A Novel Architecture of a Parallel Web Crawler

Shruti Sharma
AP Dept of CE
YMCA UST, India

A.K.Sharma
Prof & Head (CE)
YMCA UST, India

J.P.Gupta
Vice-Chancellor
JPIIT, India

ABSTRACT
Due to the explosion in the size of the WWW[1,4,5] it becomes essential to make the crawling process parallel. In this paper we present an architecture for a parallel crawler that consists of multiple crawling processes called as C-procs which can run on network of workstations. The proposed crawler is scalable, is resilient against system crashes and other event. The aim of this architecture is to efficiently and effectively crawl the current set of publically indexable web pages so that we can maximize the download rate while minimizing the overhead from parallelization

Keywords
WWW, Search Engines, Crawlers, Parallel Crawlers.

1. INTRODUCTION
The World-Wide Web has undergone explosive, exponential growth. As a consequence, users find themselves unable to browse the ever-changing, distributed hyperlink structure of the web. Furthermore, they are subjected to information overload – information is too *abundant*. With the increasing number of information resources on the Web, it is often more difficult to locate the resources that are relevant to a given need. Henceforth, Search Engines[2,3] are becoming equally important tool in locating relevant information.

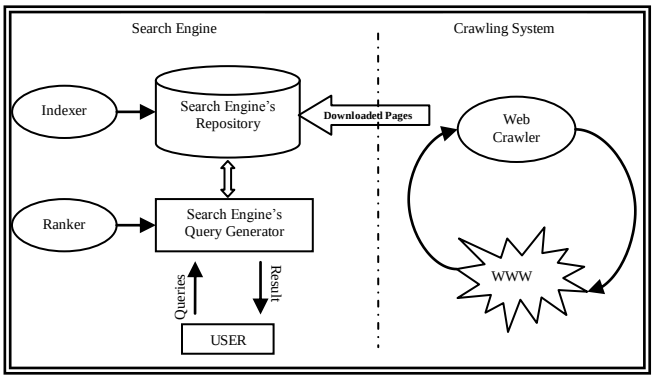

FIGURE 1: Functional block diagram of a Search Engine

Such search engines rely on massive collections of web pages that have been crawled by web crawlers, which traverse the web by following hyperlinks thereafter storing downloaded pages in a large repository that is later indexed for efficient execution of user queries.

The functional block diagram of Search Engine is shown in Figure 1. Crawling can be viewed as a graph search problem. The Web is seen as a large graph with pages as its nodes and hyperlinks as its edges. A crawler starts at a few of the nodes (seeds) and then follows the edges to reach other nodes. The process of fetching a page and extracting the links within it is analogous to expanding a node in graph search. A web crawler contacts millions of web sites in a short period of time and consumes extremely large network, storage and memory resources. Since these loads push the limit of existing hardware, the task should be carefully partitioned among processes and they should be carefully coordinated.

A web crawler [9, 10, 11, 14] is an automatic web object retrieval system that exploits the web's dense link structure. It has two primary goals:

1. To seek out new web objects, and
2. To observe changes in previously-discovered web objects (web-event detection).

As the size of the Web grows, it becomes more difficult – or impossible – to crawl the entire Web by a single process. Many search engines run multiple processes in parallel. This type of crawler is referred to as a *parallel crawler* as shown in figure 2. In this paper we present an architecture for a parallel crawler that consists of multiple crawling processes, which are referred to as C-procs[3,5,7,8]. Each C-proc performs the basic tasks that a single-process crawler conducts. It downloads pages from the Web, stores the pages locally, extracts URLs from them and follows their links. A parallel crawler has many important advantages as compared to a single-process crawler:

Scalability: Due to enormous size of the Web, it is not possible for a single process crawler to achieve the required download rate therefore it is essential to run a parallel crawler.

Network-load dispersion: Multiple crawling processes of a parallel crawler may run at geographically distant locations, each downloading "geographically-adjacent" pages. This dispersion becomes necessary when a single network cannot handle the heavy load from a large-scale crawl.

Network-load reduction: In addition to the dispersing load, a parallel crawler may actually *reduce* the network load. To download the page crawler, first has to go through the network and the collect pages. If we can somehow divide the areas to be crawled by each crawler, then we can reduce the overall network load because pages go through only local networks.

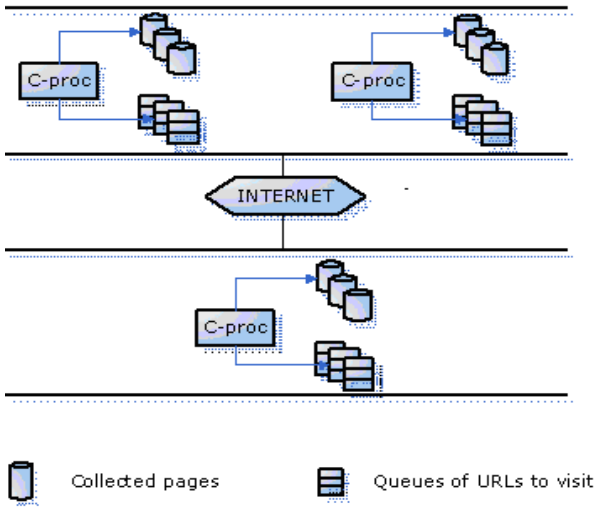

FIGURE 2 : General architecture of Parallel Crawler

We believe a parallel crawler[9,12,15] has many advantages and poses interesting challenges. To build an effective web crawler, we clearly need to address many more challenges than just parallelization. Henceforth, in this paper, a novel framework for Parallel Crawler has been proposed that efficiently crawls pages from the Web. The proposed technique uses Crawling Processes called as C-procs to download the pages from the web and store them to the search engine's Repository. Moreover, these Crawling processes are independent from each other.

2. PROPOSED SYSTEM ARCHITECTURE

In this paper we illustrate the general architecture for a parallel crawler which include multiple crawling processes; referred to as C-procs. Each C-proc performs the basic tasks that a single-process crawler conducts. It downloads pages from the Web, stores the pages locally, extracts URLs from them and follows their links. The C-proc's performing these tasks may be distributed either on the same local network or at geographically distant locations.

Intra-site parallel crawler: When all C-proc's run on the same local network and communicate through a high speed interconnect (such as LAN), we call it an intra-site parallel crawler. This scenario corresponds to the case where all C-proc's run only on the local network on the top. That is, all C-proc's use the same local network when they download pages from remote Web sites. Therefore, the network load from C-proc's is centralized at a single location where they operate.

Distributed crawler: When C-proc's run at geographically distant locations connected by the Internet (or a wide area network), we call it a distributed crawler. For example, one C-proc may run in the US, crawling all US pages, and another C-proc may run in France, crawling all European pages.

We partition the system into two major components - crawling system and mapping application. The crawling system itself consists of several specialized components, in particular a crawl manager, one or more crawling processes referred as C-procs. All of these components, plus the mapping application, can run on different machines (and operating systems) and can be replicated to increase the system performance. The crawl manager is responsible for receiving the URL input stream from the URL Mapper and forwarding it to the available C-procs.

The task of this Mapping Application is to provide the IP address corresponding to each URL in the URL-IPQueue. Figure 3, shows the main data flows through the system. The details of each component of the architecture are given below:

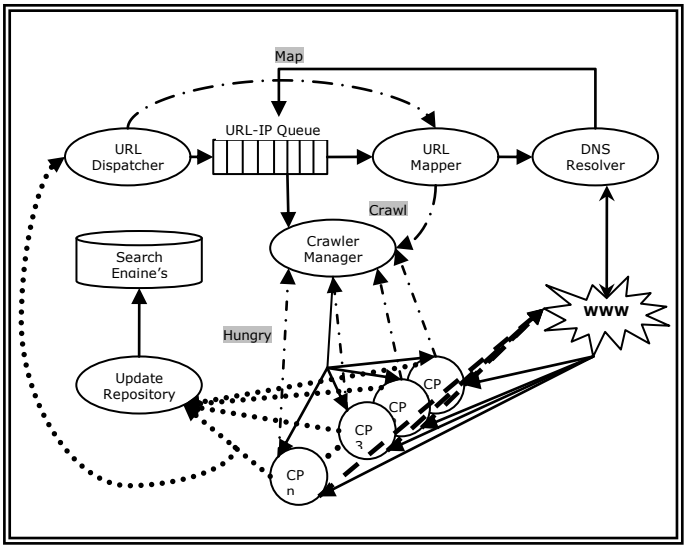

FIGURE 3: Proposed architecture of a Parallel Crawler along with the data flows

2.1 URL Dispatcher

This module reads the URLs from the repository (for refreshing) and also from various C-procs (ie the external links). As the name depicts, the task of this module is to dispatch the URLs which are required to be downloaded to the Crawl Manager. It actually adds these URLs to a shared Queue called as URL-IPQueue. It also signals Map to the URLMapper for necessary action. Its algorithm is listed below:

```
Begin
  Do Forever
   Begin
     While (URL-IP Queue not Full)
      Begin
        Read URL-IP pair from Repository;
        Read the URLs from C-procs;
        Store it into URL-IP Queue;
      End;
     Signal (Something to Map);
  End;
End;
```

2.2 DNS Resolver:

Generally the documents are known by the domain names of their servers[18, 19]. The name of the server must be translated into an IP address before the crawler can communicate with the server [16]. The internet offers a service that translates domain names to corresponding IP addresses and the software that does this job is called the Domain Name System (DNS). The

DNS resolver uses this service to resolve the DNS address for a URL.

2.3 URLMapper

The task of this component is to extract a URL-IP set from the URL-IPQueue. It examines each URL-IP pair and if IP is blank then the URL is sent to the DNS Resolver. After the URL has been resolved for its IP, it is stored back into the URL-IPQueue. It sends a signal crawl to the Crawl Manager. Its algorithm is given below:

```
URL-Mapper ( )
Begin
  Do forever
    Begin
      While (URL-IPQueue is not empty)
       Begin
         Take a URL-IP pair from the set;
         If the IP is blank then
         Begin
           Call DNS resolver to resolve URL for IP;
           Wait for the resolved URL;
        End;
        Store the Resolved URL back into URL-
        IPQueue;
        Signal (crawl) to Crawl Manager;
      End;
    signal (Hungry);
  End;
End.
```

2.4 C-procConfig.txt

It is a worker configuration file which is used by the Crawl Manager to load the initializing data. The contents of a sample file are tabulated in Table 1.

TABLE 1: Sample Contents of C-procConfig.txt

Name	Value	Description
DbUrl	Jdbc:odbc:thin:@ myhost:1521:orcl	Database URL
DbName	CrawlDb	The database Name
DBPassword	CrawlDb	The Database Password
MaxInstances	5	The maximum number of instances to be created for C-Proc Component.
LocalInstance	Yes	The instances are to be created on same or different machines.
ListIP	Localhost	If different machines are used for different C-procs then the list of th0se IP addresses.
Argument URL	5	The maximum number of URLs to be given as an argument to a C-proc.

2.5 Crawl Manager

This component waits for the signal crawl. It reads the C-procConfig.txt and as per the specifications stored in the file, it creates multiple crawling processes named as C-procs. Sets of resolved URLs from URL-IPQueue are taken and each C-proc is given a set. Its algorithm is given below:

```
CrawlManager ( )
Begin
  Read C-procConfig.txt file;
  Create multiple instances of C-procs: CP1 to CPm;
  Do forever
  Begin
    Wait (crawl);
    While (URL-IPQueue is not Empty)
    Begin
       Wait (hungry);
       Pickup resolved URLs from URL-IPQueue;
       Assemble and assign a set of URLs to an idle C-
       proc;
       Remove assigned URLs from the URL-
       IPQueue;
    End;
  End;
End.
```

2.6 C-Proc

The task of this module is to download the documents. It maintains two queues: MainQ and LocalQ. The set of URLs received from crawl Manager is stored in MainQ (see Fig. 3). It down loads the documents as per the algorithm given below:

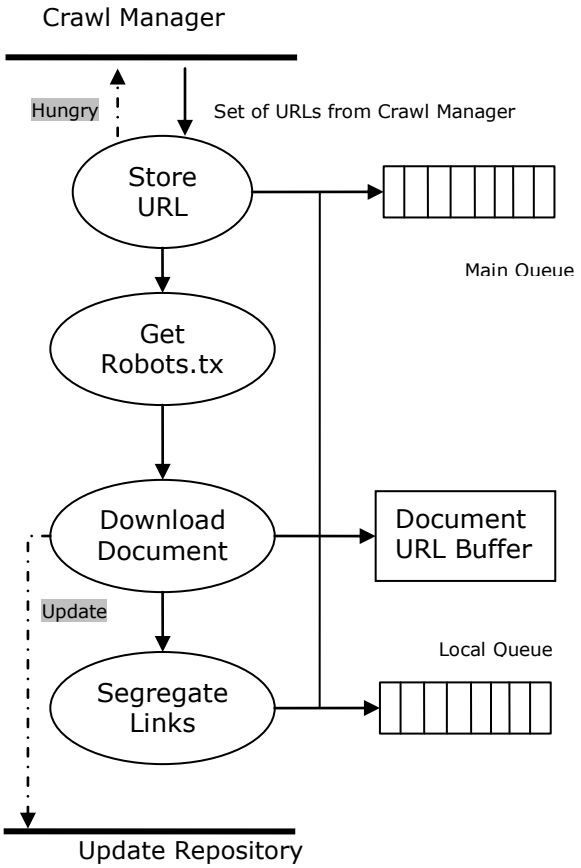

FIGURE 4 : The C-proc

```
C-Proc( )
Begin
  Store the URL set in MainQueue;
  While ( MainQueue is not empty)
  Begin
    Pickup a URL;
    Identify its protocol;
    Check for robots exclusion;
    Download document;
    Store the document and its URL in Document and
    URL Buffer;
     Segregate the internal and external Links;
    Add the internal Links to LocalQueue;
    Store the External Links in the URL-IPQueue;
    While (LocalQueue is not empty)
    Begin
       Pickup a URLfrom LocalQ;
       Download document;
       Store the document and its URL in
       Document and URL Buffer;
    End;
    Signal ( Update);
  End;
  Signal (hungry);
End.
```

2.7 Document and URL Buffer

It is basically a buffer between the c-procs and the Update Repository component. It consists of a Document-URL queue to store the documents along with there URLs that are added to the Search Engine's Repository whenever a c-proc signals update to the update repository module.

2.8 Update Repository

This process waits for the signal update repository and on receiving the same, updates the repository [16, 21] with the contents of the Document and URL Buffer. The algorithm of Update Repository Process is given below:

```
Update Repository()
Begin
  Set MaxSize to the maximum size of a batch;
  Do forever
  Begin
    Wait (update);
    No-of-records = 0;
    While (DocURLQueue is not empty & No-of-
          records < MaxSize)
    Begin
       Pickup an element from DocURLQ;
       Compress the document;
       Add to the batch of records to be updated;
       No-of-records= No-of-records+1;
    End;
    Update batch to database;
  End;
End.
```

It may be noted here that each c-proc independently downloads documents for the URL set received from Crawl Manager. Since all workers use different seed URLs, we hope that there will be minimum overlap of downloaded pages. Thus, the architecture requires no coordination overheads among the workers rendering it to be highly scalable system.

3. Experimentation & Results

Table 2 : Results for URL: http://www.modusporta.com

	Start Time (HH:MM:SS)	**End Time (HH:MM:SS)**	**Total Time (Sec)**
Parallel Crawler	**08:18:17**	**08:18:39**	**21**
Crawling process	**08:18:18**	**08:18:39**	**20**
Total Internal Links	**7**		
Invalid Link	**2**		
Valid Links	**5**		
Average Time to download and analyze page		**3.00 Seconds**	

Table 3 : Results for URL : http://www.e-paranoids.com

	Start Time (HH:MM:SS)	**End Time (HH:MM:SS)**	**Total Time (Sec)**
Parallel Crawler	**08:45:12**	**09:51:37**	**387**
Crawling process	**08:45:13**	**09:51:37**	**387**
Total Internal Links	**74**		
Invalid Link	**4**		
Valid Links	**73**		
Average Time to download and analyze page		**5,23 Seconds**	

Table 4: Comparison of Basic Crawler with Parallel Crawler

No of Links	Basic Crawler	Parallel Crawler	BC vs PC
7	**25**	**21**	**16.00%**
74	**482**	**387**	**16.60%**

The above listed tables table 2 and table 3 show the result of crawling the URLS http://www.modusporta.com and http://www.e-paranoids.com respectively. Start Time and end time depicts the start time and end time of a parallel crawling application. This crawler is implemented in java using RMI and Threads for parallel and distributed processing. Table 4 depicts the comparative analysis of a basic web crawler with a parallel web crawler that has 5 C-procs running in parallel. The new proposed Parallel Web Crawler took 16.30% less time then Basic Crawler.

4. CONCLUSIONS & FUTURESCOPE

Crawlers are being used more and more often to collect Web data for search engine, caches, and data mining. As the size of the Web grows, it becomes increasingly important to use parallel crawlers. This paper has enumerated the major components of the crawler and their algorithmic details. Parallelization of crawling system is very vital from the point of view of downloading documents in a reasonable amount of time. Also it is designed to be scalable parallel crawler. There are obviously many improvements to the system that can be made. A major open issue for future work is a detailed study of Database indexing and normalization that can be done so as to increase the work efficiency of the overall system. Inter-processes communication can also be added. Some mechanism for removing duplicate downloads shall also be included in cases where different URLs point to same page.

5. REFERENCES

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

[2] Berners-Lee and Daniel Connolly, "Hypertext Markup Language.Internetworking draft", Published on the WWW at http://www.w3.org/hypertext/WWW/MarkUp/HTML.html.

[3] Jumgoo Cho and Hector Garcia-Molina, "The evolution of the Web and implications for an incremental crawler", Prc. Of VLDB Conf., 2000.

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

[5] Junghoo Cho, "Parallel Crawlers" proceedings of www2002, Honolulu, hawaii, USA, May 7-11, 2002. ACM 1-58113-449-5/02/005.

[6] A.K.Sharma, J. P. Gupta, D. P. Agarwal, "Augment Hypertext Documents suitable for parallel crawlers", Proc. of WITSA-2003, a National workshop on Information Technology Services and Applications, Feb'2003, New Delhi.

[7] http:/research.compaq.com/SRC/mercator/papers/www/paper.html Jonathan Vincent, Graham King, Mark Udall, "General Principles of Parallelism in Search/Optimisation Heuristcs",

[8] 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.

[9] Brian Pinkerton, "Finding what people want: Experiences with the web crawler."Proc. Of WWW conf., 1994.

[10] Jumgoo Cho and Hector Garcia-Molina, "The evolution of the Web and implications for an incremental crawler", Prc. Of VLDB Conf.,2000.

[11] 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.

[12] Junghoo Cho and Hector Garcia-Molina, "Incremental crawler and evolution of web", Technical Report, Department of Computer Science, Stanford University.

[13] Alexandros Ntoulas, Junghoo Cho, Christopher Olston "What's New on the Web? The Evolution of the Web from a Search Engine Perspective." *In Proceedings of the World-Wide Web Conference (WWW),* May 2004.

[14] Michael K. Bergman, "The deep web: Surfacing hidden value", Journal of Electronic Publishing, 7(1), 2001.

[15] V. Crescenzi, G. Mecca, and P. Merialdo. "Roadrunner: Towards Automatic Data Extraction from Large Web Sites," VLDB Journal, 2001, pp. 109-118.

[16] P. G. Ipeirotis and L. Gravano, "Distributed search over the hidden-web: Hierarchical sampling and selection," In Proceedings of VLDB '02, 2002, pp. 394-405.

[17] Robots exclusion protocol. http://info.webcrawler.com/mak/projects/robots/exclusion.html.

[18] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: a highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, April 1990.

[19] D. Hirschberg. Parallel algorithms for the transitive closure and the connected component problem. In *Proceedings of the 8th Annual ACM Symposium on the Theory of Computing*, 1976.