A Scalable, Extensible Web Crawler Based On P2P Overlay Networks

P Mittal
Lecturer (IT)
YMCAIE, Faridabad
Haryana, India
09910117478
poonamgarg1984@gmail.com

A Dixit
Lecturer (CE)
YMCAIE, Fbd.
Haryana, India
09873356911
dixit_ashutosh@rediffmail.com

A.K.Sharma
Prof., HOD (CE)
YMCAIE, Fbd
Haryana, India
09811735202
ashokkale2@rediffmail.com

ABSTRACT
The World Wide Web is an interlinked collection of billions of documents. Ironically the very size of this collection has become an obstacle for information retrieval. The user has to sift through scores of pages to come upon the information he/she desires. Web crawlers are the heart of search engines. **Mercator** is a scalable, extensible web crawler, which support for extensibility and customizability. This paper explores the challenges and issues faced in using a single FIFO in Mercator as URL frontier. In addition, in Mercator, URL frontier is traversed every time as a new URL arrives. Since Mercator has **multithreaded** environment, many issues arises by using single FIFO. This paper also explores the concept of pastry in URL frontier implementation. Since single URL frontier is a constraint in multithreaded environment. Peer to peer overlay networks provide locality property, which improves application performance and reduces network usage. Hence Mercator's URL frontier uses the concept peer to peer overlay network's logic to find which is the best suitable canonical form for a newly arrived URL.Hence by using this logic an **algorithm** is designed which uses minimum comparisons to find the match. This work also explores the idea of using a single FIFO sub queue for each working thread and providing a canonical URL for each working thread. These working threads have a unique canonical URL form, which are grouped by using the concept of hash function. . Hence by using this logic an algorithm is designed through which searching is optimized hence this eager strategy is implemented by using the concept of hashing. Proposed system's design features a crawler core for handling the main crawling tasks, and extensibility through protocol and processing modules. The greatest positive impact occurred when there was a pronounceable change in the performance of the system as compared to the existing process.

Categories and Subject Descriptors
H.3.3 Information Search and retrieval

General Terms
Algorithms, Measurement, Performance, Design, Economics, Reliability, Experimentation, Security, Verification.

Keywords
Mercator, Overlay, PastryNetwork, URL frontier, Threads, Hashing.

1. INTRODUCTION
The World Wide Web [1] is a global, read-write information space. Text documents, images, multimedia and many other items of information, referred to as resources are identified by short, unique, global identifiers called Uniform Resource Identifiers so that each can be found, accessed and cross-referenced in the simplest possible way. 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.
Allan Heydon and Marc Najork [1] described "Mercator: A Scalable, Extensible web Crawler".Mercators's main support is for extensibility ant customizability. It also describes the particular components used in Mercator. Finally, we comment on Mercator's performance, which we have found to be comparable to that of other crawlers

This paper studies the challenges and issues faced in using a single FIFO in Mercator as URL frontier. Mercator is a scalable, extensible web crawler, which support for extensibility and customizability. This paper also studies the concept of exploiting network proximity in peer-to-peer overlay networks [2].This work also explores the idea of using a single FIFO sub queue for each working thread and providing a canonical URL for each working thread. These working threads have a unique canonical URL form, which are grouped by using the concept of hash function. Hence this eager strategy is implemented by using the concept of hashing. The greatest positive impact occurred when there was a pronounceable change in the performance of the system as compared to the existing process.The paper has been organized as follows: Section 2 describes constraints of existing system with single FIFO queue and constraints raised by using multiple FIFO queues and solutions of the constraints. Section 3 describes the algorithm for the proposed system and also explains how hashing

helps the proposed system. Section 4 describes the change in performance by using the proposed system. Section 5 draws the conclusion and describes the future research.

2. ARCHITECTURE

2.1 Existing System

The URL frontier is the data structure used in Mercator 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 dequeued in the order they were enqueued.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.

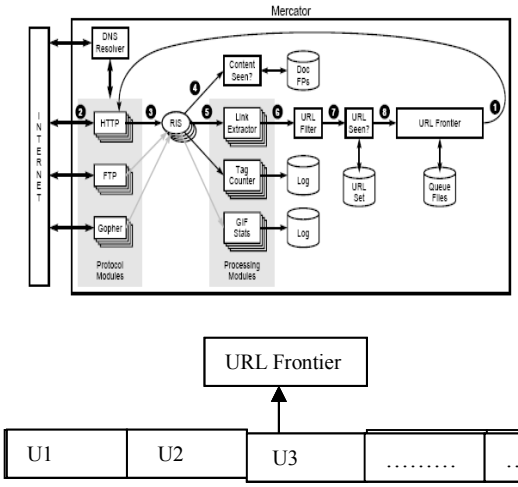


Figure 2.1 Architecture of Mercator with a single frontier

Main constraints of the existing system are:-

- Mercator uses a single FIFO as URL frontier; it is a main issue in multithreaded environment.
- Multiple HTTP requests are pending to the same server are unacceptable.
- Since Mercator uses a single FIFO, it has to put all URLs in the FIFO, which approaches to same page for a longer time.
- As a new URL arrives to check its presence we have to traverse the so much long FIFO, which is wastage of resources.
- If multiple requests are made in parallel then queues remove operation not only returns the head of the queue but the URL close to the head.

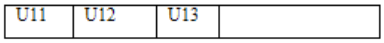
2.2 Proposed Architecture of URL Frontier

The Mercator's URL is now implemented using FIFO sub queues. There are 2 aspects of adding and removing URLs from these sub queues: -

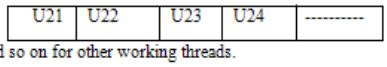
- There is exactly 1 FIFO sub queue for each working thread. Each working thread removes URLs from exactly 1 FIFO sub queue.
- When a new URL is added, the FIFO sub queue in which it is placed is determined by the URL's canonical host name.

These 2 points collectively solves the above problems and 1 more bottleneck of the crawl is solved because such a design of frontier prevents Mercator from overloading a web server. Since each working thread has its own separate FIFO sub queue and it has the form like this: -

- **For thread 1**



- **For thread 2**



And so on for other working threads.

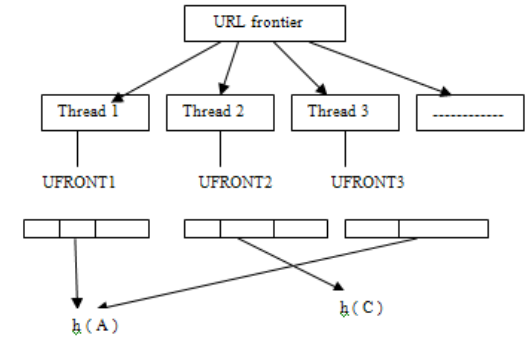


Figure 2.2 Architecture of Proposed Mercator's frontier

A canonical form of URL is specified for each working thread. Each thread has its FIFO sub queue on which add and remove operation performs. Since these threads are dynamically generated hence their FIFO sub queues are also dynamically generated as a new thread comes in existence and a canonical form of URL is provided to that thread. Each thread has a unique canonical URL form. In this way this system follows the following steps:-

- Initially the process starts with a seed URL.
- All the resulted URLs are provided to the working threads (as much different URLs we get that much working threads starts working on their seed URLs).
- On the basis of seed URL, each working thread makes its canonical form.
- Now each working thread starts with its seed URL, as a working thread gets URLs from its seed URL, as compares the URLs from the canonical URL, as it matches then it enqueued that URL in its own FIFO sub queues otherwise matches with other thread's canonical URL, as any match is found then a new working thread as well as a new FIFO sub queue corresponding to that is formed.

Main constraints of this process are: -

- If seed URLs of 2 threads results in same URL.

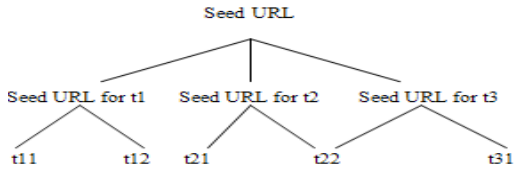


Figure 2.3 Two threads with common URL

Solution: **-** By using the concept of FCFS, repetition can be prevented.
- Suppose there are 3 working threads and suppose that 3rd thread does not have any entry after step 1 and in the coming future, so it's wastage of resources.

Solution: **-** There should be a concept of timestamp, as this timestamp is over and no new entry comes set its dirty bit as 1.As a demand for new thread arises then the thread whose dirty bit=1 is used for this and its canonical URL is changed as well as its timestamp and dirty bit are made as 0.

3. ALGORITHM

Algorithm to provide a URL to an appropriate URL Frontier Sub Queue

```
// initially we are having a seed URL
//I=1,threadcount,timestamp=0,dirtybit=0,timestampmax=50,
Result=false
while I! =threadcount
      if dirtybit[I]=0 and timestamp[I]!=timestampmax
                  increment timestamp[I];
                  k=compare (CURL [I], URL);
                  if k==1
                              Enqueue(URL,UFRONT[I]);
                              timestamp [I]=0;
                              I=threadcount;
                              Result=true;
                  End if;
                  Else
                              timestamp [I]++;
                              if (timestamp[I]>=timestampmax)
                                    dirtybit[I]=1;
                              End if;
                              I++;
                  End else;
       End if;
End while loop;
 If result==false
      For (I=1; I<=threadcount;I++)
                  If dirtybit [I] ==1
                  Enqueue(URL.UFRONT[]);
                  Make new CURL [I];
                  dirtybit [I]=0;
                  timestamp [I]=0;
                  Result=true;
                  break;
                  End if;
      End for loop;
      If (I==threadcount and result=false)
                   Make new thread (threadcount+1);
                  Enqueue(URL,UFRONT[threadcount+1]);
                   I=Threadcount++;
                  dirtybit [I]=0;
                  timestamp [I]=0;
       End if
 End if;
```

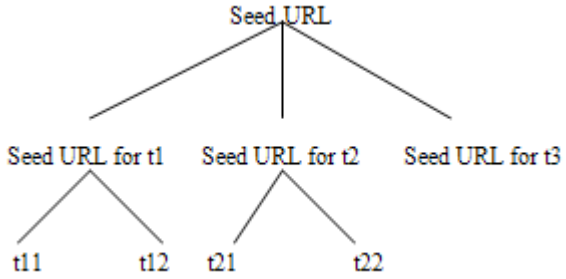
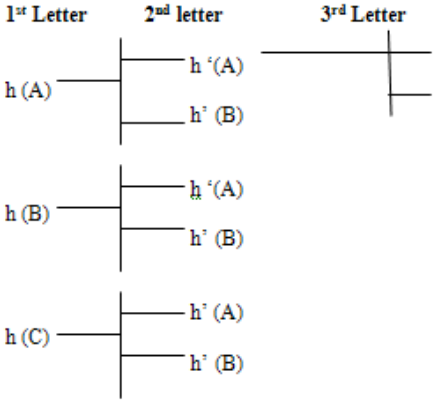


Figure 2.4. A thread with no URL

As a new URL arrives, the above algorithm starts executing for that URL.It starts with the first thread, first it compares the URL to the canonical URL of the threads. This is shown by an example: Suppose CURL has the form http://Crawler/Mercator and URL has the form http://Crawler/Mercator/details.html. First it compares the first letter by using alphabetical order or using its place value and in this way it compares the complete URL.It these URLs are equivalent, then it results in true otherwise false. This is not an appropriate way of comparison since we have millions of URLs and for these millions of URLs we have thousands of working threads. Hence it's a typical task to compare an URL with thousands of Curls. Hence hashing is used to group the threads on the basis on hash function. As shown below: -



In this way a newly arrived URL is placed to an appropriate queue. But if no match is found then a thread whose timestamp is timestampmax and dirty bit=1 is used for this URL with new canonical URL form else a new thread with URL frontier is dynamically generated.

4. PERFORMANCE

4.1 Without using Hash Function

As a new URL arrives it is to be compared with all canonical forms until no match is found. If a URL's starting few characters matches but after that no match then it has to start the process again from the first character. Best case occurs when first canonical URL matches with the arrived URL.Hence number of comparisons equals to the length of the URL.It the worst case match is found nearly at the last thread as shown in the graph.

Similarly in average case match is found nearly at the half of the working threads as shown in the graph.

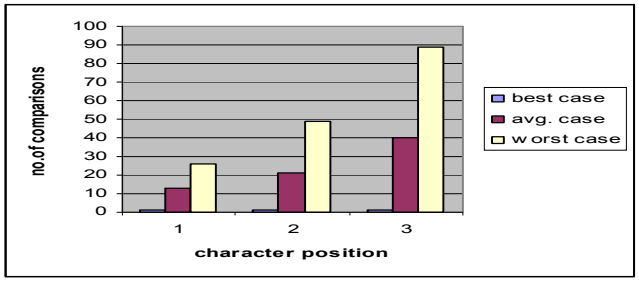


Figure 4.1 Performance using the hash function

4.2 Performance using Hash Function

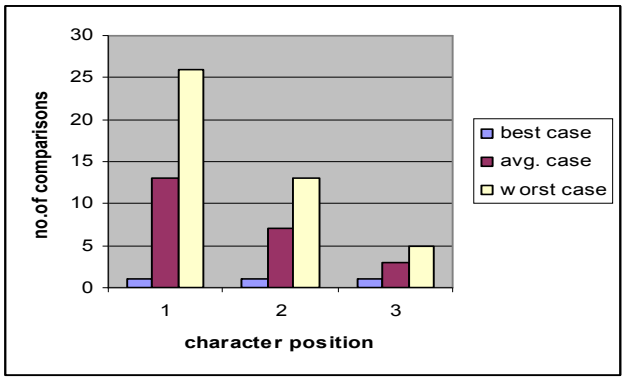


Figure 4.2 Performance using the hash function

When there are hundreds of working threads with unique canonical URL.By applying hash function on 1st character these canonical URLs can be grouped in 26 groups or less As a new URL arrives hash function is calculated on 1st character of this URL if it matches with any of the group then comparison will take place in a very small domain hence number of comparisons decreases. For best case performance is same but there is major difference in worst and average case. Since performance is inversely propositional to number of comparisons. Hence we can conclude from the graph that by using the proposed concept, that there are very less number of comparisons.

5. CONCLUSION AND FUTURE WORK

Since the existing system uses single FIFO queue and it is very difficult to handle millions of URLs in a single queue. At the same time it is difficult to perform add, remove and compare operation in multithreaded environment. This paper has enumerated the concept of multiple FIFO sub queues and each FIFO sub queue has a unique canonical URL.

These canonical URLs are used for comparison as a new URL arrives. The canonical URLs are grouped together by using the concept of hashing which shows a rapid improvement in the performance because now it requires very less number of comparisons as a new URL arrives. Proposed system's design features a crawler core for handling the main crawling tasks, and extensibility through protocol and processing modules. There are few other issues, which should be identified as the proposed work is using the concept of FCFS, but it is not always beneficial. Since, it may be the case that only first letter matches others not, hence there should be some factor other than the hash function. Existing system can also be extended to use it to index the intranet.

6. REFERENCES

[1] Allen Heydon and Mark Najork, "Mercator: A Scalable, Extensible Web Crawler".

[2] A.K.Sharma, J.P.Gupta, D.P.Agarwal, "An Alternative Scheme for Generating Fingerprints of Static Documents", accepted for publication in Journal of CSI.

[3] A. K. Sharma, J. P. Gupta, D. P. Agarwal, "A novel approach towards efficient Volatile Information Management", Proc. Of National Conference on Quantum Computing, 5-6 Oct.' 2002, Gwalior.

[4]AltaVista, "AltaVista Search Engine," WWW, http://www.altavista.com. [5] Z.Smith, "The Truth about the Web: Crawling towards Eternity", Web Techniques Magazine, 2 (5), May 1997.

[5] Douglas E. Comer, "The Internet Book", Prentice Hall of India, New Delhi, 2001.

[6] F.C. Cheong, "Internet Agents: Spiders, Wanderers, Brokers and Bots", New Riders Publishing, Indianapolis, Indiana, USA, 1996.

[7] Miguel Castro Peter Druschel_ Y. Charlie Hu_ Antony Rowstron- Exploiting network proximity in peer to peer overlay networks

[8] Mike Burner, "Crawling towards Eternity: Building an archive of the World Wide Web", Web Techniques Magazine, 2(5), May 1997.

[9] Robert C. Miller and Krishna Bharat, "SPHINX: a framework for creating personal,site-specificWeb-crawlers".

[10] Shaw Green, Leon Hurst, Brenda Nangle, Dr. Pádraig Cunningham, Fergal Somers, Dr. Richard Evans, "Sofware Agents: A Review", May 1997.

[11] Vladislav Shkapenyuk and Torsten Suel, "Design and Implementation of a High performance Distributed Web Crawler", Technical Report, Department of Computer and Information Science, Polytechnic University, Brooklyn, July 2001.

[12] Z.Smith, "The Truth about the Web: Crawling towards Eternity", Web Techniques Magazine, 2 (5), May 1997.

[13] Sergey Brin and Lawrence Page, "The anatomy of large scale hyper textual web search engine", Proc. of 7th International World Wide Web Conference, volume 30, Computer Networks and ISDN Systems, pp 107-117, April 1998.