ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 4, June 2012

Realizing Peer-to-Peer and Distributed Web Crawler

Anup A Garje, Prof. Bhavesh Patel, Dr. B. B. Meshram

Abstract—The tremendous growth of the World Wide Web has made tools such as search engines and information retrieval systems have become essential. In this dissertation, we propose a fully distributed, peer-to-peer architecture for web crawling. The main goal behind the development of such a system is to provide an alternative but efficient, easily implementable and a decentralized system for crawling, indexing, caching and querying web pages. The main function of a webcrawler is to recursively visit web pages, extract all URLs form the page, parse the page for keywords and visit the extracted URLs recursively. We propose an architecture that can be easily implemeneted on a local (campus) network and which follows a fully distributed, peer-to-peer architecture. The architecture specifications, implementation details, requirements to be met and analysis of such a system is discussed.

Index Terms— Peer to peer, Distributed,Crawling,indexing

I. Introduction

Web crawlers download large quantities of data and browse documents by passing from one hypertext link to another. A web crawler (also known as a web spider or web robot) is a program or automated script which browses the World Wide Web in a methodical, automated manner. This process is called web crawling or spidering. Many sites, in particular search engines, use spidering as a means of providing up-to-date data.

Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a website, such as checking links or validating HTML code. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for spam). A web crawler is one type of bot, or software agent. In general, it starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier (usually implemented as a queue). URLs from the frontier are recursively visited according to a set of policies.

We present several issues to take into account when crawling the Web. They lead to the fact that at design time of a crawl, its intention needs to be fixed. The intention is defined by the goal that a specific crawl is targeted at; this can differ in terms of crawl length, crawl intervals, crawl scope, etc. A major issue is that the Web is not static, but rather dynamic and thus changes on the timescale of days, hours, minutes. There are billions of documents available on the Web and crawling all data and furthermore maintaining a good freshness of the data becomes almost impossible. To always keep the crawled data up-to-date we would need to continuously crawl the Web, revisiting all pages we have once crawled. Whether we want to do this depends on the earlier mentioned crawling intention. Such an intention can for example be that we want to cover a preferably big part of the Web, crawl the Web for news on one topic or monitor one specific Web site for changes.

We discuss two different crawling strategies that are related to the purpose of a crawl: incremental and snapshot crawling. The strategies can be identified by different frontier growth behavior.

In a snapshot strategy the crawler visits a URL only once; if the same URL is discovered again it is considered as duplicate and discarded. Using this strategy the frontier is extended continuously with only new URLs and a crawl can spread quite fast. This strategy is optimal if you want to e.g. cover an either big or specific part of the Web once, or in regular intervals. The incremental crawling strategy is optimal for recurring continuous crawls with a limited scope; when an already visited URL is rediscovered it is not rejected but instead put into the frontier again. Using this strategy the frontier queues will never empty and a crawl could go on for an indefinite long time. This strategy is optimal for monitoring a specific part of the Web for changes. Following the crawling research field and relevant literature, we distinguish not only between crawling strategies but as well between crawler types. They are nevertheless related as different crawling strategies are used for different crawler types, which thus are related to the specific intentions we pursue when crawling the Web. While the crawling strategies are defined using the frontier growth behavior, the crawler types are based upon the scope of a crawl. They include types as broad, focused, topical or continuous crawling.

The two most important types of web crawling are; **broad and focused crawling**. Broad (or universal) crawls can be described as large crawls with a high bandwidth usage where the crawler fetches a large number of Web sites and goes as well into a high depth on each crawled site. This crawl type fits to the intention of crawling a large part of the Web, if not even the whole Web. Not only the amount of collected Web data is important, but as well the completeness

Anup A. Garje, Department of Computer Technology, Veetmata Jijabai Technilogical Institute,Matunga, Mumbai.India
anupg.007@gmail.com
Prof. Bhavesh Patel Department of Computer Technology, Veetmata Jijabai Technilogical Institute,Matunga, Mumbai. India
bh_patelin@yahoo.co.in

Dr. B. B. Meshram Head of Dept. of Computer Technology, Veermata Jijabai Technological Institute,Matunga Mumbai. India
bbmeshram@vjti.org.in

of coverage of single Web sites. Focused (or topical) crawls on the other side are characterized by the fact that a number of criteria are defined that limits the scope of a crawl (e.g. by limiting the URLs to be visited to certain domains); the crawler fetches similar pages topic-wise. This crawl type is used with the intention to collect pages from a specific domain, category, topic or similar.

II. CRAWLING - AN OVERVIEW

In the following section we will introduce both the Web crawler as such and some commonly known crawling strategies that can be applied to them. A Web crawler, also called a robot or spider, is a software program that starts with a set of URIs, fetches the documents (e.g. HTML pages, service descriptions, images, audio files, etc.) available at those URLs, extract the URLs, i.e. links, from the documents fetched in the previous step and start over the process previously described. That is it automatically downloads Web pages and follows links in the pages, this way moving from one Webpage to another.

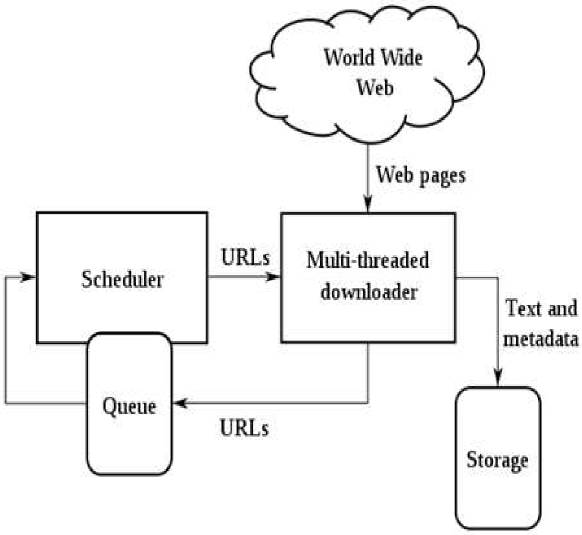

Fig :General architecture of a Web Crawler

We will now shortly describe the basic steps that a crawler is executing. What it basically does is executing different specific steps in a sequential way. The crawler starts by taking a set of seed pages, i.e. the URLs (Uniform Resource Locator) which it starts with. It uses the URLs to build its frontier, i.e. the list (that is a queue) of unvisited URLs of the crawler. In the scope of one crawl this frontier is dynamic as it is extended by the URLs extracted from already visited pages. The edge of a frontier will be limited by the number of URLs found in all downloaded documents (and by politeness restrictions that are followed for different servers). So once a URL is taken from the frontier queue it traverses the following steps:

1. The crawler scheduler checks whether this page is intended to be fetched, i.e. whether there are no rules or policies that exclude this URL.

2. The document the URL points to is fetched by the multithreaded downloader.

3. The crawler extracts links from the downloaded document.

4. Based on given rules the crawler decides whether it wants to permanently store the downloaded documents, index them, generate metadata etc...

5. The crawler feeds the extracted links to the frontier queue.

The above steps are executed for all URLs that are crawled by the Web crawler. Although a crawler has only one frontier, the frontier has multiple queues that are filled with URLs. Queues can be built based on different schemes: e.g. one queue per host. Additionally the queues can be ranked within the frontier which makes then that certain queues are served earlier by the frontier than others. A similar issue is as well the ranking of the URLs within the single queues. During the setup of a crawl, it must be decided what URLs get what priorities and get thus removed either early or late from a queue to be processed further. If a frontier is not set any limit and if the crawler disposes over unlimited hardware resources, it may grow indefinitely. This can be avoided by limiting the growth of the frontier, either by, e.g., restricting the number of pages the crawler may download from a domain, or by restricting the number of overall visited Web sites, what would at the same time limit the scope of the crawl. Whatever frontier strategy is chosen, the crawler proceeds in the same way with the URLs it gets from the frontier.

III. SYSTEM DESIGN AND IMPLEMENTATION

Architectural Details:

Our main goal is to realize a fully distributed, Peer-to-Peer web crawler framework and highlight the features, advantages and credibility of such a system. Our system, named JADE, follows a fully decentralized distributed architecture. A fully decentralized architecture means that there will be no central server or control entity, all the different components are considered to be of equal status (i.e. peers). The system uses an overlay network, which could be a local network for peer-to-peer communication and an underlay network which is the network form which information is crawled and indexed.

The overlay system provides a fully equipped framework for peer-to-peer communication. The basic requirements from such a network are an efficient communication platform, an environment for distributed data management and retrieval, a fault tolerance and self-administering and peer managing network.

Our system manly comprises of peer-entities, which form the atomic units of the system and can be used as standalone or in a network. The following diagram shows the structural components of a single peer-entity;

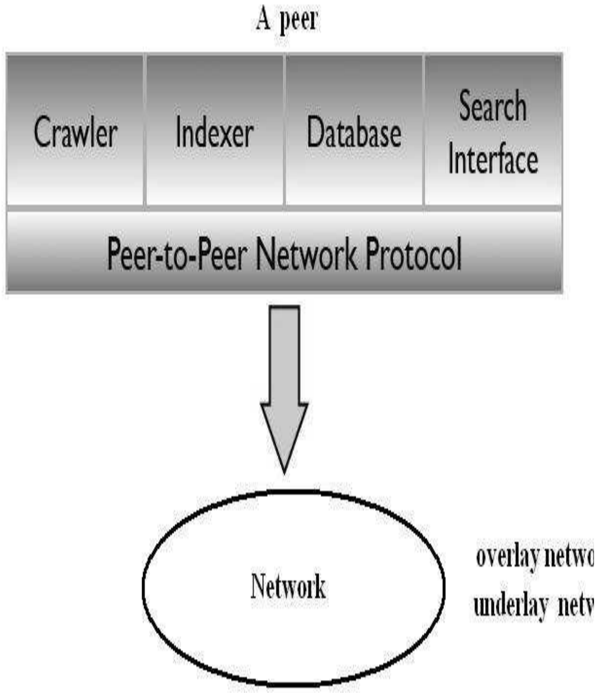

Fig : A peer entity

It consists of following components;

- Crawler
- Indexer
- Database component

If you are using Word, use either the Microsoft Equation Editor or the MathType add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation or MathType Equation). “Float over text” should not be selected.

IV. SYSTEM INTERNALS

THE CRAWLER

As mentioned earlier, the main goal of our system was to implement a fully distributed, P2P web crawler. Traditionally, crawling process consisted of recursively requesting for a webpage, extracting the links from that page, and then requesting the pages from the extracted links. Each page is parsed, indexed for keywords or other parameters and then links fro the page are extracted. The crawler then calls the extracted pages and thus the process continues.

Apart from the above mentioned crawling method, another method exists, known as the proxy method. By using a web proxy, that allows users to access pages from the web through the proxy, we could index and parse the pages that pass through the proxy. Thus only the pages visited by the user will be indexed and parsed. Thus the user unknowingly contributes in the indexing of the pages. The local caching of the visited pages improves the access time of the system. Advanced filtering can also be performed easily on the local cache of visited pages.

Our system runs a large number of processes which operate on data stacks and data queues that are filled during a web crawl and indexing process. The proposed system does a real-time indexing, that means all pages that pass the crawler are instantly searchable (in contrast to batch-processing of other search engine software).

THE INDEXER

The page indexing is done by the creation of a 'reverse word index' (RWI): every page is parsed, the words are extracted and for every word a database table is maintained. The database tables are held in a file-based hash-table, so accessing a word index is extremely fast, resulting in an extremely fast search. The RWIs are hashed form in the database which leads to that the information not stored in plaintext and therefore the security of the index holder rises, since it is not possible to conclude who created the data. At the end of every crawling procedure the index is distributed over the peers participating in the P2P network. Only index entries in the form of URLs are stored, no other caching is performed. Jade implements its index structure as a distributed hash table (DHT): a hash function is applied to every index entry; the entry is then distributed to the appropriate peer. The resulting index contains no information about the origin of the keywords stored in it. Moreover, shared filters offer customized protection against undesirable content.

THE DATABASE

The database stores all indexed data provided by the indexer and the P2P, which is added to the network. They are each peer the data to fit his DHT and through the assigned migration index of the directory. The structure of the database will be balanced, binary search tree (AVL tree) formed to a logarithmic search time on the number of elements in the tree. AVL, The name derives from the inventors and Adelson-Velsky Landis by whom this data structure for balanced data distribution was developed in 1962. The AVL property ensures maximum performance in terms of algorithmic order.

Tree nodes can be dynamically allocated and de-allocated and an unused-node list is maintained. For the PLASMA search algorithm, an ordered access to search results are necessary, therefore we needed an indexing mechanism which stores the index in an ordered way. The database supports such access, and the resulting database tables are stored as a single file. It is completely self-organizing and does not need any set-up or maintenance tasks that must be done by an administrator. Any database may grow to an unthinkable number of records: with one billion records a database request needs a theoretical maximum number of only 44 comparisons.

We have implemented Kelondro database subsystem for realizing the above mentioned requirements and features. The Kelondro database, is an open source AVL based database structure, which provides all he necessary schema, functions and methods for inserting, querying, modifying a AVL tree based database.

V. INFORMATION FLOW

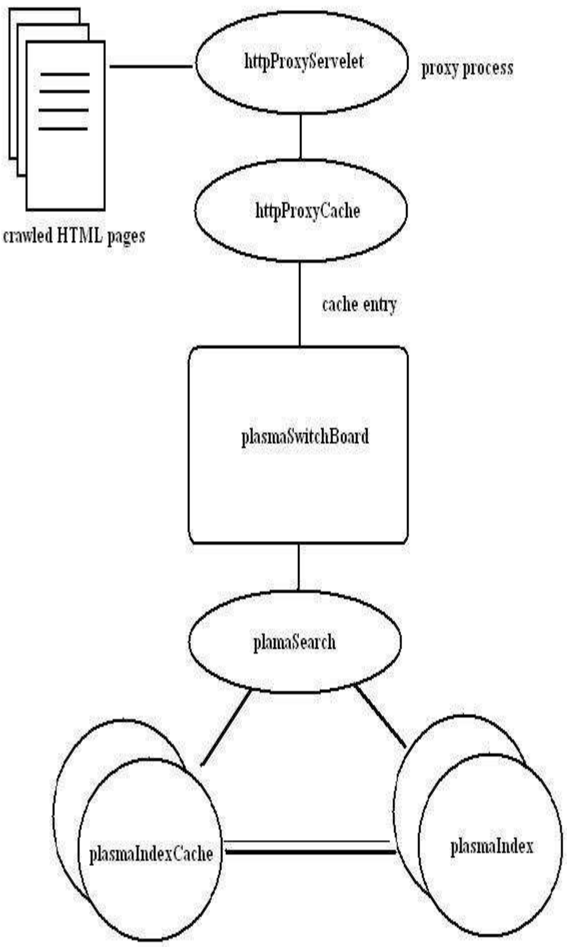

Fig : Crawler Information Flow Diagram

The above diagram shows the flow of information in the crawler system. The HTML file from a crawled URL is loaded onto the httpProxyServelet module. This module is the proxy process that runs in the background.

The file is then transferred to htpProxyCache module, which provides the proxy cache and where the processing of the file is delayed until the proxy is idle. The cache entry is passed on to the plasmaSwitchboard module. This is the core module that forms the central part of the system..

There the URL is stored into plasmaLURL where the URL is stored under a specific hash. The URL's from the content are stripped off, stored in plasmaLURL with a 'wrong' date (the date of the URL's are not known at this time, only after fetching) and stacked with plasmaCrawlerTextStack.

The content is read and splitted into rated words in plasmaCondenser. The splitted words are then integrated into the index with plasmaSearch. In plasmaSearch the words are indexed by reversing the relation between URL and words: one URL points to many words, the words within the document at the URL. After reversing, one word points to many URL's, all the URL's where the word occurrs. One single word->URL-hash relation is stored in plasmaIndexEntry. A set of plasmaIndexEntries is a reverse word index. This reverse word index is stored temporarily in plasmaIndexCache.

In plasmaIndexCache the single plasmaIndexEntry'ies are collected and stored into a plasmaIndex – entry. These plasmaIndex - Objects are the true reverse words indexes. In plasmaIndex the plasmaIndexEntry - objects are stored in a kelondroTree; an indexed file in the file system.

8 SEARCH/ QUERY FLOW

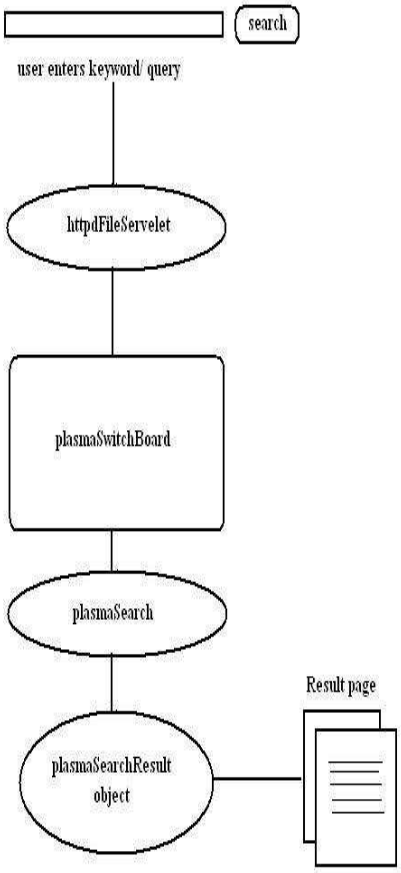

Fig: The Search/ Query Information Flow Diagram.

The above diagram shows the flow of a user query or a keyword search in the system.

The keyword or the query entered by the user is passed onto the httpdFileServelet process, which accepts the information and passes it to the plasmaSwitchBoard module.

The query is validated and checked for consistency before passing it to plasmaSearch module, which is the search function on the index. In plasmaSearch, the plasmaSearchResult object is generated by simultaneous enumeration of URL hashes in the reverse word indexes plasmaIndex. The result page is then generated from this plasmaSearchResult object

VI. SYSTEM ANALYSIS

In this section we shall discuss certain security aspects, software structure, advantages, disadvantages and future scope of the proposed system.

SECURITY & PRIVACY ASPECTS

The system largely

Sharing the index to other users may arise privacy concerns. The following properties were decide and implemented to take care of security & privacy concerns;

Private Index and Index Movement The local word index does not only contain information that a peer created by

	surfing the internet, but also entries from other peers. Word index files travel along the proxy peers to form a distributed hash table. Therefore nobody can argue that information that is provided by this peer was also retrieved by this peer and therefore by the peer's personal use of the internet. In fact it is very unlikely that information that can be found on a peer was created by the peer itself, since the search process targets only peers where it is likely because of the movement of the index to form the distributed hash table. During a test phase, all word indexes on a peer will be accessible. The future production release will constraint searches to indexes entries on the peer that have been created by other peers, which will ensure complete browsing privacy.
Word Index Storage and Content Responsibility	The words that are stored in the client's local word index are stored using a word hash. That means that not any word is stored, but only the word hash. You cannot find any word that is indexed as clear text. You can also not re-translate the word hashes into the original word. This means that you don't know actually which words are stored in your system. The positive effect is, that you cannot be responsible for the words that are stored in your peer. But if you want to deny storage of specific words, you can put them into the 'bluelist' (in the file httpProxy.bluelist). No word that is in the bluelist can be stored, searched or even viewed through the proxy.
Peer Communication Encryption	Information that is passed from one peer to another is encoded. That means that no information like search words, indexed URL's or URL descriptions is transported in clear text. Network sniffers cannot see the content that is exchanged. We also implemented an encryption method, where a temporary key, created by the requesting peer is used to encrypt the response (not yet active in test release, but non-ascii/base64 - encoding is in place).
Access Restrictions	The proxy contains a two-stage access control: IP filter check and an account/password gateway that can be configured to access the proxy. The default setting denies access to your proxy from the internet, but allowes usage from the intranet. The proxy and it's security settings can be configured using the built-in web server for service pages; the access to this service pages itself can also be restricted again by using an IP filter and an account/password combination.

VII. Conclusion

Thus, one can note that as the size and usage of the WWW increases, the use of a good Information retrieval system comprising of indexers and crawlers becomes trivial. The current web information retrieval systems provide too much of censorship and restrictive policies. There is a need for a distributed, free and a collective view of the task of information retrieval and web page indexing and caching. The system proposed aims to provide these views and design goals.

The use of a censorship-free policy avoids all the restrictions provided by current systems and enables full coverage of the WWW as well as of "hidden web" or the "deep web". This is not possible using existing systems. Also the use of Distributed Hash Tables (DHTs) and key based routing provides a solid framework for a distributed peer-to-peer network architecture. The proxy provides an added functionality of caching web pages visited by the user, which is performed in the background.

We believe that the system proposed by us will prove to be a complete implementation of a fully distributed, peer-to-peer architecture for web crawling, to be used on small networks or campus networks. We believe that we have clearly stated the requirements, implementation details and usage advantages regarding such a system and highlighted its purpose.

VIII. References

1. Kobayashi, M. and Takeda, K. (2000). "Information retrieval on the web". ACM Computing Surveys (ACM Press) 32 (2): 144–173.
2. Boldi, Paolo; Bruno Codenotti, Massimo Santini, Sebastiano Vigna (2004). "UbiCrawler: a scalable fully distributed Web crawler". Software: Practice and Experience.
3. Heydon, Allan; Najork, Marc (1999-06-26) Mercator: A Scalable, Extensible Web Crawler. http://www.cindoc.csic.es/cybermetrics/pdf/68.pdf.
4. Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine.
5. Zeinalipour-Yazti, D. and Dikaiakos, M. D. (2002). Design and implementation of a distributed 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. Springer.
6. Ali Ghodsi. Distributed k-ary System: Algorithms for Distributed Hash Tables. KTH-Royal Institute of Technology, 2006.
7. Goyal, Vikram (2003), Using the Jakarta Commons, Part I, http://www.onjava.com/pub/a/onjava/2003/06/25/commons.html