Implementation of Two-stage Smart Crawler

Abstract

The number of web pages available in the Internet is growing tremendously day to day. In this case searching relevant information in the Internet is hard task. A lot of this information is hidden behind query forms that interface to unexplored databases containing high quality structured data. Traditional search engines cannot access and index this hidden part of the Web, retrieving this hidden information is challenging task. Therefore, proposed a two-stage framework, namely Smart Crawler, for effectively harvesting deep web interfaces. In the first stage that is site locating, centre pages are searched with the help of search engines which in turn avoid visiting a large number of pages. To achieve more precise results for a focused crawl, Smart Crawler ranks websites to prioritize highly relevant ones for a given topic. In the second stage, adaptive link-ranking achieves fast in-site searching by excavating most relevant links. This system is used to develop an error-free, fast retrieval of data whenever we search for data in any domain or website. Many websites (like Facebook) has many pages in which most of the pages may be unwanted or destroyed or may be affected by any virus. For this, we have made a complete analysis of domain which separates different domains like relevant links, bad links, candidate links, active links or inactive links.

I. INTRODUCTION

A web crawler is a system for the bulk downloading of web pages. Web crawlers are used for a variety of purposes. Most prominently, they are one of the main components of web search engines, systems that assemble a corpus of web pages, index them, and allow users to issue queries against the index and find the web pages that match the queries. A related use
is web archiving where large sets of web pages are periodically collected and archived for posterity. A third use is web data mining, where web pages are analyzed for statistical properties, or where data analytics is performed on them. Finally, web monitoring services allow their clients to submit standing queries, or triggers, and they continuously crawl the web and notify clients of pages that match those queries. There are several reasons why the push model did not become the primary means of acquiring content for search engines and other content aggregators: The fact that web servers are highly autonomous means that the barrier of entry to becoming a content provider is quite low, and the fact that the web protocols were at least initially extremely simple lowered the barrier even further in fact, this simplicity is viewed by many as the reason why the web succeeded where earlier hypertext systems had failed. Adding push protocols would have complicated the set of web protocols and thus raised the barrier of entry for content providers, while the pull model does not require any extra protocols. By the same token, the pull model lowers the barrier of entry for content aggregators as well: Launching a crawler does not require any a priori buy-in from content providers, and indeed there are over 1,500 operating crawlers, extending far beyond the systems employed by the big search engines.

II. RELATED WORK

There are many literatures in the area of web crawlers. In late 1994, The RBSE Repository Based Software Engineering project first launch the Web Crawler based on two programs: first was “spider”, it maintain a queue in a relational database, and second was “mite”, it is a modified www ASCII browser that download the pages from web. Then the second WebCrawler was publicly available full-text index of a subset of the web which was based on lib-WWW to download pages, and other program to parse and order URLs for breadth first exploration of web graph.

- Internet archive Crawler
  In 1997, Mike Burner designed the Internet Archive Crawler was the first paper that focused on the challenges caused by the scale of web. It uses multiple machine to crawl the web and it crawl on 100 million URLs. Each crawler process read a list of seed URLs for its assigned sites from disk into per-site queue, and then it uses asynchronous I/O instructions to fetch pages from these queues in parallel. It has also deal with the problem of changing DNS records, so it keeps the historical archive of hostname to IP mapping.

- Google Crawler
  Later in 1998, The original Google crawling system consist of a five crawling components which was running in various process and download the pages. Each crawler process used asynchronous I/O instructions to fetch the data from up to 300 web servers in parallel. Then all the crawlers transmit downloaded pages to a single Store Server process that compressed the page and store them on disk. Google Crawler was based on C++ and Python. This crawler was integrated with the indexing process.

- Mercator Web Crawler

Pallavi Pingale , Dr. Dhanraj Verma :: Implementation of Two-stage Smart Crawler
Heydon and Najork present a web crawler which was highly scalable and easily extensible. It was written in Java. The first version was non-distributed and later the distributed version was made available which split up the URL space over the crawlers according to host name and avoid the potential bottleneck of a centralized URL server.

- **WebFountain crawler**
  It has three major components Multi threaded crawling processes, duplicate content and central controlled process responsible for assigning work. It was written in C++ and used MPI to facilitate the communication between the various processes. It was deployed on a cluster of 48 crawling machine.

- **IRLbot Web crawler**
  Recently, Yan et al. describe IRLbot, which is single process web crawler. It is able to scale to extremely large web collection without performance degradation. It crawls over two month and downloads the 6.4 billion web pages.

---

**III. SYSTEM ARCHITECTURE**

Initially, the site locating stage starts with a seed set of sites in a site database. Seeds sites are candidate sites given to Smart Crawler to start crawling, which begins by following URLs from chosen seed sites to search other pages and other domains. When the number of unvisited URLs in the database is less than a threshold value during the crawling process then Smart Crawler performs "reverse searching" of known deep web sites for centre pages (highly ranked pages that have many links to other domains) and provides these pages to the site database. The site frontier will fetch web-page URLs from the site database. The unvisited sites are given to site frontier and are prioritized by site ranker, whereas the visited sites are added to fetched site list. Site Ranker assigns a score for each unvisited site that corresponds to its relevance to the already discovered deep web interfaces. The Site Ranker is improved during crawling by an Adaptive Site Learner. It will adaptively learn from features of deep-web sites found. To achieve more accurate results for a focused crawl, Site
Classifier categorizes URLs into relevant or irrelevant for a given topic according to the homepage content.

After the most relevant site is found in the first stage, the second stage performs efficient in-site exploration for excavating searchable forms. Links of a site are stored in Link Frontier and corresponding pages are fetched. Then embedded forms are classified by Form Classifier to find searchable forms. To improve accuracy of form classifier, pre-query and post-query approaches for classifying deep-web forms are combined. Additionally, the links in these pages are extracted into Candidate Frontier. To prioritize links in Candidate Frontier, Smart Crawler ranks them with Link Ranker. When the crawler discovers a new site, the site’s URL is inserted into the Site Database. The Link Ranker is adaptively improved by an Adaptive Link Learner, which learns from the URL path leading to relevant forms.

Here, the crawler is divided into following two stages:

1) **Site Locating**
2) **In-Site Exploring**

The site locating stage helps for succeeding wide coverage of sites for a focused crawler, and the 2nd stage helps in in-site exploring stage can efficiently perform searches for web forms within a site. We have proposed an original two stage framework to address the problem of searching for hidden-web resources. The site locating technique works a reverse searching technique and incremental two-level site prioritizing technique for unearthing relevant sites, for achieving more data sources. In in site exploring stage, we design a link tree for balanced link prioritizing, eliminating bias toward webpages in popular directories in the proposed System we have added web navigation module inside 2nd stage web crawler.

**Web navigation Module**

1) **Relevant Links** :
   Relevant links is finding links have proper path of candidate links means there is no missing links between candidate links.

2) **Bad Links** :
   Bad links is finding links have improper path of candidate links means there is no missing links between candidate links.

3) **Candidate Links** :
   Super links have sub-links:
   for e.g. 
   1. www.abcbook.com/sub/ 
   2. www.abcbook.com/sub/history.html 
   3. www.abcbook.com/sub/english.htm

4) **Active Links** :
   The input website contain links which are active, it means the given link page contain number of links but some links are called from URL properly.

5) **Inactive Links** :
The input website contain links which are inactive, it means the given link page contain number of links but some links are not called from URL properly.

IV. RESULT SET

<table>
<thead>
<tr>
<th>Domain</th>
<th>Site classification</th>
<th>Form classification</th>
<th>Searchable for...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airfare</td>
<td>95%</td>
<td>98%</td>
<td>98%</td>
</tr>
<tr>
<td>Auto</td>
<td>90%</td>
<td>97%</td>
<td>98%</td>
</tr>
<tr>
<td>Book</td>
<td>99%</td>
<td>98%</td>
<td>90%</td>
</tr>
<tr>
<td>Job</td>
<td>98%</td>
<td>95%</td>
<td>97%</td>
</tr>
<tr>
<td>Movie</td>
<td>95%</td>
<td>95%</td>
<td>97%</td>
</tr>
<tr>
<td>Rental</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
</tr>
<tr>
<td>product</td>
<td>99%</td>
<td>97%</td>
<td>99%</td>
</tr>
</tbody>
</table>

Figure 2: Smart crawler Result

Figure 3 shows the domain wise separated result details. It shows the site and form classification of if each and every domain by the percentage. Also shows the searchable domains details.

V. CONCLUSION

Finally the implemented Smart Crawler is a two-stage crawler for efficiently harvesting Deep-Web Interface. It has been shown that above approach achieves both wide scopes for deep web interfaces and maintains highly efficient crawling. Smart Crawler is a focused crawler consists of two stages: site locating and balanced in site exploring. Smart Crawler performs site -based locating by reversely searching the known deep web sites for centre pages, which can efficiently find many data sources for sparse domains. Smart Crawler achieves more accurate results by ranking collected sites and focusing the crawling on a given topic. The in -site exploring stage uses adaptive link-ranking to search within a site
and design a link tree for eliminating bias toward certain directories of a website for wider coverage of web directories. Here the result makes a complete analysis of domain which separates different domains like relevant links, bad links, candidate links, active links or inactive links.

VI. REFERENCES


TO CITE THIS PAPER