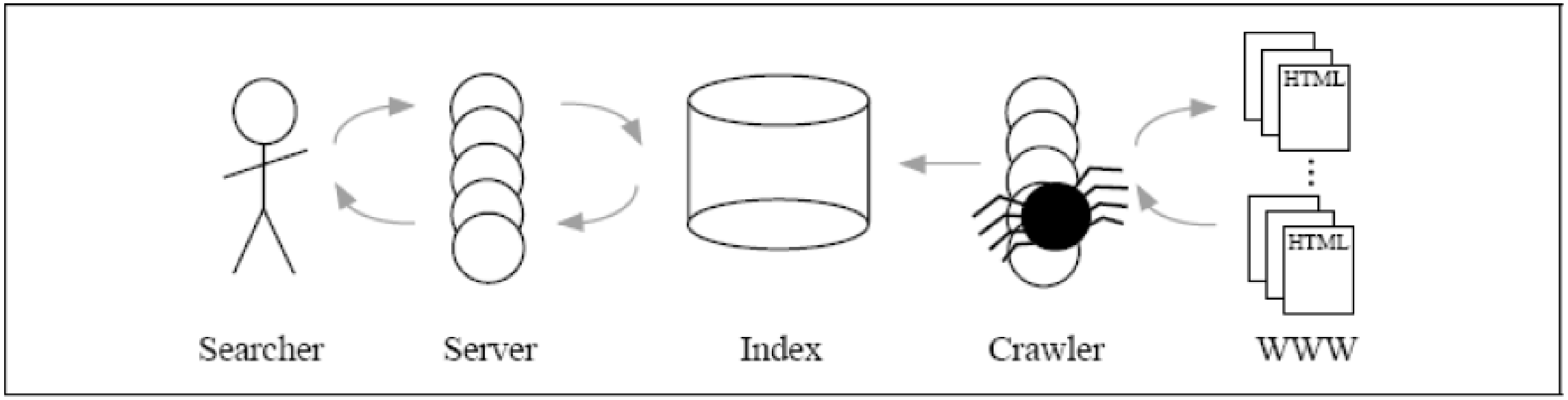
Reason-Using web Crawler

- WebCrawler saves users time when they search instead of trying to guess at a path of links from page to page.
- Often, a user will see no obvious connection between the page he is viewing and the page he seeks. For example, he may be viewing a page on one topic and desire a page on a completely different topic, one that is not linked from his current location. In such cases, by jumping to WebCrawler — either using its address or a button on the browser — the searcher can easily locate his destination page.

Functionality of web crawler

- 1. To seek out new web objects, and
- 2.To observe changes in previously-discovered web objects (web-event detection).

Web Crawler architecture



WebCrawler's overall architecture. The Crawler retrieves and processes documents from the Web, creating an index that the server uses to answer queries.

Crawler implementation

- WebCrawler’s implementation is, from the perspective of a Web user, entirely on the Web.
- The service is composed of two fundamental parts:
- **crawling**, the process of finding documents and constructing the index;
- and **serving**, the process of receiving queries from searchers and using the index to determine the relevant results. This process is illustrated schematically in previous Figure

Service of web Crawler

- **1) Crawling 2) Serving**
- **Crawling** is the means by which WebCrawler collects pages from the Web. The end result of crawling is a collection of Web pages at a central location.
- Given the continuous expansion of the Web, this crawled collection is guaranteed to be a subset of the Web and, indeed, it may be far smaller than the total size of the Web

Serving

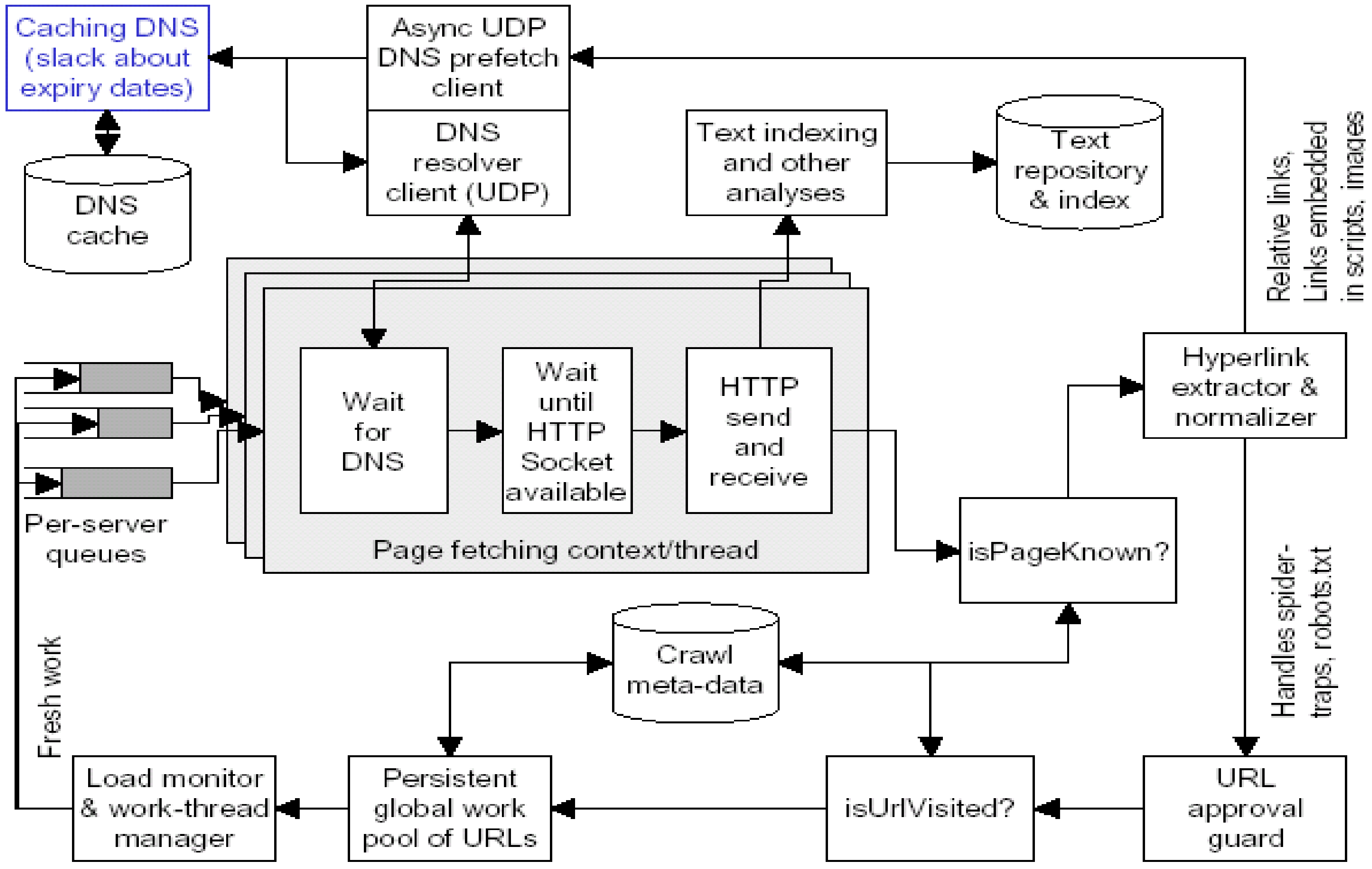
- **Serving**
- The process of handling queries from searchers and returning results in a timely way involves two main components: a high volume Hypertext Transfer Protocol (HTTP) service, and an index query process to perform the actual work of fulfilling the searcher's request.
- Requests are received from the searcher's browser via HTTP. The searcher's HTTP connection is handled by a front-end Web server whose only job is formatting search results for presentation and returning those results in HTML format to the searcher's browser.
- To handle the actual query, the front-end server sends the search request to a special query server located on its local network. The query server runs the query against the full-text index, and returns results to the front-end for formatting.

Serving cont..

- The index stored on the query server consists of two parts: a full-text index, and a metadata repository for storing page titles, URLs, and summaries.
- WebCrawler performs a search by first processing the query, looking up the component words in the full-text index, developing a list of results, ranking those results, and then joining the result list with the appropriate data from the metadata repository.

Anatomy of a crawler.

- Page fetching threads
  - Starts with DNS resolution
  - Finishes when the entire page has been fetched
- Each page
  - stored in compressed form to disk/tape
  - scanned for outlinks
- Work pool of outlinks
  - maintain network utilization without overloading it
    - Dealt with by load manager
- Continue till he crawler has collected a *sufficient* number of pages.



Caching DNS (slack about expiry dates)
DNS cache
Async UDP DNS prefetch client
DNS resolver client (UDP)
Text indexing and other analyses
Text repository & index
Relative links, Links embedded in scripts, images
Hyperlink extractor & normalizer
Wait for DNS
Wait until HTTP Socket available
HTTP send and receive
Page fetching context/thread
Per-server queues
isPageKnown?
Handles spider-traps, robots.txt
Crawl meta-data
Fresh work
Load monitor & work-thread manager
Persistent global work pool of URLs
isUrlVisited?
URL approval guard

Typical anatomy of a large-scale crawler.

DNS caching, pre-fetching and resolution

- A customized DNS component with…..

1. Custom client for address resolution
2. Caching server
3. Prefetching client

Custom client for address resolution

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

Caching server

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

Multi-threading

- logical threads
  - physical thread of control provided by the operating system (E.g.: pthreads) OR
  - concurrent processes
- fixed number of threads allocated in advance
- programming paradigm
  - create a client socket
  - connect the socket to the HTTP service on a server
  - Send the HTTP request header
  - read the socket (recv) until
    - no more characters are available
  - close the socket.
- use *<u>blocking</u>* system calls

Non-blocking sockets and event handlers

- non-blocking sockets
  - connect, send or recv call returns immediately without waiting for the network operation to complete.
  - poll the status of the network operation separately
- “select” system call
  - lets application suspend until more data can be read from or written to the socket
  - timing out after a pre-specified deadline
  - Monitor polls several sockets at the same time
- More efficient memory management
  - code that completes processing not interrupted by other completions
  - No need for locks and semaphores on the pool
  - only append complete pages to the log

Link extraction and normalization

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

Robot exclusion

- Check
  - whether the server prohibits crawling a normalized URL
  - In robots.txt file in the HTTP root directory of the server
    - species a list of path prefixes which crawlers should not attempt to fetch.
- Meant for crawlers only

Eliminating already-visited URLs

- Checking if a URL has already been fetched
  - Before adding a new URL to the work pool
  - Needs to be very quick.
  - Achieved by computing MD5 hash function on the URL
- Exploiting spatio-temporal locality of access
    - Two-level hash function.
      - most significant bits (say, 24) derived by hashing the host name plus port
      - lower order bits (say, 40) derived by hashing the path
    - concatenated bits use d as a key in a B-tree
- qualifying URLs added to frontier of the crawl.
- hash values added to B-tree.

Spider traps

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

Spider Traps: Solutions

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

Avoiding repeated expansion of links on duplicate pages

- Reduce redundancy in crawls
- Duplicate detection
  - Mirrored Web pages and sites
- Detecting exact duplicates
  - Checking against MD5 digests of stored URLs
  - Representing a relative link v (relative to aliases u1 and u2) as tuples (h(u1); v) and (h(u2); v)
- Detecting near-duplicates
  - Even a single altered character will completely change the digest !
    - E.g.: date of update/ name and email of the site administrator
  - Solution : Shingling

Thread manager

- Responsible for
  - Choosing units of work from frontier
  - Scheduling issue of network resources
  - Distribution of these requests over multiple ISPs if appropriate.
- Uses statistics from load monitor

Per-server work queues

- Denial of service (DoS) attacks
  - limit the speed or frequency of responses to any fixed client IP address
- Avoiding DOS
  - limit the number of active requests to a given server IP address at any time
  - maintain a queue of requests for each server
    - Use the HTTP/1.1 persistent socket capability.
  - Distribute attention relatively evenly between a large number of sites
- Access locality vs. politeness dilemma

Text repository

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

Storage of page-related information

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

Refreshing crawled pages

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

Estimating page change rates

- Brewington and Cybenko & Cho
  - Algorithms for maintaining a crawl in which most pages are fresher than a specified epoch.
- Prerequisite
  - average interval at which crawler checks for changes is smaller than the inter-modification times of a page
- Small scale intermediate crawler runs
  - to monitor fast changing sites
    - E.g.: current news, weather, etc.
  - Patched intermediate indices into master index

Crawler architecture

- Crawler architecture consists of following modules:
- 1. The URL frontier, containing URLs yet to be fetched in the current crawl (in the case of continuous crawling, a URL may have been fetched previously but is back in the frontier for re-fetching).
- 2. A *DNS resolution* module that determines the web server from which to fetch the page specified by a URL.
- 3. A fetch module that uses the http protocol to retrieve the web page at a URL.
- 4. A parsing module that extracts the text and set of links from a fetched web page.
- 5. A duplicate elimination module that determines whether an extracted link is already in the URL frontier or has recently been fetched.

1 URL Frontier

- **The URL frontier**
- The URL frontier at a node is given a URL by its crawl process (or by the host splitter of another crawl process). It maintains the URLs in the frontier and regurgitates them in some order whenever a crawler thread seeks a URL. Two important considerations govern the order in which URLs are returned by the frontier.
- First, high-quality pages that change frequently should be prioritized for frequent crawling. Thus, the priority of a page should be a function of both its change rate and its quality (using some reasonable quality estimate)

URL Frontier

- The second consideration is politeness: we must avoid repeated fetch requests to a host within a short time span.

2. DNS resolution

- **DNS resolution**
- Eachweb server (and indeed any host connected to the internet) has a unique *IP address*: a sequence of four bytes generally represented as four integers separated by dots; for instance 207.142.131.248 is the numerical IP address associated with the hostwww.wikipedia.org. Given a URL such as www.wikipedia.org

  in textual form, translating it to an IP address (in this case, 207.142.131.248) is a process known as *DNS resolution* or DNS lookup here DNS stands for *Domain Name Service*. During DNS resolution, the program that wishes to perform this translation (in our case, a component of the web crawler) contacts a *DNS server* that returns the translated IP address. (In practice the entire translation may not occur at a single DNS server; rather, the DNS server contacted initially may recursively call upon other DNS servers to complete the translation.)

Fetch & parsing

- 3. A fetch module that uses the http protocol to retrieve the web page at a URL.
- 4. A parsing module that extracts the text and set of links from a fetched web page.

5. duplicate elimination

- The Duplicate URL Eliminator module determines whether an extracted link is already in the URL list (log.lst file) or has recently been fetched.
- Finally, the URL is checked for duplicate elimination: If the URL is already in the log.lst file, we do not add it to the list file.
- This checking is being done by a dedicated duplicacy elimination module. This module is generally quiescent except that it wakes up once every few seconds to log crawl progress statistics (URLs crawled, list size, etc.), and thus eliminates the redundant URLs.

Crawling Process:

- The basic operation of any hypertext crawler (whether for the Web, an intranet
- or other hypertext document collection) is as follows.

- 1) The crawler begins with one or more URLs that constitute a *seed set*.
- 2) It picks a URL from this seed set, then fetches the web page at that URL.
- 3) The fetched page is then parsed, to extract both the text and the links from the page (each of which points to another URL).
- 4) The extracted text is fed to a text indexer .
- 5) The extracted links (URLs) are then added to a *URL frontier*, which at all times consists of URLs whose corresponding pages have yet to be fetched by the crawler.
- Initially, the URL frontier contains the seed set; as pages are fetched, the corresponding URLs are deleted from the URL frontier. The entire process may be viewed as traversing the web graph. In continuous crawling, the URL of a fetched page is added back to the frontier for fetching again in the future.

Features a crawler *must* provide

- 1) A *selection policy* that states which pages to download.
- 2) A *re-visit policy* that states when to check for changes to the pages.
- 3) A *politeness policy* that states how to avoid overloading Web sites.
- 4)A *parallelization policy* that states how to coordinate distributed Web crawlers.

Features a crawler *should* provide

- **Distributed:** The crawler should have the ability to execute in a distributed fashion across multiple machines.
- **Scalable:** The crawler architecture should permit scaling up the crawl rate by adding extra machines and bandwidth.
- **Performance and efficiency:** The crawl system should make efficient use of various system resources including processor, storage and network bandwidth.
- **Quality:** Given that a significant fraction of all web pages are of poor utility for serving user query needs, the crawler should be biased towards fetching "useful" pages first.
- **Freshness:** In many applications, the crawler should operate in continuous mode: it should obtain fresh copies of previously fetched pages. A search engine crawler, for instance, can thus ensure that the search engine's index contains a fairly current representation of each indexed web page. For such continuous crawling, a crawler should be able to crawl a page with a frequency that approximates the rate of change of that page.
- **Extensible:** Crawlers should be designed to be extensible in many ways – to cope with new data formats, new fetch protocols, and so on. This demands that the crawler architecture be modular.