The Issues and Challenges with the Web Crawlers

Satinder Bal Gupta

Professor, Department of Computer Science & Applications,
Vaish College of Engineering, Rohtak, Haryana, India.
Email: satinder_bal@yahoo.com

Abstract: A search engine is an information retrieval system designed to minimize the time required to find information over the Web of hyperlinked documents. It provides a user interface that enables the users to specify criteria about an item of interest and searches the same from locally maintained databases. The criteria are referred to as a search query. The search engine is a cascade model comprising of crawling, indexing, and searching modules. Crawling is the first stage that downloads Web documents, which are indexed by the indexer for later use by searching module, with a feedback from other stages. This module could also provide on-demand crawling services for search engines, if required. This paper discusses the issues and challenges involved in the design of the various types of crawlers.

Keywords: Search Engine, Web Crawler, Focused Crawler, Collaborative Web Crawler, Incremental Crawler, Parallel Crawler, Distributed Crawler, Mobile Crawler

1. Introduction

one or more URLs that constitute a seed set. It picks a URL from this seed set, and then fetches the Web page from that URL. The fetched page is then parsed, to extract both the text and the links from the page (each of which points to another URL). The extracted text is fed to a text indexer. The extracted links (URLs) are then added to a URL frontier, which at all times consists of URLs whose corresponding pages have yet to be fetched by the crawler. Initially, the URL frontier contains the seed set; as pages are fetched, the corresponding URLs are deleted from the URL frontier. In continuous crawling, the URL of a fetched page is added back to the frontier for fetching again in the future. This simple looking recursive traversal of the Web is practically much more complicated due to many requirements of a crawling system - the crawler has to be distributed, scalable, efficient, polite, robust and extensible while fetching pages of high quality. The crawler is the most important part of a search engine since the quality of results depends upon the crawler. The crawlers have to work continuously to keep the page repository up-to-date.

Web-crawler software doesn't actually move around to different computers on the Internet, as viruses or intelligent agents do. A crawler resides on a single machine. The crawler simply sends HTTP requests for documents to other machines on the Internet, just as a Web browser does when the user clicks on links.

2. Related Work

There is not much material available about commercial crawlers. One major source of information is the Google project of Stanford University [1]. Another source of information about the crawlers is the Harvest project [2], which investigates the web crawling in detail. A web crawler consumes a significant amount of network bandwidth and other resources by accessing the web resources at a fast speed. This affects the performance of the server considerably. A significant amount of resources of underlining network are consumed to build a comprehensive full text index of the web. Further, to maintain the indices of a search engine up-to-date, crawlers constantly retrieve the pages at a fast speed. Thus the crawling activities of a single search engine cause a daily load of 60GB to the web [3]. This load will increase significantly in future as the web grows exponentially in the future.

Distributed and parallel crawling [4, 5] was purposed to increase the coverage and decrease in
bandwidth usage but this does not help a lot. The distribution of the crawling function was efficiently reducing the network bottleneck from the search engine’s site and improves the quality of the results, but these are not at all optimized. Hammer and Fiddler [6, 7] has critically examined the traditional crawling techniques. They purposed web crawling approach based on mobile crawlers powered by mobile agents. The mobile crawlers are able to move to the resources that need to be accessed in order to take advantage of local data access. After accessing a resource, mobile crawlers move on to the next server or to their home machine, carrying the crawling results in their memory. Papapetrou and Samaras [8] first designed and developed UCYMicra, (a distributed web crawling infrastructure) and then IPMicra (a distributed location aware web crawler that utilizes an IP address hierarchy and allows the crawling of links in a near optimal location aware manner) a mobile crawler based system. All the crawlers presented in the literature have been grouped into the following six categories:

- Focused Crawler
- Collaborative Web Crawler
- Incremental Crawler
- Parallel Crawler
- Distributed Crawler
- Mobile Crawler

The following sections describe the architectures and the various issues and challenges in implementing these different categories of crawlers.

### 3. Focused Crawler

A focused crawler [12, 13, 14, 15] is designed to only gather documents on a specific topic. The goal of the focused crawler is to selectively seek out pages that are relevant to a pre-defined set of topics. The topics are specified not using keywords, but using exemplary documents. A focused crawler analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the Web. This section discusses the architecture of focused crawler and various issues and challenges involved in implementing the focused crawler.

#### 3.1 Architecture of Focused Crawler

The focused crawler architecture is composed of three components, namely classifier, distiller and crawler. The classifier makes the decision on the page relevancy to determine its future link expansion. The distiller identifies those hub pages, pointing to many topic related pages to determine the priority of pages to be visited. Finally, the crawling module fetches pages using the list of pages provided by the distiller. A block diagram is shown in Fig. 1.

The crawler has one watchdog thread and many worker threads. The watchdog is in charge of checking out new work from the crawl frontier, which is stored on disk. New work is passed to workers using shared memory buffers. Workers save details of newly explored pages in private per-worker disk structures.

The focused crawling is capable of steadily collecting relevant resources and identifying popular, high-content sites from the crawl. The focused crawler learns the specialization from examples, and then explores the Web, guided by relevance. It filters at the data-acquisition level, rather than as a post-processing step.

The most crucial evaluation of focused crawler is to measure the harvest ratio, which is the rate at which relevant pages are acquired and irrelevant pages are effectively filtered off from the crawl. This harvest ratio must be high, otherwise the focused crawler would spend a lot of time to eliminate irrelevant pages, and it may be better to use an ordinary crawler instead.

A focused crawler has the following advantages as compared to the other crawlers:

a) The focused crawler steadily acquires the relevant pages while other crawler quickly loses its way, even though they start from the same seed set.

b) It can discover valuable Web pages that are many links away from the seed set, and on the other hand prune millions of Web pages that may lie within same radius. Thus high quality collections of Web documents on specific topics can be built.

c) It can also identify regions of the Web that are dynamic or grow as compared to that are relatively static.

d) It can discover largely overlapping sets of pages and is robust.
3.2 Issues and Challenges with Focused Crawler

The following four issues were found with the focused crawlers:

**Missing Relevant Pages:** One issue with focused crawlers is that they may miss relevant pages by only crawling pages that are expected to give immediate benefit.

**Maintaining Freshness of Database:** Many HTML pages consist of information that gets updated on daily, weekly or monthly basis. The crawler has to download these pages and updates them into the database to provide up-to-date information to the users. The crawling process becomes slow and puts pressure on the Internet traffic if such pages are large in number. Thus, a major issue is to develop a strategy that manages these pages.

**Network Bandwidth and Impact on Web Servers:** The focused crawling techniques download many irrelevant Web pages that lead to consumption of network bandwidth. They adopt polling method or deploy multiple crawlers for the maintenance of freshness of database. Both methods consume lots of bandwidth. Also there is a crippling impact on the performance of Web servers if the crawlers are visiting them for all information. Thus, another issue is to develop some method to retrieve only highly related pages and alternate techniques to poll the Web server so that the underlying resources are not overloaded.

**Absence of Particular Context:** The focused crawler uses the best strategy to download the most relevant pages based on some criteria. The crawler focuses on a particular topic but in the absence of a particular context, it may download large number of irrelevant pages. Thus the challenge is to develop focused crawling techniques that focus on particular context also.

4. Collaborative Web Crawler

A Collaborative Web Crawler [16, 17] is a group of crawling nodes, in which each crawling node is responsible for a specific portion of the Web. This section discusses the architecture of Collaborative Web crawler and various issues and challenges involved in implementing the Collaborative Web crawler.

4.1 Architecture of Collaborative Web Crawler

A Collaborative Web Crawler (CWC) system uses more than one gatherer/crawler. These gatherers are organized in a collaborative fashion to explore a Web-space, to generate summaries, and to store these summaries for future reference. In order to achieve the maximum efficiency in CWC, these are coordinated so that the load is balanced and the overhead is minimized. Coordination is achieved by partitioning a Web-space into sub-space and assigning each subspace to a processor. Each processor is responsible for building the summaries for those URLs that are contained within its assigned sub-space. During the construction of a summary, new URLs may be discovered by the Crawler.

A method that allows the processors to communicate with each other is used to implement CWC. There are two types of information needed for the communication in CWC: foreign URLs, where a processor needs to send a URL given by a crossing hyperlink to the processor assigned to it; and coordination signals, used to re-map and load rebalance the system.

TSpaces (Tuple spaces) are used to support both foreign URLs and coordination signals. In a TSpaces server, unstructured tuples may be posted to a universally visible TSpaces. The tuples may be used and removed using a "Take" command and they may be read using a "Read" command by any processor in the system. Fig. 2 shows the architecture of CWC.

There are many load balancing measures that a CWC system optimize to increase the efficiency:
Work balance: The amount of work that a processor needs to perform is measured by the total time that a processor spends on retrieving pages from the partition on generating the summary of these pages, and on processing new hyperlinks from these pages.

Space balance: The amount of disk space needed by a processor is proportional to the total size of the summaries that a processor needs to build.

Minimize crossing links: Whenever a processor finds a page that is not in its partition, it may pass the information to the processor that is responsible for crawling page.

4.2 Issues and Challenges with the Collaborative Web Crawler

The following three challenges have been found with the Collaborative Web Crawler:

Overlap: In the collaborative crawling, it is possible that different crawling nodes download the same page multiple times. Multiple downloads of the same page are clearly undesirable. Thus the challenge is to develop techniques that reduce or eliminate these overlaps of pages.

Diversity: It is possible that the crawling is biased toward a certain domain name. For instance, a crawler might find a crawler trap, which is an infinite loop within the Web that dynamically produces new pages trapping the crawler within this loop. Thus the challenge is to develop techniques that are not biased and avoid the traps.

Communication overhead: In a collaborative crawling, the participating crawling nodes need to exchange URLs to coordinate the overall crawling work. To quantify how much communication is required for this exchange, the communication overhead is defined in terms of the exchanged URLs per downloaded page. The challenge is to minimize this overhead.

5. Incremental Crawler

The incremental crawler [9, 10, 11] continuously crawls the Web, revisiting pages periodically. During its continuous crawl, it may also purge some pages in the local collection, in order to make room for newly crawled pages. This section discusses the architecture of incremental crawler and various issues and challenges involved in implementing the incremental crawler.

5.1 Architecture of Incremental Crawler

The architecture for an incremental crawler is shown in Fig. 3. The architecture consists of three major modules namely RankingModule, UpdateModule and CrawlModule and three data structures namely AllUrls, CollUrls and Collection. The AllUrls data structure maintains information about all URLs that the crawler has discovered, and the CollUrls data structure maintains information about the URLs that are/will be in the Collection. CollUrls is implemented as a priority-queue, where the URLs to be crawled early are placed in the front. The URLs in CollUrls are chosen by the RankingModule.

The RankingModule constantly scans through AllUrls and the Collection to make the refinement decision. For instance, if the crawler uses PageRank as its importance metric, the RankingModule constantly reevaluates the PageRanks of all URLs, based on the link structure captured in the Collection. When a page not in CollUrls turns out to be more important than a page within CollUrls, the RankingModule replaces the less-important page in CollUrls with the more-important page. The URL for this new page is placed on the top of CollUrls, so that the UpdateModule can crawl the page immediately. The RankingModule also discards the less-important page from the Collection to make space for the new page.
The UpdateModule constantly extracts the top entry from CollUrls, requests the CrawlModule to crawl the page, and puts the crawled URL back into CollUrls. The position of the crawled URL within CollUrls is determined by the page's estimated change frequency. It also records the checksum of the page from the last crawl and compares that checksum with the one from the current crawl, to estimate how often a particular page changes. This comparison helps the UpdateModule to tell whether the page has changed or not. Two estimators EP and EB are used for the change frequency of a page. The estimator $EP$ is based on the Poisson process model, and the estimator $EB$ is based on a Bayesian inference method.

The CrawlModule crawls a page and saves/updates the page in the Collection, based on the request from the UpdateModule. The CrawlModule also extracts all links/URLs in the crawled page and forwards the URLs to AllUrls. The forwarded URLs are included in AllUrls, if they are new. Multiple CrawlModule's may run in parallel, depending on how fast the need to crawl pages.

5.2 Issues and Challenges with Incremental Crawler

There are two main challenges with the incremental crawlers:

**Keep the local collection fresh:** The freshness of a collection can vary widely depending on the strategies used by the crawler to crawl the Web. Thus keeping the collection fresh is a big challenge as best policies need to be used.

**Improve quality of the local collection:** The crawler should increase the quality of the local collection by replacing less important pages with more important ones. This is necessary because the pages are constantly created and removed and some of the new pages can be more important than existing pages in the collection. Also the importance of existing pages changes over time. Thus improving the quality of collection is another challenge.

6. Parallel Crawler

Crawling the whole Web with a single process is not possible since Web is growing rapidly. Therefore, many popular search engines employ multiple crawler processes in parallel to perform an efficient crawl. This type of a crawler is referred to as a parallel crawler [4]. This section discusses the architecture of parallel crawler and various issues and challenges involved in implementing the parallel crawlers.

6.1 Architecture of the Parallel Crawler

The general architecture of a parallel crawler is shown in Fig. 4. A parallel crawler consists of multiple crawling processes, referred as C-proc's (Crawling Processes). 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 the downloaded pages and then follows these URL’s. Depending on how the C-proc’s divide the URL’s collected from the downloaded pages, some of the extracted URL’s may be sent to other C-proc’s. The C-proc’s performing these tasks may be distributed either on the same local network or at geographically distant locations. Depending on the location of these processes there are two different architectures for parallel crawling- Intra-Site parallel Crawler and Distributed Crawler.

**Intra-site parallel crawler:** All C-proc’s run on the same local network and communicate with each other through a high-speed local network in an Intra-site parallel crawler. Here, all C-proc’s use the same local network when they download pages from the Remote Web sites. Therefore, the local network would have to handle the lots of
traffic caused by the requests that are made by all the C-proc’s.

**Distributed crawler:** When C-proc’s run at geographically distant locations connected by the Internet then it is called distributed crawler. The individual C-proc’s communicate among them using the Internet. Though the amount of requests would be the same the network traffic is distributed and the individual networks would not be strained as in an Intra-site parallel crawler.

A parallel crawler has the following important advantages, as compared to a single-process crawler:

a) The size of the Web is enormous and hence, it is better to run a parallel crawler. A single-process crawler cannot achieve the required download rate in certain cases.

b) Multiple crawling processes of a parallel crawler may run at geographically distant locations, each downloading “geographically-adjacent” pages. The network load can be dispersed to multiple regions using parallel crawlers. This dispersion is necessary when a single network cannot handle the heavy load from a large-scale crawl.

c) A parallel crawler also reduces the network load. This is possible since the different C-proc’s download pages local to their region.

**6.2 Issues and Challenges with Parallel Crawler**

Due to multiple C-proc’s working independently in a parallel crawler, there are many issues that need to be solved before implementing a parallel crawler. The various issues and challenges with the parallel crawlers are as follows:

**Multiple downloading of pages:** When a number of C-proc’s are working independently it is possible that more than one might download the same page multiple number of times, this is known as an overlap. This is because one C-proc is not aware that another C-proc has already downloaded the page. Existence of overlaps in a crawl become a challenge and tends to reduce the efficiency of a crawler. Generally, each of the C-proc’s work independently and do not care about overlaps. Each C-proc starts with its own seed and does not communicate with the other C-proc’s. It is believed that since each C-proc starts with an independent seed the amount of overlap is not very significant. In another technique, to reduce the overlapping of pages the Web is divided into partitions and each partition is assigned to a C-proc. Each C-proc downloads pages that are in its own partition and if the C-proc comes across a link that does not belong to its partition the link is reported to a central coordinator who would assign it as a seed to another C-proc. In yet another technique, the Web is partitioned before beginning the crawl and each C-proc extracts URL’s that are in its partition if an out of partition URL is seen it is directly reported to the C-proc to whose partition the URL belongs. In this technique there is no central coordinator.

**Quality of pages:** The quality of pages is determined from the ranking of returned pages. While crawling, the most important pages are downloaded first. However the C-proc’s in the parallel crawler do not have the complete image of the Web or the image of the Web with other crawlers hence downloading the most important pages first becomes a challenging task.

**Increased bandwidth Consumption:** To address the above two challenges, the C-proc’s have to establish communication with other C-proc’s in order to have the complete image of the Web. This leads to another problem of increased bandwidth consumption. The communication may grow significantly if the number of C-proc’s increase.
and would have to be reduced to maintain the efficiency of the crawl, which is a big challenge [11].

7. Distributed Crawler

A distributed crawler [5] is a Web crawler that operates simultaneous crawling agents. Each crawling agent runs on a different computer, and in principle some crawling agents can be on different geographical or network locations. On every crawling agent, several processes or execution threads running in parallel keep several hundred TCP connections open at the same time. This section discusses the architecture of distributed crawler and various issues and challenges involved in implementing the distributed crawler.

7.1 Architecture of Distributed Crawler

Shkapenyuk et al [5] design a flexible system that can be adapted to different applications and strategies with a reasonable amount of work. The crawler design is separated into two main components, referred to as crawling application and crawling system as shown in Fig. 5.

The crawling application decides what page to request next given the current state and the previously crawled pages, and issues a stream of URL requests to the crawling system. The crawling system downloads the requested pages and supplies them to the crawling application for analysis and storage.

The crawling system is in charge of tasks such as robot exclusion, speed control, and DNS resolution, while the application implements crawling strategies. The crawling system consists of several components that can be replicated for higher performance. Both crawling system and application can also be replicated independently, and several different applications could issue requests to the same crawling system.

The major advantages of the distributed crawler are that they completely avoid random I/O, while other systems use caching to catch most of the random I/O, and a fast disk system to handle the still significant remaining accesses. They use a few powerful downloader’s that open hundreds of asynchronous connections as compared to a thread for each queue that opens one connection using synchronous I/O.

7.2 Issues and Challenges with Distributed Crawler

The following six challenges have been found with the distributed crawlers:

Assignment of URL’s among different agents: The major challenge in distributed crawler is how the URLs are assigned efficiently and dynamically to download among the crawling agents. The assignment must take into consideration the various constraints like the rate of requests to Web servers should be minimize, the crawling agents should be located appropriately on the network, and the URLs should be exchanged effectively.

Priority in Crawling: The next challenge is how to efficiently prioritize the crawling frontier considering the dynamic nature of the Web.

Effective way of partitioning the collection: The next challenge is to find an effective way of partitioning the collection in such a way that the query answering phase work by querying not all partitions, but only the smallest possible subset of the partitions. The chosen subset should be able to provide a high number of relevant documents.

Load Balancing: The next challenge is to determine an effective way of balancing the load among the different index servers. There must be a good strategy to distribute the data in order to balance the load as much as possible.

Network bandwidth consumption: Network bandwidth is a scarce resource and is a big challenge. Therefore, when queries are resolved in
a distributed fashion, the servers that should be contacted must be determined efficiently. Also the number of necessary servers should be minimum. **Efficient cache design:** The next challenge is to design an effective cache that have high hit ratio and also overcomes the network constraints.

8. Mobile Crawler

The mobile crawlers [6, 7, 8] are constructed as mobile agents. Crawler mobility provides sophisticated crawling algorithms and avoids some of the inefficiencies associated with the strategies used by current crawlers. The mobile crawling is an efficient, scalable solution to establish a specialized search index in the highly distributed, decentralized and dynamic environment of the Web. This section discusses the architecture of Mobile crawler and various issues and challenges involved in implementing the mobile crawler [18, 19, 20, 21, 22].

8.1 Architecture of Mobile Crawler

Mobility in the context of Web crawling is the ability of a crawler to transfer itself to each Web server of interest before collecting pages on that server. After completing the collection process on a particular server, the crawler together with the collected data moves to the next server or to its home system. Mobile crawlers are managed by a crawler manager, which supplies each crawler with a list of target Web sites and monitors the location of each crawler. Fig. 6 shows architecture of Mobile Crawler.

A mobile crawler is sent to each Web server that contains relevant information for a set pages already selected. The crawler obtains a list of target locations from the crawler manager, when the crawler starts. These are called seed URLs. The crawler manager also uploads the crawling strategy into the crawler in form of a program. This program tells the crawler which pages are considered relevant and should be collected. It also generates the crawler path through the Web site. The crawler must migrate to a specific remote site using one of the seed URL’s as the target address, before the actual crawling begins. At the remote site, the crawling algorithm is executed.

After retrieving all the pages, the mobile crawler either returns to the crawler manager or, in case the list of seed URLs is not empty, it migrates to the next Web site on the list and continues. Once the mobile crawler has successfully migrated back to its home, all pages retrieved by the crawler are transferred to the search engine via the crawler manager.

The major advantages of a mobile crawler over the traditional crawler are as follows:

i) The mobile crawlers can access Web pages locally with respect to the server. This saves network bandwidth by eliminating request/response messages used for data retrieval.

ii) The mobile crawlers can select only the relevant pages before transmitting them over the network. This saves network bandwidth by discarding irrelevant information directly at the data source.

iii) The mobile crawlers reduce the content of Web pages before transmitting them over the network. This saves network bandwidth by discarding irrelevant portions of the retrieved pages.

iv) The mobile crawlers can compress the content of Web pages before transmitting them over the network. This saves network bandwidth by reducing the size of the retrieved data.

8.2 Issues and Challenges with the Mobile Crawlers

The following three major issues and challenges have been found with the Mobile crawlers:
Security: There are severe security problems due to crawler migration and remote execution of code because a mobile crawler may contain harmful code. Thus the challenge is to find some method so that mobile crawlers can be differentiated from harmful code.

Integration of the mobile crawler virtual machine into the Web: The mobile crawling is effective only when the mobile crawler virtual machine is available on most of the machines. This integration can be achieved through Java Servlets. Thus, the next challenge is to make the remote web servers compatible with the Java environment.

Less research in mobile crawling algorithms: The current crawling algorithms have not been designed with crawler mobility in mind. New algorithms need to be developed considering the crawler mobility.

9. Conclusions

All the web crawlers presented in the literature have been grouped into six categories. In this paper, I have discussed about the basic architectures presented by the researchers and the various issues and challenges faced in the development of these crawler architectures. I have found the many of the issues and challenges in these architectures are common i.e. reducing the network bandwidth consumption, maintaining the freshness of the database and maintaining the quality of pages etc.

The mobile crawler was constructed as mobile agent. The major challenges in designing the mobile crawler were to maintain the security, non-availability of required environment on most of the machines and less research in mobile crawling algorithms. Further, mobile crawlers are found to be the new paradigm and needs to be explored to get its benefits.

References


