declarations from a web site before downloading any real content from it. Our crawling system before adding a new URL to the table with the URLs to be crawled, it examines whether this URL is excluded by the robot protocol or not. If a URL is excluded, it will not be added to the table and thus will not be crawled.

Secondly this crawler pays attention to its visits between pages of the same domain. It starts visiting all the URLs of the table whose level is not zero and extracts their URLs. And it repeats this procedure for all internal URLs it finds. In other words, it does not take the first URL of the table, finds its URLs and then visits these URLs to extract their internal URLs. The crawler visits the pages of the same domain with such a frequency that does not overload the pages and this is very important especially for the portals that are visited million times every day. Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3-4 minutes. The lower bound of access intervals of the crawler that we describe is 10 seconds. Another crucial point of our crawling system is that the time it visits the same pages changes daily. Therefore the pages are not overloaded the same time every day and they are able to meet efficiently requests of users during the day.

3.2 Content Based Adaptation of the Crawler

The second important characteristic of our crawling system is adaptation. This crawler is capable of adapting its visiting frequency to the rate of change of the page content. This is implemented by the following procedure. After a page has been crawled several times, the system examines the rate of change of its content and proportionally increases or decreases the frequency it visits this page. In more detail, the table of the Database, where the URLs that will be crawled are stored, has among others, the following fields: size, date, fr (frequency) and fr_max (maximum frequency). Size represents the size of the page, date represents the last date the page was crawled, fr_max represents the crawler's visiting frequency to the page and fr represents the number of crawlings that should be done before the page can be recrawled again. In each crawl, fr is decreased until it becomes zero. Then, the page should be crawled and after the crawling procedure, fr takes the value of fr_max. We present a flow diagram below that explains better this part of our crawling system.

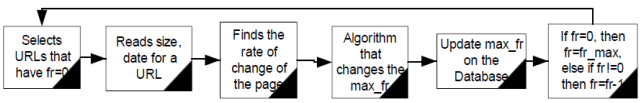

Figure 3: Flow diagram that shows how the system manipulates visiting frequency

The algorithm that calculates new max_fr, based on the rate of change of a page is the following:

records → the entries of a page at the table
limit → the number of records that we examine for changes on the content of the page.
counter → a variable that increases when the content of two sequential records differs.

variable x takes the minimum value between records and limit

if counter is equal to zero
then variable b takes the value of x
and new_maxfr becomes the lower bound of (maxfr + b) / 2
else if counter is not equal to zero
then variable b takes the value of (x / counter)
and new_maxfr becomes the lower bound of (maxfr + b) / 2

For example, we assume that at the table, there are 45 records of a page that its max_fr is 3 and we would like to examine the last 20 records for changes on its content. If it has changed 10 times during the period of the last 20 crawlings, then according to the above algorithm, we have:

x = 20;
b = 20/10 = 2;
new_maxfr = (3+2)/2 = 5/2 = 2,5 → 2.

That means that the crawler should crawl this page more often and in particular every two days and not three. Finally, we should mention that an efficient crawler is only interested in collecting the differences in the content of a certain page. Visiting frequently a page whose content does not change, provides no information to the system and burdens both the system and the page.

3.3 To Describe a Selective Incremental Algorithm

Crawlers whose procedure is quite simple, crawl all pages without carrying out any inspection to find out whether the pages have changed or not. Thus they are crawling pages whose content is exactly the same to that of the last crawling, extracting no information and managing only to overload both pages and the system. However, there are also crawlers that choose not to crawl a page that has not changed recently. This is clever enough but there is a drawback as the crawling system may not be informed of the changes of the pages the internal URLs point at. For instance, the following figure represents a page and its internal links. If page A will not be crawled by the system because it has not changed recently, then none of the pages B, C, D, E, F and G will be crawled. Nevertheless, their content may have changed but the system will not be able to be informed of it.

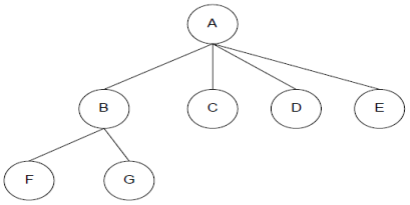

Figure 4: A page and its internal links

A selective incremental crawler is capable to resolve the situation when a page has not changed but the pages its internal URLs point at have changed. For example, if page A has not changed, the crawler examines whether pages B, C, D and E have changed or not. If a high percentage of them have changed, page A will be crawled, otherwise it will not and the

crawler will take the next URL from the table with the URLs to be crawled. From experiments that we made, we came to the conclusion that we crawl page A when 40% of its pages have changed. This percentage is not constant but we recalculate it every time is the turn of page A to be crawled. In proportion to the number of pages that have changed or not, we increase or decrease the percentage respectively.

Another way of recrawling pages is by checking every page separately whether it has changed or not. For instance, the crawler will first examine whether page A has changed and if it has, it is crawled and then the next URL of the list is taken to be examined. Otherwise, if page A has not changed, the crawler examines its internal pages for changes. Thus, it checks page B. If its content differs from the one that was crawled the last time, it is crawled and afterwards page C is examined. If page B is the same, pages F and G are then checked. This way of recrawling may be the simplest, but it is a very time-consuming procedure and it presupposes that each page is not unknown to the system. The crawling system should know the tree diagram of each page, in other words its internal links.
On the contrary, the recrawling policy that is followed by the crawler that we describe, seems to be more clever than the one that we just reported. It is quicker, flexible and efficient but the basic and most important difference from the above policy is that each page is considered as a black box, in other words as something unknown that the system should approach and analyze. If the examination of the content of a page shows that the page has changed then it is still treated as something unknown to the system. The page ceases to seem unfamiliar to the system only when it is found that its content has not changed. In this case, the system knows the internal URLs of the page which examines to find out how many of them have changed and decide whether to crawl the page or not.

Our crawler is quite friendly to the pages it visits as it crawls them only when they have changed. Moreover, it is very efficient because it is interested in crawling information that is fresh, ignoring data that are not up-to-date. We should also remark on the cleverness of the crawler we describe. With just a look at the database, it avoids visiting pages whose content is stable and their crawling would put additional load both on the crawling system and on the pages. Finally the crawler learns more about the pages as it runs and its low speed which is a consequence of the fact that it is a selective incremental crawler, is not a disadvantage of the system but it makes it seem even more friendly to the pages it visits.

4. Enhancing Security and to Avoid Traps

A crawler trap is a URL or set of URLs that cause a crawler to crawl indefinitely. Some crawler traps are unintentional. For example, a symbolic link within a file system can create a cycle. Other crawler traps are introduced intentionally. For example, people have written traps using CGI programs that dynamically generate an infinite web of documents. The crawler we describe does not read CGI programs and thus it is not threatened.

There is no automatic technique for avoiding crawler traps. However, sites containing crawler traps are easily noticed due to the large number of documents discovered there. A human operator can verify the existence of a trap and manually exclude the site from the crawler's purview using the customizable URL filter.

The URL filtering mechanism provides a customizable way to control the set of URLs that are downloaded. Before adding a URL to the table with the URLs to be crawled, the crawling system consults the user-supplied URL filter. The URL filter class has a single crawl method that takes a URL and returns a boolean value indicating whether or not to crawl that URL.

Nevertheless, it is possible a crawler to include a collection of different URL filter subclasses that provide facilities for restricting URLs by domain, prefix or protocol type and for computing conjunction, disjunction or negation of other filters. Users may also supply their own custom URL filters, which are dynamically loaded at start-up.

The crawler, we describe in this paper also uses a time limit in order to avoid traps. As we mentioned above, obvious traps are gigabyte-sized or even infinite web documents. Similarly, a web site may have an infinite series of links (eg. "domain.com/home?time=101" could have a self-link to "domain.com/home?time=102" which contains link to "…103", etc…). Deciding to ignore dynamic pages results in a lot of skipped pages and therefore the problem is not completely fixed. Our crawler uses a time limit which refers to the period a page is being crawled. If a page while crawling passes this limit, it stops being crawled and the next page is fetched to be crawled, encountering efficiently this kind of traps.

5. Future Work – Conclusion

Because of the dynamism of the Web, crawling forms the back-bone of applications that facilitate Web information retrieval. In this paper, we describe the architecture and implementation details of our crawling system, also presented some preliminary experiments. We explained the importance of extracting only content from web pages and how this can be implemented by a mechanism content analysis, corresponding crawling policies and clever systems that extract content of high quality. These are obviously many improvements to the system that can be made. A major open issue for future work is a detailed study of how the system could become even more distributed, retaining though quality of the content of the crawled pages.

When a system is distributed, it is possible to use only one of Its component or easily add a new one to it.

References

[1] D. Sullivan, "Search Engine Watch," Mecklermedia, 1998.
[2] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Stanford University, Stanford, CA, Technical Report, 1997.

[3] O. A. McBryan, "GENVL and WWW: Tools for Taming the Web," in Proceedings of the First International Conference on the World Wide Web, Geneva, Switzerland, 1994.
[4] B. Kahle, "Archiving the Internet," Scientific American, 1996.
[5] J. Gosling and H. McGilton, "The Java Language Environment," Sun Microsystems, Mountain View, CA, White Paper, April 1996.
[6] J. E. White, Mobile Agents, MIT Press, Cambridge, MA, 1996.
[7] C. G. Harrison, D. M. Chess, and A. Kershenbaum, "Mobile Agents: Are they a good idea?," IBM Research Division, T.J. Watson Research Center, White Plains, NY, Research Report, September 1996.

Author Profile

Pallavi received the B.TECH degree in information technology and M.TECH. degree in Computer Science and Engineering from maharshi dayanand university in 2013 and 2015,respectively.