Abstract

In today’s era, Web has been rapidly deepened by myriad searchable databases online, where data are hidden behind query interfaces. There has been increased interest in techniques that help efficiently locate deep-web interfaces. However, due to the large volume of web resources and the dynamic nature of deep web, achieving wide coverage and high efficiency is a challenging issue. The review paper contains survey on the basic crawler and the smart crawlers. The two-stage framework, namely SmartCrawler, for effectively harvesting deep web interfaces. In the first stage that is site locating, center 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, SmartCrawler 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. To eliminate bias on visiting some highly related links in hidden web directories, design a link tree data structure to acquire wider coverage for a website. The experimental result on a set of representative domains show the agility and accuracy of proposed crawler framework which efficiently retrieves deep web interfaces from large-scale sites and obtains higher harvest rates than other crawlers.
I. INTRODUCTION

In the recent years, the Web has been rapidly deepened with the prevalence of databases online. 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.

The crawler consists of multiple processes running on different machines connected by a high-speed network. Each crawling process consists of multiple worker threads, and each worker thread performs repeated work cycles. At the beginning of each work cycle, a worker obtains a URL from the Frontier data structure, which dispenses URLs according to their priority and to politeness policies. The worker thread then invokes the HTTP fetcher. The fetcher first calls a DNS sub-module to resolve the host component of the URL into the IP address of the corresponding web server, and then connects to the web server, checks for any robots exclusion rules, and attempts to download the web page.

Web crawlers download web pages by starting from one or more seed URLs, downloading each of the associated pages, extracting the hyperlink URLs contained therein, and recursively downloading those pages. Therefore, any web crawler needs to keep track both of the URLs that are to be downloaded, as well as those that have already been downloaded (to avoid unintentionally downloading the same page repeatedly). The required state is a set of URLs, each associated with a flag indicating whether the page has been...
downloaded. The operations that must be supported are: Adding a new URL, retrieving a URL, marking a URL as downloaded, and testing whether the set contains a URL. There are many alternative in-memory data structures that support these operations. However, such an implementation does not scale to web corpus sizes that exceed the amount of memory available on a single machine. To scale beyond this limitation, one could either maintain the data structure on disk, or use an off-the-shelf database management system. Either solution allows maintaining set sizes that exceed main memory; however, the cost of accessing items in the set (particularly for the purpose of set membership test) typically involves a disk seek, making it a fairly expensive operation. To achieve high performance, a more specialized approach is needed.

II. SURVEY REVIEW

1. Kevin Chen-Chuan Chang observes that, across subsystems, the system integration of an integration system is itself non-trivial— which presents both challenges and opportunities beyond subsystems in isolation. On the other hand, he also observe that, across subsystems, there emerge unified in-sights of “holistic integration”— which leverage large scale itself as a unique opportunity for information integration.

2. 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 machines to crawl the web and it crawled on 100 million URLs. Each crawler process reads 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.

3. Heydon and Najork present’s 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.

4. Yan et al. describe’s URLbot, 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. In addition, the authors address the issue of crawler traps and propose ways to ameliorate the impact of such sites on the crawling process.

5. Brin and Page’s 1998 paper outlining the architecture of the first-generation Google system contains a short description of their crawler. The original Google crawling system consisted of a single URLserver process that maintained the state of the crawl, and around four crawling processes that downloaded pages. Both URLserver and crawlers were implemented in Python.

6. Shkapenyuk and Suel’s Polybot web crawler represents another “blueprint design.” Polybot is a distributed system, consisting of a crawl manager process, multiple
downloader processes, and a DNS resolver process. The paper describes scalable data structures for the URL frontier and the “seen-URL” set used to avoid crawling the same URL multiple times; it also discusses techniques for ensuring polite-ness without slowing down the crawl. Polybot was able to download 120 million pages over 18 days using four machines.

III. TWO-STAGE ARCHITECTURE

SmartCrawler is designed with a two-stage architecture, site locating and in-site exploring. The first site locating stage finds the most relevant site for a given topic, and then the second in-site exploring stage uncovers searchable forms from the site.

Generally, the site locating stage starts with the set of sites in database. Seeds sites are candidate sites given for SmartCrawler to start crawling, which begins by following URLs from chosen seed sites to explore other pages and other domains. When the number of unvisited URLs in the database is less than a threshold during the crawling process, SmartCrawler performs reverse searching of known deep web sites for center pages and feeds these pages back to the site database. Site Frontier fetches homepage URLs from the site databases, which are ranked by Site Ranker to prioritize highly relevant sites. The Site Ranker is improved during crawling by an Adaptive Site Learner, which adaptively learns 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. The most relevant site is found in the first stage.

![The two-stage architecture of SmartCrawler](image)

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 and embedded forms are classified by Form Classifier to find searchable forms. Additionally, the links in these pages are extracted into Candidate Frontier. To prioritize links in Candidate Frontier, SmartCrawler ranks them with Link Ranker. Note that site locating stage and in-site exploring stage are mutually intertwined. 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.

IV. THE PROCESS OF SITE LOCATING

The site locating stage finds relevant sites for a given topic, consisting of site collecting, site ranking, and site classification.

1. **Site Collecting**—SmartCrawler strives to minimize the number of visited URLs, and at the same time maximizes the number of deep websites. To achieve these goals, using the links in downloaded webpages is not enough. This is because a website usually contains a small number of links to other sites, even for some large sites.

2. **Site Ranking**—In SmartCrawler, Site Ranker assigns a score for each unvisited site that corresponds to its relevance to the already discovered deep web sites.

3. **Site Classifier**—After ranking Site Classifier categorizes the site as topic relevant or irrelevant for a focused crawl, which is similar to page classifiers in FFC and ACHE.

V. CONCLUSION

In this review paper, finally all the survey is related to SmartCrawler which 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. SmartCrawler is a focused crawler consists of two stages: site locating and balanced in site exploring. SmartCrawler 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. SmartCrawler 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.

VI. REFERENCES


To Cite This Article