University of Michigan
EECS 489

Web Crawler

Joel Acevedo - joelace
Giselle Agosto - gagosto
Maria Ortiz de Zuniga – mozlch

Professor Morley Mao

December 16, 2005

I. Project Overview 3
II. Related Work 3
III. System Description 4
1. Thread Manager 4
2. Crawler Thread 5
a) URL Retrieval from Link Queue 5
b) Robots Exclusion Check 6
c) HTTP Download 6
d) Content Type Filter 6
e) Content Seen Test 6
f) Save File 7
g) Link Extractor 7
h) URL Filter 7
i) URL Seen Test 8
j) Store Internal and External URLs Found 8
3. Scheduler Thread 8
a) Get Most Referenced Links 8
b) Queue Mapping 9
c) Update and Reorder of Link Queues 9
IV .Scheduling Policies 9
V. Results 11
VI. References 13

I. Project Overview

Our project consists of designing and implementing an efficient general purpose web crawler. A web crawler is an automated program that accesses a web site and traverses through the site by following the links present on the pages systematically. The main purpose of web crawlers is to feed a data base with information from the web for later processing by a search engine; purpose that will be the focus of our project.

II. Related Work

Most related work in this area is associated with popular search engines and their crawling algorithms and detailed architecture is kept as a business secret. However, web crawlers such as RBSE, the WebCrawler, the World Wide Web Worm, the crawler of the Internet Archive, an early version of the Google crawler, Mercator, Salticus, the WebFountain and the WIRE have published descriptions of their architecture. Besides structural issues, research about Web crawling has focused on parallelism, discovery and control of crawlers for Web site administrators, accessing content behind forms (the “hidden” web), detecting mirrors, keeping the freshness of the search engine copy high, long-term scheduling , and focused crawling. In addition, there have been studies on characteristics of the Web, that affect directly the performance of a Web crawler such as detecting communities, characterizing server repose time, studying the distribution of web page changes and proposing protocols for web servers to cooperate with crawlers. [Castillo2004].

III. System Description

Figure 1 shows de architecture of JoMaGic web crawler. This section describes the purpose and implementation of each of the components shown.

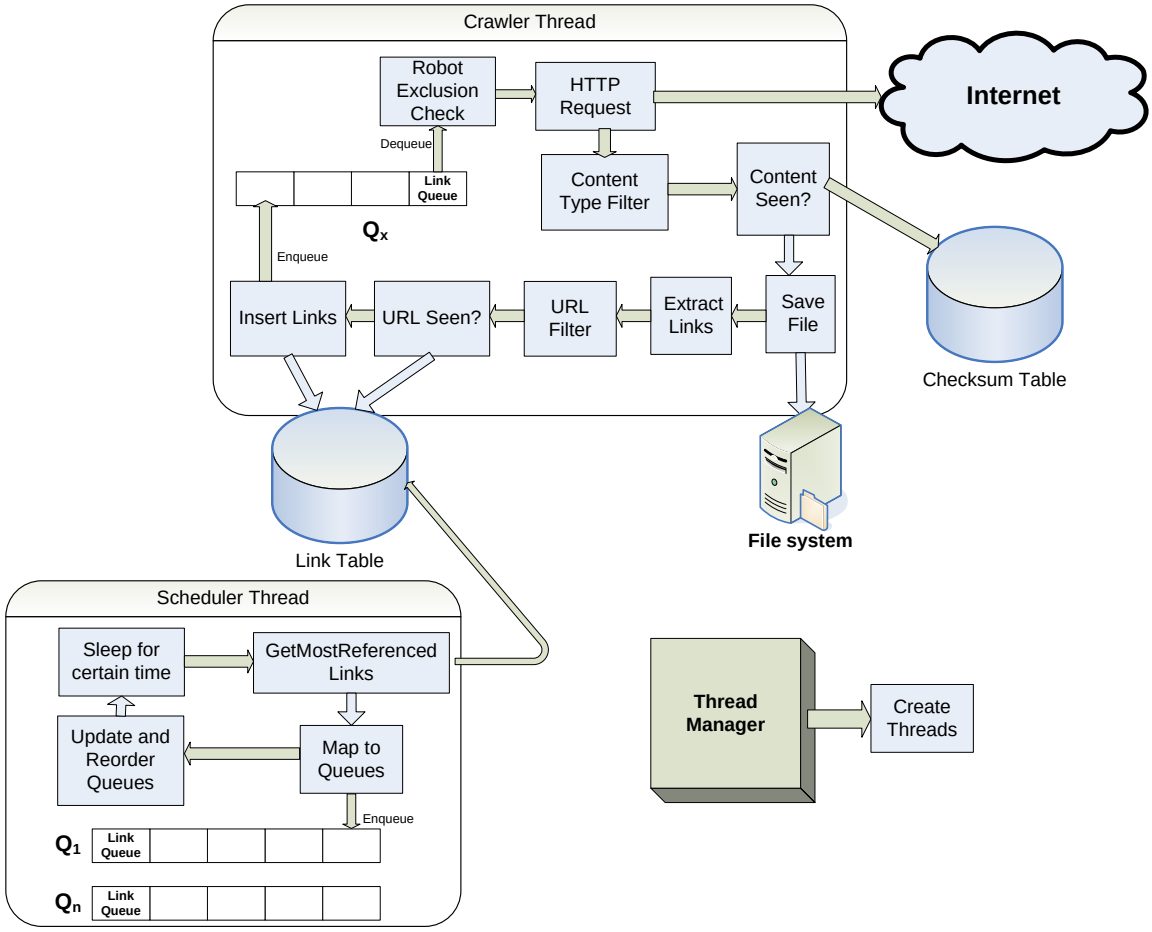

Figure 1 Web Crawler Architecture

1. Thread Manager

The Thread Manager is invoked by the top-level application with a user specified number that describes the number of concurrent connections. The Thread Manager creates and initializes a thread for each of the desired connections, starts them and kills them when the crawling is done or cancelled.

2. Crawler Thread

The Crawler Thread is the component that carries out most of the work in the web crawler. The crawling starts with one or more seed URLs that are specified by the user and inserted into the Link Queue during the crawler's initialization stage. For illustration purposes, the whole process carried out by each Crawler Thread will be divided into modules (that match those depicted in figure 1) and each module will be described separately.

a) URL Retrieval from Link Queue

The Crawler Thread starts by retrieving the URL at the head of its Link Queue if there is one. As will be described shortly, JoMaGic uses an array of Link Queues to store the URLs found before they are processed and this array is stored inside the Scheduler. Each Crawler Thread has access to one and only one of the queues that constitute the Link Queue array to ensure that only one thread is accessing a host at a given time. This guarantees that a given web server will not be overloaded with requests because only one thread in the web crawler will be connecting to it.

Link Queue

As the name implies, this data structure holds the links that have been discovered and keeps them in the order they will be examined. In JoMaGic, each crawler thread has access to one link queue. This data structure is not a regular queue but a queue of link objects like the ones depicted below in figure 2.

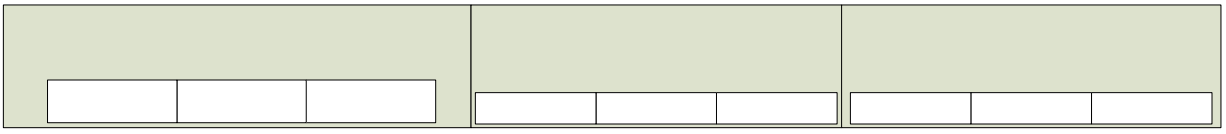

Figure 2: Structure of each Crawler Thread's Queue Link

As can be seen, each element in the Link Queue consists of a host name, rank number and an internal links queue or sub-queue. The scheduler keeps each element (host, rank, sub-queue) in the Link Queue in order based on the rank of each host (see next section). When the Crawler Thread requests a URL it retrieves the first URL in the sub-queue of the element that is stored in the front of the Link Queue. After retrieval the URL is taken out of the sub-queue. When an element's sub queue is emptied, the whole element is dequeued from the Link Queue.

b) Robots Exclusion Check

Each thread has a hash table which maps excluded web pages to a particular host. Every time a URL is retrieved from the Link Queue, its host will be looked up in this table to see if it can be crawled or not. If the host for this page is not in the hash table, its robots.txt file will be requested and parsed in order to extract the excluded pages for that host.

c) HTTP Download

If the addressed web page should not be excluded, the Crawler Thread proceeds to download the page's content using the http protocol.

d) Content Type Filter

Before processing the downloaded page, the thread checks its MIME type by examining the HTTP response's *Content-Type* header. If the MIME type is not *text/html* the Crawler Thread discards the URL and restarts the process by retrieving a new URL from its Link Queue.

e) Content Seen Test

If the downloaded object passes the Content Type Filter, a test is run on it to make sure that its content has not been processed before by the web crawler.

This is done by calculating a checksum based on the byte content of the page and comparing it with those previously stored in a database. The checksum is calculated using a class which is part of the Java API and it serves as a fingerprint that identifies the content of a web page.

f) Save File

If the page content passes the Content Seen Test described above, the Crawler Thread proceeds to store it in the file system as an html file. The file is stored in a path, relative to a root directory, that corresponds directly to the file's URL. For instance, if the URL of the downloaded page is http://amadeus.uprm.edu/~s00016/web the file will be saved as:

<root dir> \amadeus_uprm_edu\~s00016\web\index.html

g) Link Extractor

After saving the web page, the Crawler Thread parses the page extracting references to other objects as they are found. Many web crawlers are limited to hyperlinks but JoMaGic extracts all references by means of the *href* HTML attribute, and HTML frames. As each link is found it is also classified as external or internal. An external link is a link which URL points to an object in a different host from that of the web page where it was found. An internal link has the same host as the web page where it was found.

h) URL Filter

The URL filter rejects URLs that JoMaGic is not interested in based on the extension of the file it points to. The filter look for addressed files containing popular file extensions such as pdf, doc, ppt, avi, mpeg, jpg, bmp and mov, and discard them from the set of URLs that will be processed in the future.

i) URL Seen Test

Both the internal and external URLs that survive the URL Filter have to be checked against a record of visited pages stored in the database. All the URLs that are already stored in the database are discarded and those that do not exist in it are inserted. Before rejecting the previously visited URLs, all URLs that get to the URL Seen Test are used to calculate the ranking of the hosts that have been seen by all threads up to this point.

j) Store Internal and External URLs Found

During this stage, the thread stores the internal and external links found for later processing. Internal links are inserted by the thread both in the database and in the sub-queue from where the URL where they were found was extracted. External links are inserted in the database so that the Scheduler can decide when to extract them, in which of the Link Queues to insert them and their position inside the selected Link Queue.

3. Scheduler Thread

The scheduler thread is triggered at fixed intervals of time during the crawling process. Each time it carries out the following procedure:

a) Get Most Referenced Links

In this stage, the scheduler calculates the rank of all the visited hosts so far and determines which are the ones with the highest rank. The next section Scheduling Policies explains in detail how this “rank” is determined for all the hosts that are crawled.

b) Queue Mapping

For each host that results from the calculations described above, the Scheduler gets all links that have not been processed and performs one of the following: the Scheduler either creates a new element (host, rank, sub-queue) and inserts it in an available Link Queue or it inserts it into its appropriate sub-queue, depending on whether the URL's host exists already somewhere in the Link Queue array or not.

c) Update and Reorder of Link Queues

At this stage, the Scheduler Thread goes through each Link Queue, updates the ranks of each of its elements and reorders it. The reorderind is based on each element's rank.

IV .Scheduling Policies

The long-term scheduling policy implemented by the JoMaGic web crawler is based on the importance metric known as back link count. An importance metric is basically a way to measure the most referenced web pages in the Internet and therefore download them before other pages with less importance. Back link count is based on determining the importance of a given web page by counting the number of pages that link to it or reference it. Intuitively, a page *p* that is linked to many pages is more important than a page that is seldom referenced [Cho98]. In order to calculate the importance of a web page *I(p)* based on back link count, all the back links for a certain web page throughout the Web must be counted. Our crawler estimates this measurement using *I'(p)*, which is an estimate of the back link count for a page *p* based on the pages that have been crawled so far.

The following algorithm taken from [Cho98] entails how back link count is calculated. We can see that for every url u, its back link count is equal to the number of times u appears in a queue called links, which holds all the URLs that have been extracted from all the pages that have been visited so far. Once this number is obtained the queue that holds the URLs to be downloaded (url queue) is reordered based on it.

backlink count, *I*(*p*)
foreach *u* in url queue
  backlink count[*u*] = number of terms *u* in links
  sort url queue by backlink count[*u*]

The back link count algorithm described above was implemented in the JoMaGic web crawler as follows: Everytime a group of links are extracted from a certain webpage by the thread in charge of doing so, these links go to a database which holds all of the links that have been obtained throughout the crawl, their back link count and their current status (not processed, in process, processed). If a link is in the database, but has still not been assigned to any thread for crawling it is considered not processed, if it is inside a thread's queue waiting to be processed, it is considered in process, and finally if it has already been visited and is no longer in a thread's queue it is considered processed. When this happens, there's a database function (url_seen) that will verify if this link is already in the database. If it is, the backlink count for that link will be incremented, if it isn't the link will be added to the database. Every *n* number of seconds, the scheduler thread will query the database for the *x* hosts with a largest back link count and that have still not been processed. The scheduler will take the URLs returned by the database and will allocate them to different threads for crawling. The way this allocation to threads is done is as follows: First the scheduler will verify if there's any thread that is already downloading from the same host as any of the URLs that it is about to allocate. It then allocates these URLs to their respective threads. The URLs that did not map to any of the threads, will then be distributed throughout the threads in a round robbing fashion. Once all the URLs have been allocated to a thread, the queues that hold the links to be crawled for each thread

will be updated and reordered. This will ensure that each thread is always crawling first the host in its queue with the largest back link count. The update and reorder of each queue is done as follows: Each thread will query the database to obtain the new back link count for each of the hosts that it has in its queue. This back link count is obtained by adding all of the back link counts of all the URLs that belong to this host and are currently in the database. Once the new back link count is obtained for each host in the queue, the queue is reordered so that the first host in the queue is the one with the largest back link count.

The short term scheduling policy used in the JoMaGic web crawler is breadth first ordering. This is based on the notion that the most important pages in a host are the first ones encountered, and as we begin entering deeper in to a hosts' pages they become less important. When a thread extracts links from a page it will sort them as external and internal. If the links are internal, meaning that they belong to the same host as the page where they were found, the thread that found them will insert them in its queue and in the database and will mark them as in process. Therefore, the scheduler will never have to deal with these links. Using this approach pages are inserted in the queue, in the order that they are found, ensuring that the ones that where found first are crawled first.

V. Results

In order to measure the performance of our web crawler a series of metrics will be used:

1) **Percentage of Important Pages** – This metric, proposed by [Cho98] uses the percentage of pages with high importance that were crawled in order to measure the crawler's performance. In order to do this a target back link count G must be chosen. Then H is the number of pages that were crawled and have a back link count > G, and T is the number of pages that

were downloaded. Therefore H/T is the crawler's performance. If the H is equal to T the crawler's performance is 1.

2) **Diversity of Pages downloaded** – This metric shows how many different domains were visited by the crawler. This data can be obtained by querying the database after the crawl has finished for all the different hosts that have were processed.
3) **Pages Downloaded per Time –** This metric can be used to measure how fast is our web crawler. When the crawl is finished, the database can be queried for all the processed URLs and this number is divided by the total time that the crawl lasted.

Since our web crawler has a series of parameters that can be customized like the number of threads, the scheduling time and the maximum size of the queue for a thread, these parameters can be changed in order to see how they affect the performance metrics described above.

A web crawler is a very powerful tool that can be used to obtain useful data from the Internet. The following valuable Internet information can be obtained with the JoMaGic web crawler:

1) **Duplicate pages in the Web** – Since our crawler verifies the checksum of a given web page against the checksums of the pages that have been crawled already, it can easily count how many times two pages with the same content were found.

2) **Percentage of Broken Links** – Since our crawler keeps track of all the pages that are being visited , it could keep a count of how many of these pages are answered with a 404 Not Found instead of a 200 OK.

3**) Pages with a Higher Level have Higher Back Link Count** – The JoMaGic web crawler keeps track of the back link count of all the URLs that are found. This number can be used to prove the scheme that we are using for short term scheduling. We can compare the back link count of pages that are found first when crawling pages in a host against the back link count of pages that were found deeper in the same host.

4) **Most Important Web pages** – When the crawling is done, we can determine the most important pages in the Web, based on the highest back link counts.

Due to the fact that a crawl that can be used to obtain the above data accurately and to measure performance has to be very extensive and must be run for a day or more, these measurements are not available for this report. The crawls to obtain these data are being run as this report is being written. Therefore this data will be available for the face to face grading.

VI. References

1. Castillo, C., Marin, M., Baeza-Yates, R., Rodriguez, A. (2004). Scheduling Algorithms for Web Crawling. In Latin American Web Conference (WebMedia/LA-WEB), Riberao Preto, Brazil, 2004. IEEE Cs. Press

2. A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. World Wide Web Conference, 2(4):219–229, April 1999.

3. J. Cho, H. Garc´ıa-Molina, and L. Page. Efficient crawling through URL ordering. In Proceedings of the seventh conference on World Wide Web, Brisbane, Australia, April 1998