A survey Paper on Web Crawler and their Implementation

*1Dr. B.V. Ramana Murthy, *2B. Mallikarjun Rao

*1Department of Computer Science and Engineering, *2Research scholar, Rayalaseema University.

Abstract— A program that automatically download pages from the World Wide Web is known as Web Crawlers or Spider programs or Bots. A crawler visits many sites to obtain data which is analyzed and mined in a location, either online or off-line. There is a need for crawlers to help applications stay as pages and connections are added, deleted, moved or modified. Crawlers can be used in applications in business intelligence. Crawlers are used by organizations to collect information about their competitors and potential collaborators. Crawlers monitor Web sites and pages of interest, so that a user or community can be notified when new information appears in certain places. The Crawlers can also be used for malicious applications, for example, to harvest email addresses to be used by spammers or collect personal information to be used in phishing and other identity theft attacks.

Keywords— Crawlers, Frontiers, Text, Focused, Topical, FIFO, Algorithms.

I. INTRODUCTION (HEADING 1)

A program that automatically download pages from the WWW is called as Spider [5], Bots or Crawlers [1]. As the data on the internet is scattered among many web pages that are served by web servers, users who browse the internet will follow connections to access information moving from one page to the next page. A crawler visits many sites to obtain data which is analyzed and mined in a location, either online or off-line. Once all the pages are fetched and saved in a repository, we are done. However, as the Web is a dynamic entity evolving at rapid rates. Hence there is a need for crawlers to help applications stay as pages and connections are added, deleted, moved or modified.

The organization of the paper is as follows: The section 1 introduces the Web Crawler concepts. The Section 2 introduces the basic tasks of Web Crawlers and their process. In Section 3 the Crawlers are described in detail. The Section 4 describes the algorithms used in web crawlers and the applications are described briefly in Section 5. Web Crawler process The process starts from a set of resource locators and then uses the connection within them to fetch other pages. The links in these pages are, in turn, extracted and the pages are visited. The process repeats until a sufficient number of pages are visited or some other objective is achieved. The basic crawler is described by fig 1, which fetches one page at a time.

A data structure of unvisited URLs are maintained by a program called the frontier. The data structure is
initialized with seed URLs which may be provided by
the user or program. The program iterates and picks
the next URL from the frontier, fetches the page
corresponding to the URL through agreed protocols,
parses the retrieved pages to extract their URLs, adds
newly found URLs to the frontier, and stores the page
in a local repository. This process is terminated when
a certain number of pages have been crawled. The
program may also be forced to stop if the frontier
becomes empty, which happens rarely due to the
large number of links.

These programs attempt to store the frontier in the
main memory because a large frontier size is feasible.
The programmer must decide the priority of URLs
and must be discarded when the frontier is filled up.

In the FIFO crawler program the link comes from
the top of the queue and new links are added to the
tail of the queue. Once the frontier reaches its
maximum size, the crawler can add to the queue only
one unvisited URL from each new page crawled. This
requires some data structure to be maintained with
visited links. The crawl history is a time-stamped list
of links fetched by the crawler tracking its path
through the Web. A URL is entered into the history
only after the corresponding page is fetched. This
history may be used for post-crawl analysis and
evaluation.

If pages are visited in the order specified by the
priority values in the frontier, then we can say that we
have a best-first crawler. The priority queue may be a
dynamic array that is always kept sorted by URL
scores. At each step, the best URL is picked from the
head of the queue. Once the corresponding page is
fetched, the URLs extracted from it must, in turn, be
scored. They are then added to the frontier such that
the sorting order of the priority queue is maintained.
Keeping a separate hash table for look-up is an
efficient way to avoid duplicate links in the frontier.
Once the maximum URL’s are reached, only the best
URLs are kept and the frontier must be pruned after
each new set of links is added.

To fetch pages, crawler program sends an HTTP
request to the server hosting the page and reads the
response. The crawler program parses the response
headers for status codes and redirections, which are to
be detected and broken by storing URLs from a
redirection chain in a hash table and halting if the
same URL is encountered twice.

The crawler program is built to timeout
connections and prevents unnecessary waiting for
responses from slow servers. Once a page is
downloaded, the crawler parses its content and
extracts information to support the crawler’s master
application and allows to keep extracting links to be
added to the frontier. Parsing may be simple URL
extraction from hyperlinks, or analysis of the HTML
code, by which object model establishes the structure
of an HTML page as a tag tree, as shown in Fig 1.2.

HTML parsers build the tree in a depth-first manner,
as the HTML source code of a page is scanned
linearly. But a growing portion of Web pages are
written in formats other than HTML. Some web
pages are not written in HTML, and are difficult to
parse for links by the crawlers.

Crawlers are to be aware of spider traps, which is
a web site where the URLs of dynamically created
links, are modified based of actions taken by the user
or crawler. However these new links do not lead to
existing or new content, but to dynamically created
form pages, or to pages that have already been
visited. Thus a crawler program could go on crawling
inside the web site forever without actually fetching
any new content. These traps are harmful to the
crawler and the server sites, which wastes bandwidth,
disk space to download and store duplicate or useless
data. The crawler caught in a spider trap also be
filling a server-side database with bogus entries. The
database may eventually become filled to capacity,
and the site may be disabled as a result, which is a
type of denial of service attack and is carried out
unwittingly by the crawler.

Rather than crawling pages from the entire Web,
we may want to crawl only pages in certain
categories. To launch such a crawler from an initial
seed set of pages relevant to that category, and see if
any new pages discovered should be added to the
directory, either directly under the category in
question or one of its subcategories. A focused
crawler [4, 6] attempts to bias the crawler towards
pages in certain categories in which the user is
interested. Crawlers, especially when efficient, can
put a significant strain on the resources of Web
servers, mainly on their network bandwidth. A
crawler that sends many page requests to a server in
rapid succession, say ten or more per second, is
considered impolite. The reason is that the server would be so busy responding to the crawler that its service to other requests, including those from human browsing interactively, would deteriorate. In the extreme case a server inundated with requests from an aggressive crawler would become unable to respond to other requests, resulting in an effective denial of service attack by the crawler.

To prevent such incidents, it is essential for a crawler to put in place measures to distribute its requests across many servers, and to prevent any one server from receiving requests at more than some fairly set maximum rate. In a concurrent crawler, this task can be carried out by the frontier manager, when URLs are dequeued and passed to individual threads or processes.

Basically each thread or process works as an independent crawler, except that access to the shared data structures must be synchronized. In particular a frontier manager is responsible for locking and unlocking the frontier data structures so that only one process or thread can read or write to them at one time. Additionally, the frontier manager would maintain and synchronize access to other shared data structures such as the crawl history for fast look-up of visited URLs. This practice not only is required by politeness toward servers, but also has the additional benefits of limiting the impact of spider traps and not overloading the server, which will respond slowly. A crawler consumes three main resources: network, CPU, and disk. Each is a bottleneck with limits imposed by bandwidth, CPU speed, and disk seek times. The sequential crawler makes a very inefficient use of these resources because at any given time two of them are idle while the crawler attends to the third.

For many preferential crawling tasks, labeled examples of pages are not available in sufficient numbers to train a focused crawler [4, 6] before the crawl starts. Instead, a small set of seed pages and a description of a topic of interest to a user or user community. The topic can consist of one or more seeds or even a short query. Preferential crawlers that start with only such information are often called topical crawlers [7, 11, 16]. They do not have text [3] classifiers to guide crawling. 4. Implementation

The crawler algorithm must specify the order in which new URLs are extracted from the frontier to be visited. These mechanisms determine the graph search algorithm [2] implemented by the crawler. A simple breadth-first crawling [9] algorithm will tend to fetch the pages with the highest PageRank by definition. A focused crawler [8] attempts to bias the crawler towards pages in certain categories in which the user is interested. A focused crawler [8] based on a classifier, to build a text classifier using labeled example pages from the focused web site. Then the classifier would guide the crawler by preferentially selecting from the frontier those pages that appear most likely to belong to the categories of interest, according to the classifier's prediction. To train the classifier, example pages are drawn from various categories in the taxonomy. The classification algorithm used was the naive Bayesian method. The focused crawler is the use of a distiller. The distiller applies a modified version of the HITS algorithm [28] to find topical hubs. These hubs provide links to authoritative sources on a focus category. The distiller is activated at various times during the crawl and some of the top hubs are added to the frontier. The Context-Focused Crawlers are another type of focused crawlers, use naive Bayesian classifiers as a guide, but in this case the classifiers are trained to estimate the link distance between a crawled page and a set of relevant target pages [18]. To see why this work, imagine looking for information on machine learning. The focused crawler is the use of a distiller, it applies a modified version of the HITS algorithm [28] to find topical hubs. These hubs provide links to authoritative sources on a focus category. The distiller is activated at various times during the crawl and some of the top hubs are added to the frontier.

The Context-Focused Crawlers are another type of focused crawlers. They also use naive Bayesian classifiers as a guide, but in this case the classifiers are trained to estimate the link distance between a crawled page and a set of relevant target pages [18]. A topical crawler [7,11, 16], a MySpiders applet, is designed to demonstrate two topical crawling algorithms[1,7,11,16], best-N-first and InfoSpiders [10]. It is interactive in that a user submits a query and the Web is crawled in real time. As pages deemed relevant are crawled, they are displayed in a list that is kept sorted by a user-selected criterion: score or recency. The score is simply the content similarity between a page and the query and the recency of a page is estimated by the last-modified header, if returned by the server.

The disadvantage is that the ranking algorithms cannot take advantage of global prestige measures, such as PageRank, available to a traditional search engine.

Crawler Applications are business intelligence, whereby organizations collect information about their competitors and potential collaborators. Another use is to monitor Web sites and pages of interest, so that a user or community can be notified when new information appears in certain places. There are also malicious applications of crawlers, for example, that harvest email addresses to be used by spammers or collect personal information to be used in phishing and other identity theft attacks. The most widespread
use of crawlers is, however, in support of search engines. The most widespread use of crawlers is, however, in support of search engines. In fact, crawlers are the main consumers of Internet bandwidth. They collect pages for search engines to build their indexes. Well known search engines such as Google, Yahoo! and MSN run very efficient universal crawlers designed to gather all pages irrespective of their content. 5. Conclusions Several research issues around crawlers have received attention. One key question is how to identify the environmental signals to which crawlers should attend in order to determine the best links to follow. Rich cues such as the markup and text within Web pages, as well as features of the link graph built from pages already seen, are all reasonable sources of evidence to exploit.

The typical use of all crawlers has been for creating and maintaining indexes for general purpose search engines. Crawlers are becoming important tools to support applications such as specialized Web portals, live crawling, and competitive intelligence. Another characteristic of the way in which crawlers have been used by search engines up to now is the one-directional relationship between users, search engines, and crawlers. Users are consumers of information provided by search engines, search engines are consumers of information provided by crawlers, and crawlers are consumers of information provided by users.

REFERENCES


