© 2020 IJSRST | Volume 7 | Issue 7 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology
DOI : 10.32628/IJSRST

Study of Algorithms and Techniques for Effective Web Crawling

Suman Kumar
M. Phil. Students, Department of Physics, B. R. A. Bihar University, Muzaffarpur-842001, Bihar, India

ABSTRACT

In this present paper, we report on study of algorithms and techniques for effective web crawling. electrical components for fully implantable device. This paper discusses the various components of a fully implantable multi-electrode recording system, and the design issues surrounding each component.
Keywords: IC, Fully Differential Low Noise Amplifiers, CMFB

I. INTRODUCTION

The typical design of search engines is a "cascade", in which a Web crawler creates a collection which is indexed and searched. Most of the designs of search engines consider the Web crawler as just a first stage in Web search, with little feedback from the ranking algorithms to the crawling process. This is a cascade model, in which operations are executed in strict order: first crawling, then indexing, and then searching [1-4].
Our approach is to provide the crawler with access to all the information about the collection to guide the crawling process effectively. This can be taken one step further, as there are tools available for dealing with all the possible interactions between the modules of a search engine. The indexing module can help the Web crawler by providing information about the ranking of pages, so the crawler can be more selective and try to collect important pages first.
Web crawling is the process used by search engines to collect pages from the Web. This work studies Web crawling at several different levels, ranging from the long-term goal of crawling important pages first, to the short-term goal of using the network connectivity efficiently, including implementation issues that are essential for crawling in practice. We start by designing a new model and architecture for a Web crawler that tightly integrates the crawler with the rest of the search engine, providing access to the metadata and links of the documents that can be used to guide the crawling process effectively [5].

II. EXPERIMENTAL SETUP

We tested several scheduling policies in two different datasets corresponding to Chilean and Greek Web pages using a crawler simulator. This section describes how the dataset and how the simulator works.

2.1. Datasets: .cl and .gr

The Web, and found that the Web graph is self-similar in several senses and at several scales, and that this self-similarity is pervasive, as it holds for a number of different parameters. Top-level domains are useful because

they represent pages sharing a common cultural context; we consider that they are more useful than large Web sites because pages in a Web site are more homogeneous. Note than a large sub-set of the whole Web (and any non-closed subset of the Web) is always biased by the strategy used to crawl it [6].

We worked with two datasets that correspond to pages under the .cl (Chile) and .gr (Greek) top-level domains. We downloaded pages using the WIRE crawler [89] in breadth-first mode, including both static and dynamic pages. While following links, we stopped at depth 5 for dynamic pages and 15 for static pages, and we downloaded up to 25,000 pages from each Web site. We made two complete crawls on each domain, in April and May for Chile, and in May and September for Greece. We downloaded about 3.5 million pages in the Greek Web and about 2.5 million pages in the Chilean Web. Both datasets are comparable in terms of the number of Web pages, but were obtained from countries with wide differences in terms of geography, language, demographics, history, etc [7].

2.2. Crawler simulator

Using this data, we created a Web graph and ran a simulator by using different scheduling policies on this graph. This allowed us to compare different strategies under exactly the same conditions.

The simulator 1 models:

- The selected scheduling policy, including the politeness policy.
- The bandwidth saturation of the crawler Internet link.
- The distribution of the connection speed and latency from Web sites, which was obtained during the experiment described in Section 3.3
- The page sizes, which were obtained during the crawl used to build the Web graph.

We considered a number of scheduling strategies. Their design is based on a heap priority queue whose nodes represent sites. For each site-node we have another heap with the pages of the Web site, as depicted in Figure 1.

At each simulation step, the scheduler chooses the top website from the queue of Web sites and a number of pages from the top of the corresponding queue of Web pages. This information is sent to a module that simulates downloading pages from that Website [8].

III. SIMULATION PARAMETERS

The parameters for our different scheduling policies are the following:

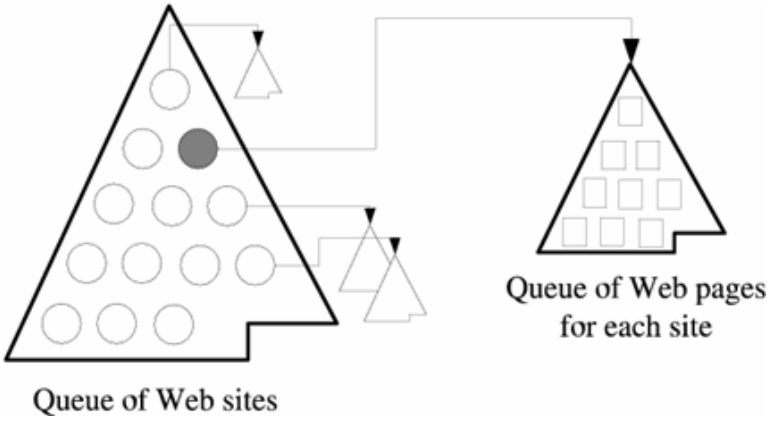

Figure 1: Queues used in the crawling simulator. We tested the scheduling policies using a structure with two levels: one queue for Web sites and one queue for the Web pages of each Web site.

- The policy for ordering the queue of Web sites, related to long-term scheduling.
- The policy for ordering the queues of Web pages, related to short-term scheduling.
- The interval w in seconds between requests to a single Web site.
- The number of pages c downloaded for each connection when re-using connections with the HTTP Keep-alive feature.
- The number r of maximum simultaneous connections, i.e.: the degree of parallelization. Although we used a large degree of parallelization, we restricted the robots to never open more than one connection to a Web site at a given time.

3.1. Interval between connections (w)

As noted above, a waiting time of $w = 60$ seconds is too large, as it would take too long to crawl large Web sites. Instead, we use $w = 15$ seconds in our experiments.

Liu *et al.* show that total time of a page download is almost always under 10 seconds. We ran our own experiments and measured that for sequential transfers (which are usually faster than parallel transfers) 90% of the pages were transferred in less than 1.5 seconds, and 95% of the pages in less than 3 seconds, as shown in Figure 2.

From the total time, latency is usually larger than the actual transfer time. This makes the situation even more difficult than what was shown in Figure 2, as the time spent waiting cannot be amortized effectively [9].

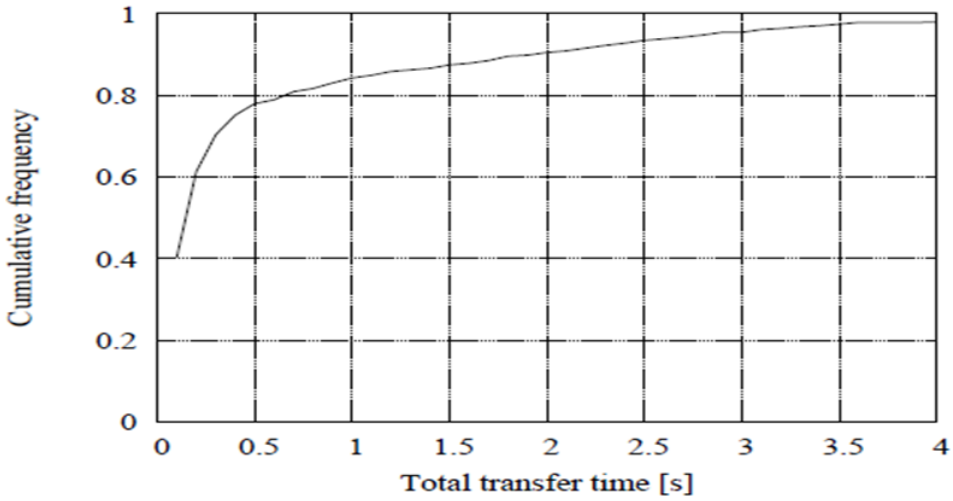

Figure 2: Total download time for sequential transfer of Web pages; this data provides from experiments in the Chilean Web and was used as an input for the Web crawler simulator.

3.2. Number of pages per connection (c)

We have observed the log files of several Web servers during this thesis. We have found that all the Web crawlers used by major search engines download only one page per each connection, and do not re-use the HTTP connection. We considered downloading multiple pages in the same connection to reduce latency, and measured the impact of this technique in the quality of the scheduling.

The protocol for keeping the connection open was introduced as the Keep-alive feature in HTTP/1.1; the configuration of the Apache Web server enables this feature by default and allows for a maximum of 100

objects downloaded per connection, with a timeout of 15 seconds between requests, so when using $c > 1$ in practice, we should also set $w \cdot 15$ to prevent the server from closing the connection [10].

3.3. Number of simultaneous requests (r)

All of the robots currently used by Web search engines have a high degree of parallelization, downloading hundreds or thousands of pages at a given time. We used $r = 1$ (serialization of the requests), as a base case, $r = 64$ and $r = 256$ during the simulations, and $r = 1000$ during the actual crawl.

IV. CONCLUSIONS

As we never open more than one connection to a given Web site, r is bounded by the number of Web sites available for the crawler, i.e.: the number of Web sites that have unvisited pages. If this number is too small, we cannot make use of a high degree of parallelization and the crawler performance in terms of pages per second drops dramatically. The scarcity of large Web sites to download from is especially critical at the end of a large crawl, when we have already downloaded all the public pages from most of the Web sites. When downloading pages in batches, this problem can also arise by the end of a batch, so the pages should be carefully selected to include pages from as many Web sites as possible.

V. REFERENCES

[1]. Demetrios Zeinalipour-Yazti and Marios D. Dikaiakos. Design and implementation of a dis-tributed crawler and filtering processor. In Proceedings of the fifth Next Generation Information Technologies and Systems (NGITS), volume 2382 of Lecture Notes in Computer Science, pages 58–74, Caesarea, Israel, June 2002. Springer.

[2]. Larry Wall, Paul Eggert, Wayne Davison, and David MacKenzie. GNU patch. http://www.gnu.org/software/patch/patch.html, 2000.

[3]. WebTrends corporation. http://www.webtrends.com/, 2004.

[4]. WebDAV resources. http://www.webdav.org/, 2004.

[5]. Eveline A. Veloso, Edleno de Moura, P. Golgher, A. da Silva, R. Almeida, A. Laender, B. Ribeiro-Neto, and Nivio Ziviani. Um retrato da web brasileira. In Proceedings of Simposio Brasileiro de Computacao, Curitiba, Brasil, July 2000.

[6]. Pang-Ning Tan and Vipin Kumar. Discovery of web robots session based on their navigational patterns. Data Mining and Knowledge discovery, 6(1):9–35, 2002.

[7]. Danny Sullivan and Chris Sherman. Search Engine Watch reports. http://www.searchengine-watch.com/reports/, 2004.

[8]. Markus Sobek. Google dance the index update of the Google search engine. http://dance.efactory.de/, 2003.

[9]. Knut Magne Risvik and Rolf Michelsen. Search engines and web dynamics. Computer Networks, 39(3), June 2002.

[10]. Yanhong Li. Toward a qualitative search engine. IEEE Internet Computing, pages 24 – 29, July 1998.