A Migrating Parallel Exponential Crawling Approach to Search Engine

Jitendra Kumar Seth
Assistant Professor, Department of Information Technology, Ajay Kumar Garg Engg. College, Ghaziabad, India, mrjkseth@yahoo.co.in

Ashutosh Dixit
Senior Lecturer in Computer Science Department, YMCA, Faridabad, Hariyana, India
dixit_ashutosh@rediffmail.com

Abstract. Search engines have become important tools for Web navigation. In order to provide powerful search facilities, search engines maintain comprehensive indices of documents available on the Web. The creation and maintenance of Web indices is done by Web crawlers, which recursively traverse and download Web pages on behalf of search engines. Analysis of the collected information is performed after the data has been downloaded. In this research, we propose an alternative, more efficient approach to building parallel Web crawlers based on mobile crawlers. Our proposed crawlers are transferred to the remote machines where they downloads the web pages and make some other processing in order to filter out any unwanted data locally before transferring it back to the search engine(central machine). This reduces network load and speeds up the indexing phase inside the search engine. In our approach design and implementation of web crawler that can grow parallelism to the desired depth and can download the web pages that can grow exponentially in size is being proposed. The parallel crawler first filters and then compresses the downloaded web pages locally before transmitting it to the central machine. Thus the crawlers save the bandwidth of network and take the full advantage of parallel crawling for downloading and speedup the process.

Keywords: Quality of index, seaach engine, web crawlers, parallel crawlers, mobile crawlers, communication bandwidth, Crawl machine, CDB, Crawler Application, filter, and compress.

1. Introduction

The World Wide Web ("WWW" or simply the "Web") is a global information medium which users can read and write via computers connected to the Internet. It is not easy to find your web pages among 1 billion web pages currently published online. The size of the Web has doubled in less than two years, and this growth rate is projected to continue for the next two years. In the context of Internet for useful information, a search engine is a program, or series of programs that, scan and index the web pages on the internet. A crawler is a program that retrieves and stores pages from the Web, commonly for a Web search engine. A crawler often has to download hundreds of millions of pages in a short period of time and has to constantly monitor and refresh the downloaded pages. Roughly, a crawler starts off by placing an initial set of URLs, in a queue, where all URLs to be retrieved are kept and prioritized. From this queue, the crawler gets a URL (in some order), downloads the page, extracts any URLs in the downloaded page, and puts the new URLs in the queue. This process is repeated. Collected pages are later used for other applications, such as a Web search engine.

Given this explosive growth, we see the following specific problems with the way current search engines index the Web:

Scaling: The concept of "download-first-and-index-later" will likely not scale given the limitations in the infrastructure and projected growth rate of the Web. Using the estimates for growth of Web indices provided in [1], a Web crawler running in the year 2000 would have to retrieve Web data at a rate of 45Mbit per second in order to download the estimated 480GB of pages per day that are necessary to maintain the index. Looking at the fundamental limitations of storage technology and communication networks, it is highly unlikely that Web indices of this size can be maintained efficiently.

Efficiency:$_3$ Current search engines add unnecessary traffic to the already overloaded Internet. While current approaches are the only alternative for general-purpose search engines trying to build a comprehensive Web index, there are many scenarios where it is more efficient to download and index only selected pages.

Quality$_3$ of Index: The results of Web searches are overwhelming and require the user to act as part of the query processor. Current commercial search engines maintain Web indices of up to 110 million pages [1] and easily find several thousands of matches for an average query. Thus increasing the size of the Web index does not automatically improve the quality of the search results if it simply causes the search engine to return twice as many matches to a query as before.

Since we cannot limit the number of pages on the Web, we have to find ways to improve the search results in such a way that can accommodate the rapid growth of the Web.

Therefore, we expect a new generation of specialized search engines to emerge in the near future.

2. Literature Survey

According to Junghoo Cho, Hector Garcia-Molina[2], many search engines often run multiple processes in parallel to perform the task of parallel crawling , so that download rate is maximized. In particular, following issues make the study of a parallel crawler challenging and interesting:

Overlap: When multiple processes run in parallel to download pages, it is possible that different processes download the same page multiple times.

Quality: Often, a crawler wants to download "important" pages first, in order to maximize the "quality" of the downloaded collection. However, in a parallel crawler, each process may not be aware of the whole image of the Web that they have collectively downloaded so far.

Communication bandwidth: In order to prevent overlap, or to improve the quality of the downloaded pages, crawling processes need to periodically communicate to coordinate with each other. However, this communication may grow significantly as the number of crawling processes increases.

According to Jan Fiedler and Joachim Hammer [3], in the mobile based web crawling approach, the mobile crawler move to data source before the actual crawling process is started. The use of mobile crawlers for information retrieval requires an architecture which allows us to execute code (i.e. crawlers) on remote systems.

A critical look at the available literature indicates the following issues to be addressed towards design of an efficient crawler.

- The traditional parallel web crawlers download the web pages on a single machine which causes bottleneck at the network level.

- The traditional migrating crawlers do not perform any compression and filtering before transmitting the web pages to the central machine. Moreover the migrating web crawlers generally migrates themselves up to a single level of migration depth hence they are unable to take benefit of desired level of migration depth hence reduces the degree of parallelism.

3. A migrating parallel exponential web crawling approach

In this work the concept of download first transmits later is being proposed by using which data can be locally downloaded, filtered and compressed before transmitting it to the search engine server. Design and implementation of parallel migrating web crawler that can grow parallelism to the desired depth and can download the web pages that can grow exponentially in size is being proposed. The proposed parallel migrating crawler also first filters and then compresses the web pages locally before any transmission of file to the central machine (search engine). Thus, the crawler saves the bandwidth of channel and takes the full advantage of parallel crawling and speedup the process.

Architecture of migrating parallel exponential web crawler: The architecture of the proposed crawler is shown below in figure 1.1. Parallel migrating crawler with exponential growth consists of the two major modules

- Web crawler application
- Crawl machine module

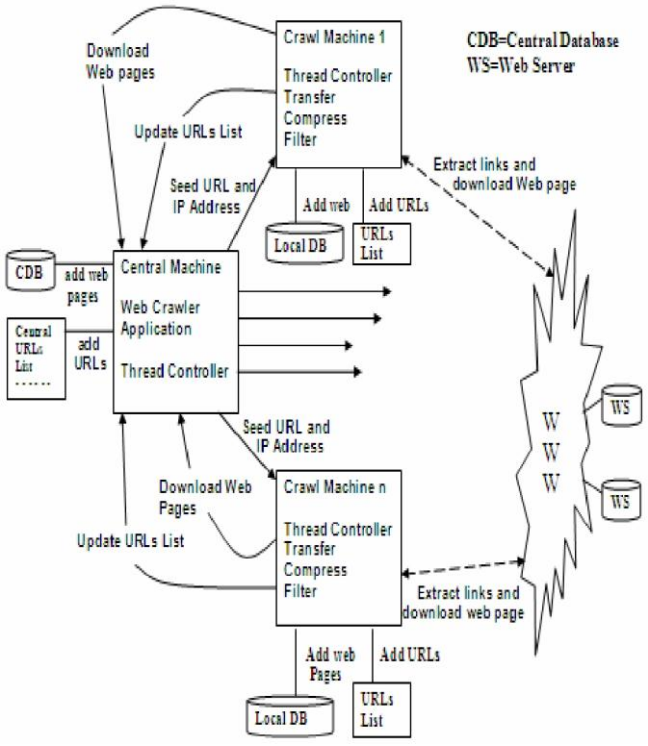

Fig 1.1 Architecture of Parallel migrating crawler with exponential growth

In continuation of the figure 1.1 the next level of crawl machine modules running on n number of machines are shown in figure 1.2.

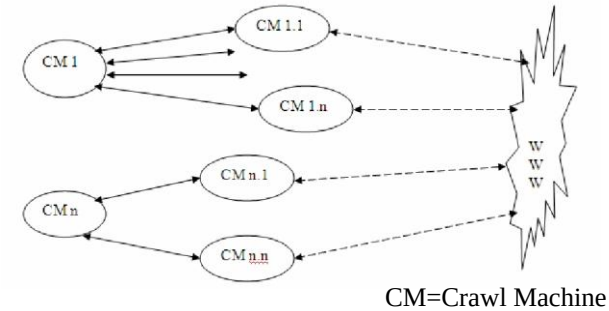

Fig 1.2 Architecture of Parallel migrating crawler with exponential growth

Web crawler application (search engine or central machine):

This is the main module that represents the central machine or more generally search engine, prompts the user to enter the seed Urls to be crawl and the IP addresses of the machine to which the crawling process will migrate. It is a multithreaded module that creates the required numbers of thread of crawling processes and controls these processes on remote machines. This module dynamically updates its central database of downloaded web pages getting from different crawling threads running on remote machines and central Urls list based on current crawling Urls by different crawling processes running on remote machines.

Each crawling process downloads web pages and stores them in its local database that is referred to a directory named crawlerdatabase.The central machine stores downloaded web pages received from different crawling processes(crawl machine module) on remote machines into its central database that is a directory. Each remote machine crawling process also acts as a central machine generates the threads of crawling processes and migrate them on different remote machines and the next level remote machines running the crawling process also acts as a central machine and so on. The level of migration increases until we achieve the satisfactory level of parallelism in crawling which improves the performance of overall crawling.

This module consists of:

- Counter_wca
- URLlistupdate_wca
- pageDownloader_wca
- ThreadController_wca

Counter_wca

The counter module is used to control the depth of crawling. This module initializes a counter variable to the desired number of depth of crawling. Then it sends the counter value to each next level crawl machines. Each crawl machine decrements the counter value by one and forwarded the updated counter value to the next level crawl machines. The next level crawl machine decrements the counter value by one and forwards the updated counter value to the next level crawl machine and so on this process of counter forwarding continues until the counter value reaches to zero. As the next level crawl machine find the counter value zero it stops migrating crawling process to the next level.

```
Algorithm  Counter_wca
Begin
  Initialize a counter variable to
  the desired number of depth of
  crawling; Send the counter value to
  the next level of crawl machines;
End
```

URLlistupdate_wca

This module is responsible for updating the current crawling status of each crawl machine visualizing on the central machine. It also updates the central URL list which is also visualized in a synchronized manner at the central machine. As soon as a new URL is found by any crawl machine it gets the URL and adds the URL to the central URL list.

```
Algorithm  URLlistupdate_wca
Begin
   While  (not  receiving  "done"  from  the
   crawl  machine)
         {
```

```
            Receive URL from any
            crawl machine module;

            Display crawling status;
            Update the corresponding
            crawl machine status at
            the central machine;

            If URL can not resolve then
               Display status and do
               not add URL in central
               URL list;
            Else
               Add URL to the central URL
               List;
        }
End
```

pageDownloader_wca

The crawl machines downloads the web pages from the web and then filter the web pages based on some user choice and then compress the filtered web pages in a zip file and finally transfer the zip file to the central machine. This module is responsible for download the zip files from the crawl machines and store the downloaded zip files in the central local 'crawlerdatabase' directory with synchronized access.

```
Algorithm  pageDownloader_wca
Begin
   While  receiving  zip  files
     {
       Create zip file with the same
       name in crawlerdatabase
       directory as transferring Crawl
       machine's (1…n) zip file name;

       Receive contents of zip file;
       Write contents to the zip file
       ; Display zip file received ;
       Next  zip  file;
      }
  End
```

This module is also implemented as a part of each crawl machine module.

ThreadController_wca

The thread controller module generates a number of threads set by the programmer. Each thread is responsible to migrate the crawling process to the destination machine and also control the data transfer between the WebCrawler applications and crawl machines. The thread controller module also supplies the seed urls to the crawl machines. The URL list update module and page downloader modules are the part of the thread controller. The thread controller module also synchronizes the access to the central crawlerdatabase directory and the

```
Algorithm  ThreadController_wca
Begin
    Create desired number of threads
    of crawling Process;

    Assign a seed URL and IP address of
    the crawl machine to each thread;
    Each thread makes connection to the
    Appropriate crawl machine;

    Migrate the crawling process to
    the destination crawl machine;
    Synchronize and control the
    transmission between the web
    crawler application and Crawl
    machines using URLlistupdate_wca
    and pagedownloader_wca;
End
```

Crawl Machine Module

Each crawl machine modules are the process that takes the different seed URLs from the central machine and each crawl machine crawl the web independently. Each crawl machine are also capable to act as a central machine that can connect and supply the seed URLs to other next crawling level independent crawl machines which in turn can connect and supply seed URLs to other next crawling level independent crawl machines and so on, which causes it can grow the crawling size exponentially. The crawl machines extract the URLs links from the web pages and update the local URL list and parallel update the URL list of

the central machine. Each crawl machine also downloads the web pages from the web and updates its local page database.

The crawl machine module consists of the following components-

- Counter_cm
- ThreadController_cm(Same as ThreadController_wca)
- pageDownloader_cm
- Filter_cm
- Compress_cm
- Transfer_cm

Counter_cm : This module is similar to Counter_wca.

```
Algorithm  Counter_wca
Begin
  Decrement the received counter
  value by 1;
  If  counter  value=0  then
    Don't migrate the crawling
    process to next level;
  Else
    Send the counter value to the
    next level of crawl machine;
End
```

pageDownloader_cm

The page downloader module crawls the web starting with the seed URL supplied by the thread controller module of the previous level crawl machine or from the central machine. It uses four data structures vectortosearch, vectorsearched, vectormatches and a URL List. It stores each new extracted URL from web pages at the last of vectortosearch, vectormatches and the URL list. Each time it takes a URL in FIFO order from the vectortosearch once the URL is resolved and crawled on the web it deletes the URL from the vectortosearch and add this URL at the last of vectorseached.When this module finds URL from the web pages it checks the vectormatches for the URL, if this URL is already in vectormatches then the URL is ignored otherwise it is added to the last of vectotosearch, vectormaches and URL List.

```
Algorithm pageDownloader_cm
Begin
  While (vectortosearch is not empty or
  crawl size reaches to the max. limit)
    {
     Take the URL from vectortosearch
     in FIFO Order;
     If URL is not robot safe
       then Break;
     If URL can not resolve
       then Break;
     Extract the file on the  URL;
     Store the file in the
     crawlerdatabase; While there
     is a hyperlink in the file
       {
        Extract the URL from the  file;
        If URL can not resolved then
          Write message to the
          invoking process thread;
          Continue;
        If URL is not robot safe
          then Write message to the
          invoking application
          thread;
          Continue;
        If URL is not in
          vectormatches then Add URL
          to vectortosearch,
          vectormatches, and URL List;
          Send URL to the invoking
          process thread;
       }
     Increase crawl  size by 1;
    }
  Send message "done" to the invoking
thread; End
```

Filter_cm

This module filters the web pages in the crawlerdatabase directory based on some user choice. In this work filtering is done based on file extensions such as .html, .txt etc.However files can be filtered as per requirements. This module picks up the filtered files from the crawlerdatabase and stores in a different directory named "filtered" on the same machine local disk drive**.**

```
Algorithm
filter_cm Begin
  Create a directory "filtered" on
  the local disk drive;
  Read crawlerdatabase directory;
  While there is .html file in
  crawlerdatabase
       Add .html files to the
       directory "filtered";
End
```

Compress_cm

This module is responsible for compressing all the files in the filtered directory into a single .zip file. This module first creates a directory named "zip_n" on local disk drive and then creates a .zip file into the zip_n directory then reads all the files in the filtered directory and then adds all the files into a single .zip file.

```
Algorithm  compress_cm
Begin
  Create a directory named zip_n
  on local disk drive;
  Crate a .zip file in zip_n directory
  ;  While  there  are  files  into
  "filtered" directory
            Add  files  to  .zip  file;
End
```

Transfer_cm

This module transfer the .zip files in the zip_n directory getting after compression to the previous level crawl machine's zip_n directory or finally to the central machine's crawlerdatabase directory.

```
Algorithm  transfer
Begin
     While there are files in
     zip_n directory
        {
         Send zip file to the previous
         level crawl machine's zip_n
         directory or at the last
         level to the central
         machine's crawlerdatabase
         directory; Send contents of
         zip files; Next zip file;
        }
End
```

4. Performance

Focus was on comparing the performance of parallel migrating web crawler with exponential growth with a standalone conventional crawler that does not use migration. This standalone crawler will download the pages locally and does not make any crawling process migration or transmission of data to any other machine as opposed to parallel migrating web crawler with exponential growth that will migrate the crawling process to other machines at some desired crawling depth and finally all crawling machines transmits the web pages to the central machine.

Time Measurement

After the execution of the conventional crawler and the parallel migrating crawler with exponential growth on five URLs sets, the following observations were made:

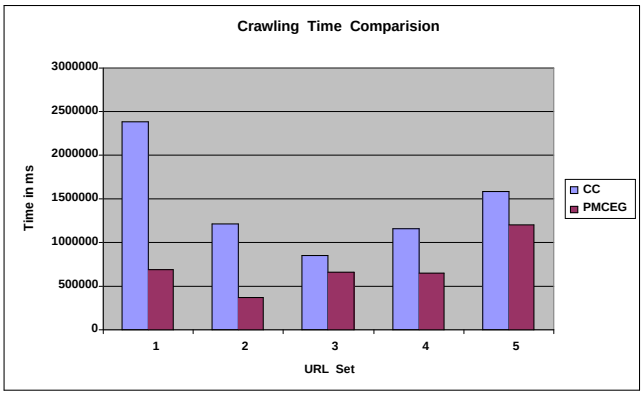

Fig 1.3 Crawling time comparison of Conventional crawler and parallel migrating web crawler with exponential growth

CC= Conventional crawler

PMCEG= Average crawling time of parallel migrating crawler with exponential growth

Average crawling time of Conventional crawler= 1439693.60 milliseconds.

Average crawling time of parallel migrating crawler with exponential growth= 715637.2 milliseconds.

% Benefit in time=100 – ((715637.2/1439693.6)*100)=51

Quality Measurement

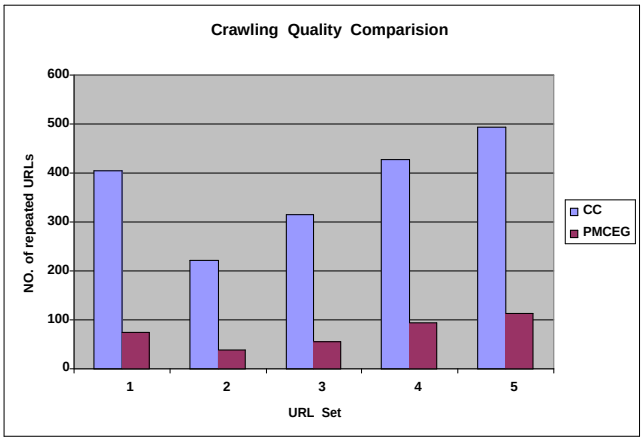

Fig 1.4 Crawling quality comparison of Conventional crawler and parallel migrating web crawler with exponential growth

Average number of repeated URLs in Conventional crawler= 372.2

Average number of repeated URLs in parallel migrating crawler with exponential growth= 74.6

%Benefit in quality=100 – ((74.6/372.2)*100) =80

Network Resource Utilization

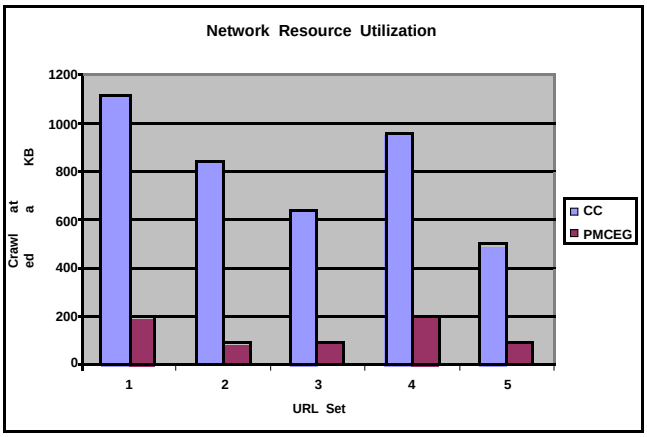

Fig 1.5 Network resource utilization comparison of Conventional crawler and parallel migrating web crawler with exponential growth

Average Crawled data in Conventional crawler= 810.68 KB

Average Crawled data in parallel migrating crawler with exponential growth= 136.62 KB

%Benefit in network bandwidth=

100-((136.62/810.68)*100)=83

5. Conclusion

The traditional parallel web crawlers download the web pages on a single machine, which causes bottleneck at the network level. The traditional migrating crawlers do not perform any compression and filtering before transmitting the web pages to the central machine. Moreover the Migrating web crawlers generally migrates themselves up to a single level of migration depth hence they are unable to take benefit of desired level of migration depth hence reduces the degree of parallelism. In this work the concept of download first transmits later is being proposed by using which data can be locally downloaded, filtered and compressed before transmitting it to the search engine server. In this work design and implementation of web crawler that can grow parallelism to the desired depth and can download the web pages that can grow exponentially in size is being proposed. The parallel crawler first filters and then compresses the downloaded web pages locally before

Transmitting it to the central machine. Thus the crawlers save the bandwidth of network and take the full advantage of parallel crawling for downloading and speedup the process.

6. Future work

In this work the following future aspects are arising. While downloading the web pages, we are using socket programming for file data transmission and dynamic URL list update it can be improved by using ftp protocol to reduce the time of file transmission. The socket programming produces the more number of lines of codes, the code optimization can be done by using some other technologies like RMI, JSP or servlet.

We have not provided any security measures in system connections, crawling process and in transmission of data among machines. It can be implemented some security mechanism.

The crawling process does not migrate automatically to the other machines we have to run the crawling machine module manually on other machines that will receive the crawling process by the central machine. This migration can be automatic using some other techniques like agelet etc.

References

1. Sullivan, D., Search Engine Watch, Mecklermedia, 1998, http://www.searchenginewatch.com.
2. Junghoo Cho, Hector Garcia-Molina,Parallel CrawlersWWW2002, May 7–11, 2002, Honolulu, Hawaii, USA. ACM 1-58113-449-5/02/0005
3. Joachim Hammer and Jan Fiedler ,Using Mobile Crawlers to Search the Web Efficiently.UF Technical Report, Number TR98-007, Gainesville, FL, June 1998
4. Douglas E. Comer, "The Internet Book", Prentice Hall of India, New Delhi, 2001.
5. Francis Crimmins, "Web Crawler Review".
6. A.K. Sharma, J.P. Gupta, D. P. Aggarwal, PARCAHYDE: An Architecture of a Parallel Crawler based on Augmented Hypertext Documents.
7. A.K.Sharma, Charu Bishnoi, Dimple Juneja, "A Multi-Agent Framework for agent based focused crawlers", Proc. Of International Conference on Emerging Technologies in IT Industry, pp- 48, ICET-04, Punjab, India, November 2004.
8. Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change, 2000. Submitted to VLDB 2000, Research track.
9. [9 Sriram Raghavan Hector GarciaMolina , Crawling the Hidden Web, Computer Science Department Stanford University Stanford, CA 94305, USA
10. The Deep Web: Surfacing Hidden Value.http://www.completeplanet.com/Tutorials/Deep Web.
11. S. Lawrence and C. L. Giles. Searching the World Wide Web. Science, 280(5360):98, 1998.
12. http://en.wikipedia.org/wiki/Web_crawler
13. S. Lawrence and C. L. Giles. Accessibility of information on the web. Nature, 400:107{109, 1999.
14. S. Raghavan and H. Garcia-Molina. Crawling the hidden web. Technical Report 2000-36, Computer Science Department, Stanford University, December 2000.
15. Design and Implementation of a High-Performance Distributed Web Crawler Vladislav Shkapenyuk and Torsten Suel
16. Distributing crawling techniques – A survey by Vikas Badgujar, Ashutosh Dixit, A.K.Sharma National conference on emerging trends in computing and Communicating ETTC-07.
17. C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, "The Harvest Information Discovery and Access System," in Proceedings of the Second International World Wide Web Conference, pp. 763-771, 1994.

AUTHOR'S PROFILE

Mr. Jitendra Kumar Seth received B.Tech. Degree in Computer Science and Engineering from Uttar Pradesh Technical University India, in 2004, and M.Tech Degree in Computer Science and Engineering from Shobhit University, Meerut, India in 2009. He is currently Assistant Professor in Dept. of Information Technology at Ajay Kumar Garg Engg. College, Ghaziabad, India. His research areas are Search Engine, Web Crawlers, Computer Network, java and Web programming, Mobile Computing and Algorithms.

Mr. Ashutosh Dixit is B.Tech, M.Tech in Computer Science. He is currently working as a senior lecturer in YMCA, Faridabad, Hariyana.His research area is web crawler, search engine, computer network and security.