AIN SHAMS UNIVERSITY
Faculty of Computer &
Information Sciences
Scientific Computing Department

A Memory Efficient Approach for Arabic Web Crawling

Thesis submitted to the Department of Scientific Computing
Faculty of Computer and Information Sciences
Ain Shams University

In partial fulfillment of the requirements for the degree
of Master in Computer and Information Sciences

By

Doaa Ezzat Mohammed Mahmud

B.Sc. in Computer and Information Sciences (2005)
Ain Shams University – Cairo

Under the supervision of

Prof. Dr. Mohamed F. Tolba

Scientific Computing Department
Faculty of Computer and Information Sciences
Ain Shams University

Dr. Mohamed Abdeen

Computer Science Department
Faculty of Computer and Information Sciences
Ain Shams University

Dr. Nagwa Badr

Information Systems Department
Faculty of Computer and Information Sciences
Ain Shams University

Cairo-2010

Acknowledgement

I acknowledge my deep gratitude to ALLAH the most beneficent and most merciful, who helped me complete this work on a level that I hope will please the reader.

First of all I would like to thank my dear family, my husband, Mother, Father and brothers Tarek and Mohamed whose kindness, care and never ending support and encouragement made me the person I am today.

Special thanks are due to my supervisors; Prof. Dr. Mohamed Fahmy Tolba, Dr. Mohamed Abdeen, and Dr. Nagwa Badr at the Faculty of Computer and Information Sciences, Ain Shams University, who constantly guided me in elaborating this thesis, assisted me in understanding and analyzing problems, and continuously provided support and valuable comments.

I would also like to thank Dr. Hossam Mahgoub who made a lot of valuable contributions and help me finish this work.

Finally I would like also to thank all my friends who pushed me either directly or indirectly to finish this work in the moments I thought it never will.

Abstract

The plentiful content of the World Wide Web is useful to millions. Some simply browse the Web through entry points. But many information seekers use a search engine to begin their Web activity. In this case, users submit a query, typically a list of keywords, and receive a list of Web pages that may be relevant, typically pages that contain the keywords. Now, search engines became very essential information resources for net users and they form a very important commercial industry.

Web crawlers represent a significant component in web search engines. They are responsible for making a local copy of web pages and keeping this local copy up-to-date by periodically refreshing these pages. This copy is then indexed for further fulfillment of user queries. Web crawlers start with a set of seed URLs, download web pages, and extract links from the downloaded pages for further download. This process is repeated till available resources are exhausted.

Designing an efficient web crawler has many challenges. One of these challenges is to find an appropriate architecture to distribute the crawling work over multiple machines. Another challenge is the refreshing policy of the local collection, since the web is changing very rabidly, it's very challenging to maintain the local copy up-to-date, and this requires studying the evolution of web pages. Downloading important pages first also presents yet another challenge. Due to the massive size of the web, it is practically impossible to download the whole web. In the literature the significant part of the web is only considered. Duplicate download avoidance and language identification are also considered web crawling challenges.

Refreshing web pages is significant for search engines because of the very dynamic nature of the web. Ideally, a web page should be re-crawled as soon as it is changed. However, this is practically impossible. Therefore, page change time needs to be estimated. Researchers found that the only way to predict the change time in the future is to look at the change history in the past. In other words, the change rate helps compute the refresh rate.

In the literature a refresh technique, called curve fitting policy, is cited. This technique is a re-crawl scheduling policy based on the life time of a page. It treats the page as a set of shingles, and then computes the divergence between each snapshot and the base snapshot (the first snapshot). Then it obtains number of change profiles that summarize the change history of the page.

Many web users are interested in Arabic web browsing whether the reason is academic or commercial. To make the curve fitting policy more suitable for Arabic web pages, this thesis proposes some modification to this policy. The proposed technique can reduce the memory consumption by about 90%. It also reduces the time needed to calculate the refresh rate by about 50%.

This memory efficient web crawler has only one problem. This problem is the huge delay in the total time. In many cases, the total time after the proposed modification is more than five times the total time before this modification. To speed up the crawling process, this thesis proposes a parallel technique for this Arabic web crawler. This research proved that the optimal number of processors needed for this parallelization is 10 processors. Using this number of processors, the total time is reduced by 42% from the sequential time. This parallel technique is very efficient when applied on web pages with large contents.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Search engines are enabling tools to mine the massive information content of the World Wide Web. They are very useful for many information seekers on the web. They have evolved over the last two decades until reaching the level of maturity we see today, encompassing, quick, and easy to use.

There are many challenges in building good search engines. One of these challenges is the continuous and rapid growth of the web. The growth rate of the web is even more dramatic. According to [1], and [2], the size of the web has doubled in less than two years.

Aside from the newly created pages, the existing pages are continuously updated [3], [4], [5], [6]. For example, in a study of over half a million pages over 4 months [6], it's found that about 23% of pages changed daily. In the .com domain 40% of the pages changed daily, and the half-life of pages is about 10 days (in 10 days half of the pages are gone, i.e., their URLs are no longer valid). In [6], it's reported that a Poisson process is a good model for web page changes.

Another challenge is the interlinked nature of the web that sets it apart from many other collections. Several studies aim to understand how the web's linkage is structured and how that structure can be modeled [7], [8], [9], [10], [6]. One study, for example, suggests that the link structure of the web is somewhat like a "bow-tie" [7]. That is, about 28% of the pages constitute a strongly connected core (the center of the bow tie). About 22% form one of the tie's loops: these are pages that can be reached from the core but not vice versa. The other loop consists of 22% of

the pages that can reach the core, but cannot be reached from it. The remaining nodes can neither reach the core nor can be reached from the core.

Search engines have main five components [11], as shown in Figure 1.1:

1- A crawling module for downloading web pages.
2- An indexing module for generating a lookup table for the downloaded pages.
3- A page repository for containing a local copy of the downloaded pages.
4- A query engine for fulfilling the user queries.
5- A page ranking module for sorting the search results.

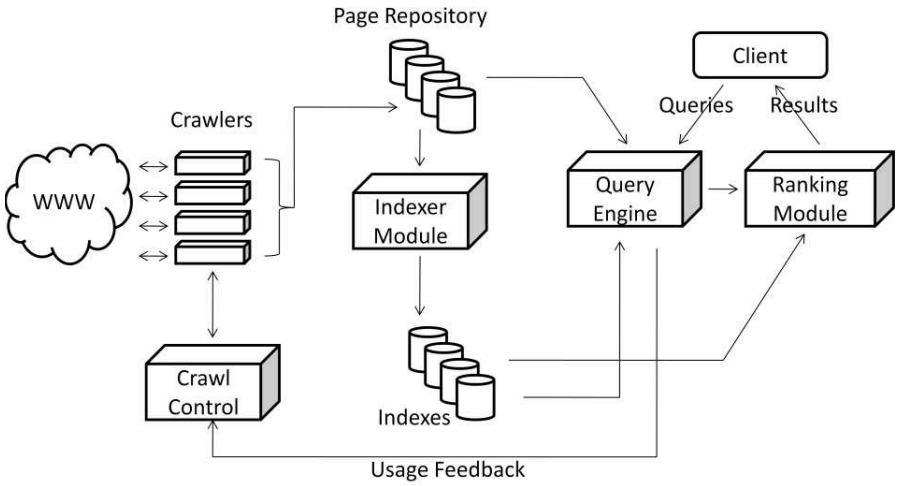

Figure 1.1: The general architecture of search engines.

The crawling module is responsible for making a local copy of web pages and keeping this local copy up-to-date by periodically refreshing these pages. The decision to refresh a web page is a tradeoff between the resource utilization and the freshness of the page content. There are various policies as to when to perform a page refresh. One of these policies is the curve fitting policy [12]. This policy depends on the information longevity.

The following sections are organized as follows: section 1 provides a detailed description for each component in the search engine. Section 2 focuses on the web crawling. Section 3 highlights the motivation of the work presented in this thesis. A summarized explanation for the main objectives of the work presented in this thesis is given in section 4. And finally section 5 outlines the organization of the remaining parts of the thesis.

1.1 Preliminaries

The general architecture of the search engine is described in the following sub-sections.

1.1.1 The Crawling Module

Every engine relies on a crawler module to provide the grist for its operation (shown on the left in Figure 1.1). Crawlers are small programs that `browse' the web on the search engine's behalf, similarly to how a human user would follow links to reach different pages. The programs are given a starting set of URLs, whose pages they retrieve from the web. The crawlers extract URLs appearing in the retrieved pages, and give this information to the crawler control module. This module determines what links to visit next, and feeds the links to visit back to the crawlers. (Some of the functionality of the crawler control module may be implemented by the crawlers themselves.) The crawlers also pass the retrieved pages into a page repository. Crawlers continue visiting the web, until local resources, such as storage, are exhausted.

This basic algorithm is modified in many variations that give search engines different levels of coverage or topic bias. For example, crawlers in one engine

might be biased to visit as many sites as possible, leaving out the pages that are buried deeply within each site. The crawlers in other engines might specialize on sites in one specific domain, such as governmental pages. The crawl control module is responsible for directing the crawling operation.

Once the search engine has been through at least one complete crawling cycle, the crawl control module may be informed by several indexes that were created during the earlier crawl(s). The crawl control module may, for example, use a previous crawl's link graph to decide which links the crawlers should explore, and which links they should ignore. Crawl control may also use feedback from usage patterns to guide the crawling process (connection between the query engine and the crawl control module in Figure 1.1).

1.1.2 The Indexing Module

The indexing module is responsible for the process of generating a lookup table with all the URLs that point to pages containing a given word. This module is the most critical part of any search engine as it is considered its core. To index the web is quite a challenging task. Firstly, it requires huge resources and takes days to complete. Secondly, periodic crawling and rebuilding of the index is necessary because the contents of the web change rapidly and most incremental indexing-techniques perform poorly with huge changes. Finally, storage formats for the index must be carefully designed, for example, a compressed index may improve query performance, however, there is a tradeoff between this performance gain and the decompression overhead at query time.

The basic stages for indexing are as follows:

1- Extracting all the words from each document.
2- Removing stop words (e.g. an, and, by, for, of, the, etc...).

3- Normalizing words.

4- Eliminating very high and very low frequency terms.

5- Assigning a term weight using statistics.

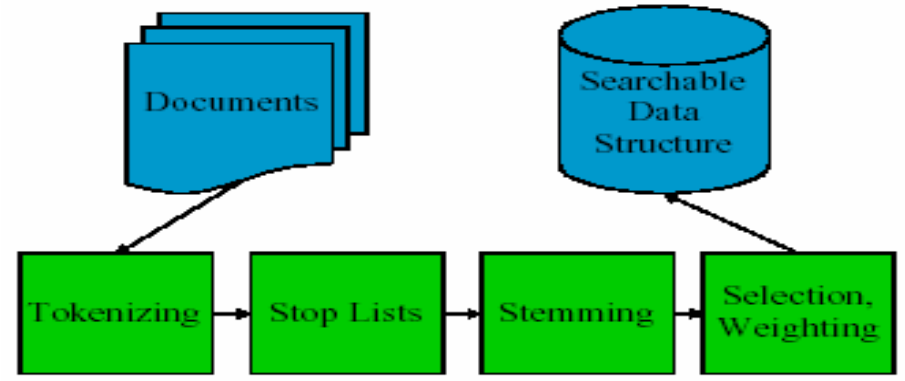

Figure 1.2: The basic stages of the indexer.

There are several types of indexes [11]. A link index represents a graph for the crawled portion of the web, with nodes (pages) and edges (hypertext links). Text index, however, is a lookup for identifying pages relevant to a query. The utility index provides access to pages of a given length, pages of a certain importance, or pages with some number of images in them.

Text index may be one of the following: inverted files, suffix arrays, and signature files. The inverted indexes are the most commonly used in indexing web pages. The inverted index structure is defined as follows:

- An inverted index over a collection of web pages consists of a set of inverted lists, one for each word (or index term).
- The inverted list for a term is a sorted list of locations where it appears in the collection.
- The posting is a pair of an index term *w*, and a corresponding location, *l*.
- Most text-indexes maintain a lexicon (a list of all the terms in the index with some term-level statistics).

BTrees, first presented in [13], are data structures that are commonly used in indexing large amount of data items. All leaves in a BTree are maintained on the same level. A very large number of items can be stored in a BTree of a small height. Furthermore, the maximum height of a BTree index determines the maximum number of accesses for a search, so the search time will be minimized if BTrees are used in indexing a very large amount of documents. It is argued that BTrees are ideal for indexing web pages [14], [15].

1.1.3 The Page Repository

The page repository is defined as follows: A cache of the visited pages that is maintained by a search engine beyond the time required to build the index. This cache allows serving out result pages quickly. It performs two basic functions: First, it is an interface for the crawler to store pages. Second, it provides an efficient access for the indexing module to retrieve the pages.

The challenges for page repository are:

1- Scalability: It must be distributed across a cluster of computers because of the huge size of the web.
2- Dual access modes: It must support two different access modes: Random access is used to quickly retrieve a specific page to the end user. Streaming access is used to receive the entire collection to the indexing module.
3- Large bulk updates: It must handle modifications. As new versions of pages are received, the space of old versions is reclaimed. Also conflicts between updating and accessing must be avoided.
4- Obsolete pages: It must have a mechanism for detecting and removing obsolete pages.

There are two policies for page distribution among repository nodes [11]: