Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 2, Issue. 6, June 2013, pg.244 – 250

RESEARCH ARTICLE

A Novel Architecture of Mercator: A Scalable, Extensible Web Crawler with Focused Web Crawler

Sarnam Singh[1], Nidhi Tyagi[2]

[1]M.Tech, School of Computer Engineering & I.T., Shobhit University (Shobhit Institute of Engineering & Technology), (Deemed – to-be University), Meerut, U.P, India

[2]Assistant Professor, School of Computer Engineering & I.T., Shobhit University (Shobhit Institute of Engineering & Technology), (Deemed – to-be University), Meerut , U.P, India

[1] *sarnammmb@gmail.com, sarnamsingh@yahoo.co.in*

Abstract— This Paper described A Novel Architecture of Mercator: A Scalable, Extensible Web Crawler with Focused Web Crawler. We enumerate the major components of any Scalable and Focused Web Crawler and describe the particular components used in this Novel Architecture. We also describe this Novel Architecture support for Extensibility and downloaded user's support information. We also describe how the Focused Web Crawler component integrates with Mercator: A Scalable, Extensible Web Crawler and also describe their functionality of every component and how to work together. We also describe how this Novel Architecture downloaded maximum pages from web in minimum time and sure partially extract web pages which is needed to users.

I. INTRODUCTION

A Web crawler is an Internet bot that systematically browses the World Wide Web, typically for the purpose of Web indexing.

A Web crawler may also be called a Web spider [12], an ant, an automatic indexer, or (in the FOAF software context) a Web scutter [13].

The World Wide Web having over Million Pages and continues to grow rapidly so it is difficult to find exact user's needed information so we proposed A Novel Architecture of Mercator :- A Scalable , Extensible Web Crawler using focused web Crawler.

Currently general purpose search engines retrieve and download unwanted information therefore network load is increases so our proposed Architecture first take query from users then Extract important word and eliminates bibliography word then check in database if it is not found in database then find out similar keywords from world dictionary and make a world database. This word database connect to the focused web crawler and focused web crawler focus the exact world and their URLs this URLs store in URL database and connect to the Mercator :- A Scalable , Extensible Web Crawler then this web Crawler download the exact information which is needed to users.

1.1 Mercator :- A Scalable , Extensible Web Crawler

Mercator:- A Scalable , Extensible Web Crawler is already exit web crawler which is parallel down load pages like (.pdf, .txt, .doc, .html, .jpeg, etc) this parallel download these files.

Scalable

This Web Crawler Architecture is designed to scale up to entire web, and has been used to fetch of millions of web documents. Hence the vast majority of our data structures are stored on disk, and small parts of them are stored in memory for efficiency.

Extensible

This Web Crawler Architecture is designed in modular way, with the expectation that new functionality will be added by third parties.

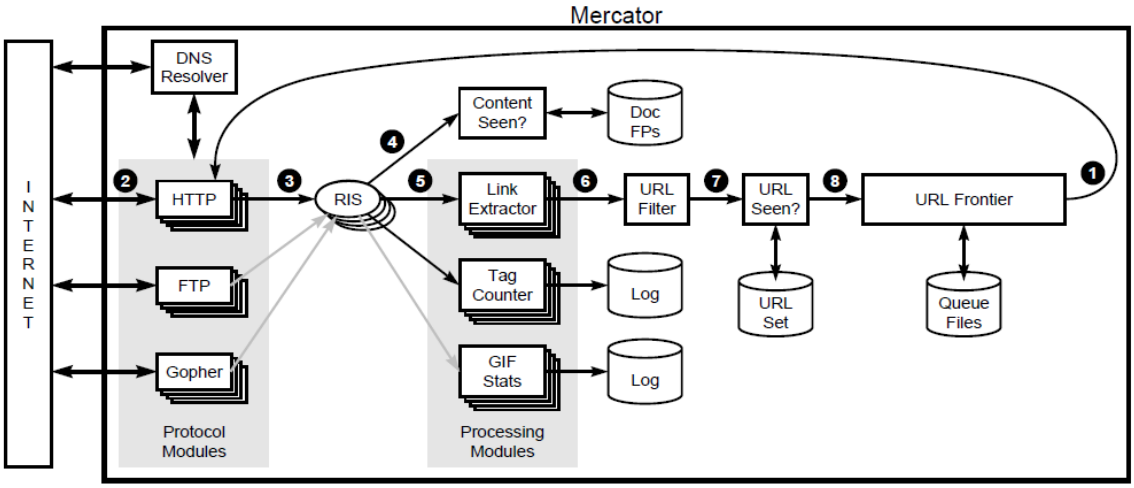


Figure 1: Mercator's main components.

1.2 Focused Web Crawler

Focused Crawler which is seeks, acquires, indexes, and maintains pages on a specific set of topics that represent a relatively narrow segment of the Web.

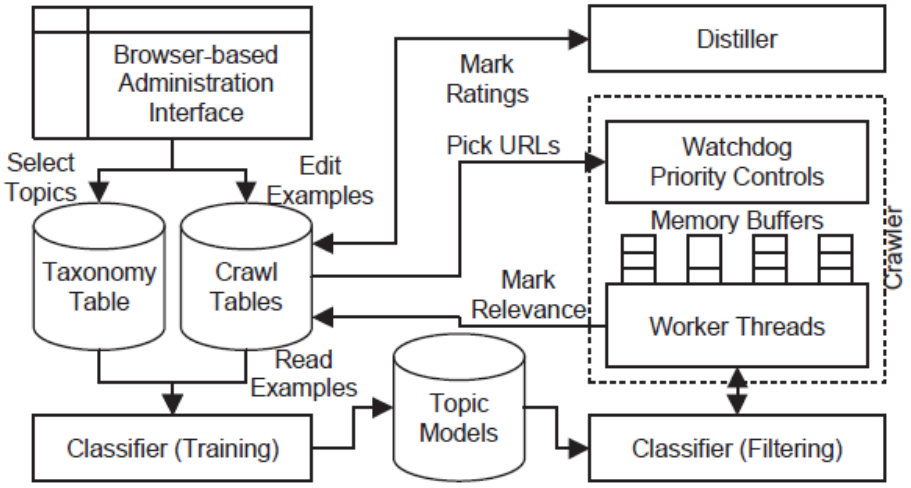


Fig. 2. Block diagram of the focused crawler showing how the crawler, classifier and distiller are integrated.

1.3 Proposed System Architecture

Our proposed Architecture linked with world dictionary for find out similar meaningful world and these word store in word database then this word's database word one by one access by Focused Crawler then create URL database and these URL documents downloaded by Mercator :- A Scalable , Extensible Web Crawler.

II. PROBLEM IDENTIFICATION

In the Existing System Mercator download relevant pages which is related to the user's query and Focused Crawler download more relevant pages which is closely related to the user's query.

Mercator Web Crawler can't download more relevant pages and Focused Web Crawler can't download .pdf files, .doc files, .text files etc in parallel.

Our A Novel Architecture of Mercator: A Scalable, Extensible Web Crawler with Focused Web Crawler download .pdf files, .text files, .doc files, .html files, .xml files etc in parallel and close related to user's query.

III. A NOVEL ARCHITECTURE OF MERCATOR :- A SCALABLE , EXTENSIBLE WEB CRAWLER USING FOCUSED WEB CRAWLER ADMINISTRATION

The user query may be single word or collection of word or may be URL therefore our proposed Architecture have important component called Query Identifier which Identify the type of query.

3.1 Component

A. Query Identifier
B. Word's Filter
C. Query Word Database
D. Words Dictionary & Symmetric Matching
E. Word Database
F. URLs Database
G. Focused Crawler's Component

G.1 Distiller
G.2 Classifier

H. Mercator's Component

H.1 URL Frontier
H.2 URL Seen
H.3 URL Set
H.4 URL Filter
H.4 Link Extractor
H.5 Content Seen
H.6 RIS (Rewind Input Stream)
H.7 DNS

Query identifier

This Component identifies which type query is fired by user that is (single word, set of words, & URL).

Words Filter

This component extract bibliographic information and store reaming words in a database called Query words database

Query Words Database

Store words which collect from User's Query. This database simply implemented in M.S. Accesses Database and Queue Data Structure because in the Queue Data

Structure we accesses word from Front and same time word insert through Rear.

Other thing in this M.S. Database we catch redundancy of word through key technique.

Words Dictionary

Our Crawler search out more relevant words from words dictionary using any one Matching technique

Word Database

In Word Database stores the collection of words which is search out from words Dictionary. The word Database simply way in M.S. Access Database in Queue Data Structure.

URLs Database

In the URLs Database stores the collection of URLs which find by our web Crawler and Distiller component of Focused Web Crawler. Here also we use M.S. Accesses Database.

Focused Crawler's Component

I Distiller [1]

Distiller is used to identify pages with large numbers of links to relevant pages

II Classifier [1]

The classifier evaluates the relevance of a hypertext document with respect to the focus topics.

Mercator's Component

URL Frontier[2]

The URL frontier is the data structure that contains all the URLs that remain to be downloaded. Most crawlers work by performing a breath-first traversal of the web, starting from the pages in the seed set. Such traversals are easily implemented by using a FIFO queue.

In a standard FIFO queue, elements are de queued in the order they were en queued. In the context of web crawling, however, matters are complicated by the fact that it is considered socially unacceptable to have multiple HTTP requests pending to same server. If multiple requests are to be made in parallel, the queue's remove operation should not simply return the head of the queue, but rather a URL close to the head whose host has no outstanding request.

To implement this politeness constraint, the default version of Mercator's URL frontier is actually implemented by a collection of distinct FIFO sub queues. There are two important aspects to how URLs are added to and removed from these queues. First, there is one FIFO sub queue per worker thread.

That is, each worker thread removes URLs from exactly one of the FIFO sub queues. Second, when a new URL is added, the FIFO sub queue in which it is placed is determined by the URL's canonical host name. Together, these two points imply that at most one worker thread will download documents from given web server.

In actual World Wide Web crawls, the size of the crawl's frontier numbers in the hundreds of millions of URLs. Hence, the majority of the URLs must be stored on disk.

URL Seen [2]

In the course of extracting links, any web crawler will encounter multiple links to the same document.

To avoid downloading and processing a document multiple times, a URL-seen test must be performed on each extracted link before adding it to the URL frontier.

URL Filter [2]

The URL filtering mechanism provides a customizable way to control the set of URLs that are downloaded. Before adding a URL to the frontier, the worker thread 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 to not crawl that URL. Mercator includes a collection of different URL filter subclasses that provide facilities for restricting URLs by domain, prefix, or protocol type, and for computing the conjunction, disjunction, or negation of other filters. Users may also supply their own custom URL filters.

Link Extractor[2]

Link Extractor extract the link from each download page.

Content Seen [2]

Many documents on the web are available under multiple, different URLs. There are also many cases in which documents are mirrored on multiple servers. Both of these effects will cause any web crawler to download the same document contents multiple times. To prevent processing a document more than once, a web crawler may wish to perform a content-seen test to decide if the document has already been processed. Using a content-seen test makes it possible to suppress link extraction from mirrored pages, which may result in a significant reduction in the number of pages that need to be downloaded. Mercator includes just such a content-seen test, which also offers the side benefit of allowing us to keep statistics about the fraction of downloaded documents that are duplicates of pages that have already been downloaded. The content-seen test would be prohibitively expensive in both space and time if we saved the complete contents of every downloaded document. Instead, we maintain a data structure called the document fingerprint set that stores a 64-bit checksum of the contents of each downloaded document. We compute the checksum using Border's implementation of Rabin's fingerprinting algorithm. Fingerprints offer provably strong probabilistic guarantees that two different strings will not have the same fingerprint. Other checksum algorithms, such as MD5 and SHA, do not offer such provable guarantees, and are also more expensive to compute than fingerprints. When the crawling the entire web, the document fingerprint set will obviously be too large to be stored entirely in memory.

Unfortunately, there is very little locality in the requests made on the document fingerprint set, so caching such requests has little benefit. We therefore maintain two independent sets of fingerprints: a small hash table kept in memory, and a large sorted list kept in a single disk file. The content-seen test first checks if the fingerprint is contained in the in-memory table. If not, it has to check if the fingerprint resides in the disk file. To avoid multiple the disk seeks and reads per disk search, Mercator performs an interpolated binary search of an in-memory index of the disk file to identify the disk block on which the fingerprint would reside if it were present. It then searches the appropriate disk block, again using interpolated binary search. We use a buffered variant of Java's random access files, which guarantees that searching through one disk block causes at most two kernel calls (one seek and one read). We use a customized data structure instead of a more generic data structure such as a B-tree because of this guarantee. It is worth noting that Mercator's ability to be dynamically configured would easily allow someone to replace our implementation with a different one based on B-trees. When a new fingerprint is added to the document fingerprint set, it is added to the in-memory table. When this table fills up, its contents are merged with the fingerprints on disk, at which time the in memory index of the disk file is updated as well. To guard against races, we use a readers-writer lock that controls access to the disk file. Threads must hold a read share of the lock while reading from the file, and must hold the write lock while writing to it.

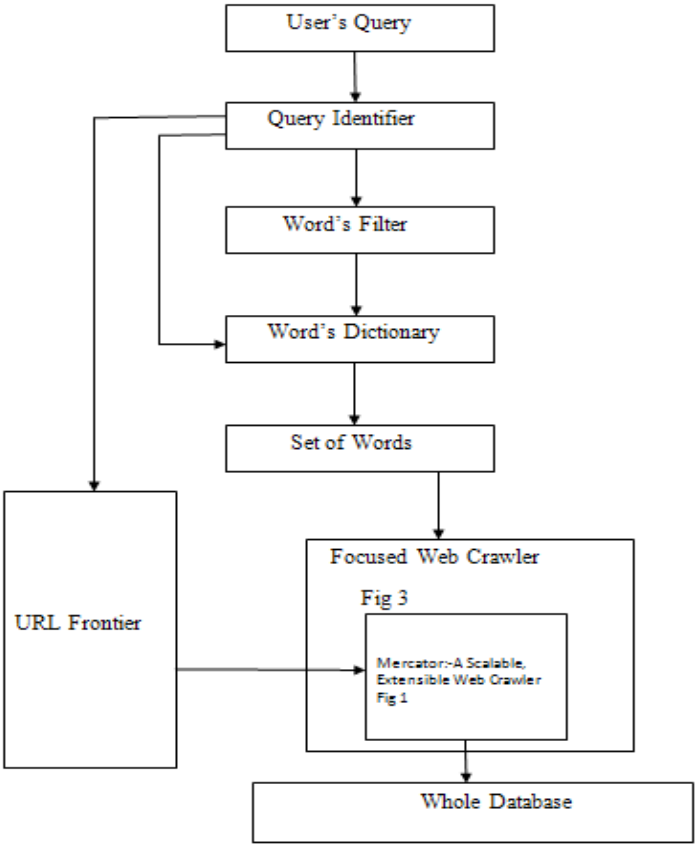
RIS (Rewind Input Stream) [2]

Mercator's design allows the same document to be processed by multiple processing modules. To avoid reading a document over the network multiple times, we cache the document locally using an abstraction called a Rewind Input Stream (RIS). A RIS is an input stream with an open method that reads and caches the entire contents of a supplied input stream (such as the input stream associated with a socket). A RIS caches small documents (64 KB or less) entirely in memory, while larger documents are temporarily written to a backing file. The RIS constructor allows a client to specify an upper limit on the size of the backing file as a safeguard against malicious web servers that might return documents of unbounded size. By default, Mercator sets this limit to 1 MB. In addition to the functionality provided by normal input streams, a RIS also provides a method for rewinding its position to the beginning of the stream, and various flexing methods that make it easy to build MIME-type-specific parsers. Each worker thread has an associated RIS, which it reuses from document to document. After removing a URL from the frontier, a worker passes that URL to the appropriate protocol module, which initializes the RIS from a network connection to contain the document's contents. The worker then passes the RIS to all relevant processing modules, rewinding the stream before each module is invoked.

DNS (Domain Name Service) [2]

Before contacting a web server, a web crawler must use the Domain Name Service (DNS) to map the web server's host name into an IP address. DNS name resolution is a well-documented bottleneck of most web crawlers. This bottleneck is exacerbated in any crawler, like Mercator or the Internet Archive crawler that uses DNS to cannibalize the host names of newly discovered URLs before performing the URL-seen test on them.

IV. PROPOSED SYSTEM ARCHITECTURE



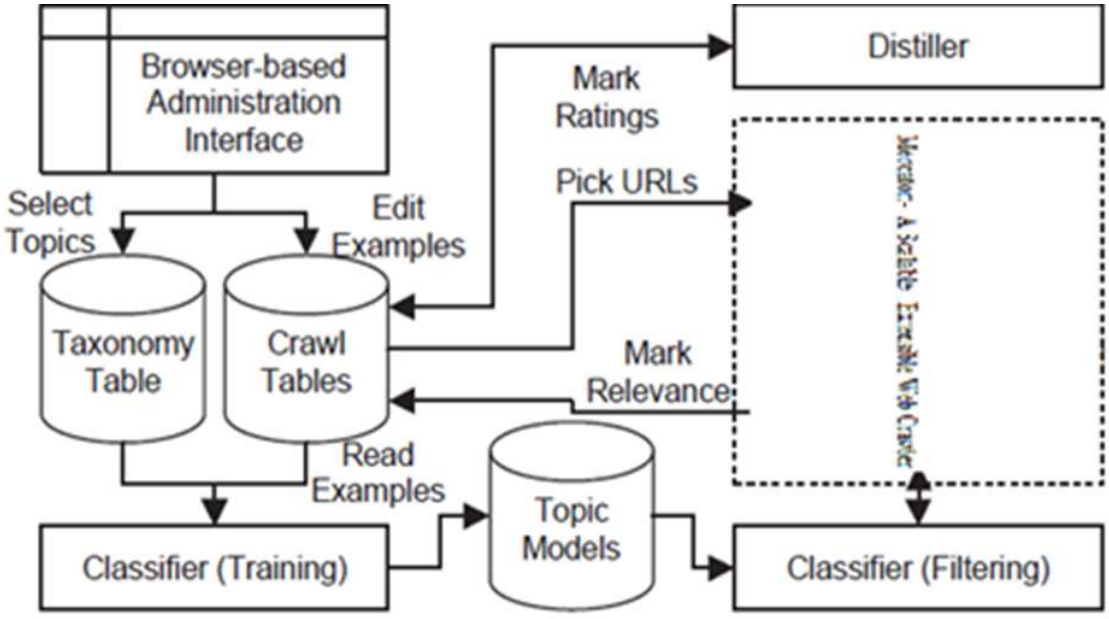


Fig 3 Focused Crawler showing how Mercator: A Scalable, Extensible web Crawler, Classifier, Distiller are integrated

V. CONCLUSION

The Proposed System Architecture download the user's query closely relevant documents and the .text, .pdf, etc pages download parallel. The Proposed system Accesses word dictionary and search out the synonyms of the words and download their relevant pages also. For Example User's query is single keywords Mouse then the Proposed system find their synonyms by word dictionary find out two meaning of Mouse one is Rat and other is Computer Mouse then our system only these two word's relevant pages download their text document and image pages parallel.

REFERENCES

[1] Soumen Chakrabarti, Martin van den Ber, Byron Dom, Focused crawling: A new Approach to topic specific Web resource discovery in ELSEVIER Publications,1999
[2] Allan Heydon and Marc Najork Mercator: A Scalable, Extensible Web Crawler.
[3] Ram Kumar Rana, IIMT Institute of Engg. & Technology, Meerut, India, Nidhi Tyagi , Shobhit University, Meerut, India, A Novel Architecture of Ontology-based Semantic Web Crawler, International Journal of Computer Applications (0975 – 8887), April 2012, Volume 44– No18
[4] Abhinna Agarwal, Durgesh Singh, Anubhav Kedia Akash Pandey, Vikas Goel, Design of a Parallel Migrating Web Crawler, International Journal of Advanced Research in Computer Science and Software Engineering , April 2012 Volume 2, Issue 4.
[5] Priyanka-Saxena Department of Computer Science Engineering, Shobhit University, Meerut, Uttar Pradesh-250001, India, Mercator as a web crawler IJCSI International Journal of Computer Science Issues, , January 2012, Vol. 9, Issue 1, No 1
[6] MARC NAJORK Microsoft Research, Mountain View, CA, USA, Web Crawler Architecture
[7] Sameendra Samarawickrama, Lakshman Jayaratne FOCUSED WEB CRAWLING USING NAMED ENTITY RECOGNITION FOR NARROW DOMAINS.
[8] Sotiris Batsakis, Euripides G.M. Petrakis, Evangelos Milios , Improving the Performance of Focused Web Crawlers.
[9] Hidetsugu Nanb, Takeshi Abekawa, Manabu Okumura, Suguru Saito, Bilingual PRESRI Integration of Multiple Research Paper Databases
[10] http://en.wikipedia.org/wiki/Web_crawler
[11] http://en.wikipedia.org/wiki/
[12] Spetka, Scott. "The TkWWW Robot: Beyond Browsing". NCSA. Archived from the original on 3 September 2004. Retrieved 21 November 2010.
[13] http://wiki.foaf-project.org/w/Scutter

