A scalable lightweight distributed crawler for crawling with limited resources

Milly W. Kc
University of Wollongong, millykc@uow.edu.au

Markus Hagenbuchner
University of Wollongong, markus@uow.edu.au

Ah Chung Tsoi
Hong Kong Baptist University, act@uow.edu.au

A scalable lightweight distributed crawler for crawling with limited resources

Abstract

Web page crawlers are an essential component in a number of Web applications. The sheer size of the Internet can pose problems in the design of Web crawlers. All currently known crawlers implement approximations or have limitations so as to maximize the throughput of the crawl, and hence, maximize the number of pages that can be retrieved within a given time frame. This paper proposes a distributed crawling concept which is designed to avoid approximations, to limit the network overhead, and to run on relatively inexpensive hardware. A set of experiments, and comparisons highlight the effectiveness of the proposed approach.

Keywords

scalable, lightweight, distributed, crawler, for, crawling, limited, resources

Disciplines

Physical Sciences and Mathematics

Publication Details

Kc, M. W., Hagenbuchner, M. & Tsoi, A. (2008). A scalable lightweight distributed crawler for crawling with limited resources. IEEE/WIC/ACM international Conference on Web Intelligence and Intelligent Agent Technology (pp. 663-666). Sydney, Australia: IEEE Computer Society.

A Scalable Lightweight Distributed Crawler for crawling with limited resources

Milly Kc
University of Wollongong
Wollongong, Australia
millykc@uow.edu.au

Markus Hagenbuchner
University of Wollongong
Wollongong, Australia
markus@uow.edu.au

Ah Chung Tsoi
Hong Kong Baptist University
Hong Kong
act@hkbu.edu.hk

Abstract

Web page crawlers are an essential component in a number of web applications. The sheer size of the Internet can pose problems in the design of web crawlers. All currently known crawlers implement approximations or have limitations so as to maximize the throughput of the crawl, and hence, maximize the number of pages that can be retrieved within a given time frame. This paper proposes a distributed crawling concept which is designed to avoid approximations, to limit the network overhead, and to run on relatively inexpensive hardware. A set of experiments, and comparisons highlight the effectiveness of the proposed approach.

1 Introduction

Information retrieval through crawling the Internet is commonly performed for a variety of tasks, for instance:
Research purposes: Snapshots of the Internet are often created for statistical or other research purposes. A snapshot provides a static dataset which greatly simplifies the analysis, or the development of new methods, compared to the evaluation on the dynamic Internet.
Web search services: The crawler is one of the core modules of Internet search engines. Google, for example, continuously retrieves very large portions of the Internet. These pages, once crawled, are indexed in a large database and are then processed by a variety of filters and ranking methods.

Crawling is not a complex process, however, challenges often arise during a crawl due to the amount of data, the variation in the types of information crawled, the dynamic nature of the web, and communication disturbances. There have been much research efforts into crawling methods [1, 2, 3, 5, 6], with particular focus on improving the crawling efficiency; as efficient information retrieval methods are crucial in obtaining a reasonably large amount of data within a short time frame from the ever expanding World Wide Web. It has been shown that it is feasible, using a standalone system, to retrieve a large amount of data from the Internet [6] with the support of high-end hardware and large supply of resources, e.g. high network bandwidth. However, it is not sufficient to address crawling efficiency alone; the crawling effectiveness should also be addressed for the following reasons:
1.) It is evident that there is a large amount of data duplication due to mirror sites, dynamic scripts, links in directory structure and other replication utilities. This data duplication takes up a crawler's processing time and resources unnecessarily; it could even trap a crawler in an infinite loop. An effective crawler should exhibit mechanism to detect and avoid this. However, there is a trade-off between the amount of processing the crawler is required to perform, and its crawling efficiency. Therefore current crawlers do not directly address data duplication issues. The crawler proposed in this paper aims at avoiding crawling duplicated data without affecting the crawling efficiency significantly.
2.) Researchers often have limited resources available to support information retrieval experiments. A crawler that supports information retrieval tasks with minimal resource requirement would be of value. Although this can be more challenging in a distributed crawler, as communication between nodes is required, it is shown in this paper that it is possible to make much more efficient use of network bandwidth when compared to a stand-alone crawler. A distributed approach allows parallelism, and hence, this can reduce dependencies on powerful computing resources.

A number of challenges with web crawling and some novel approaches are addressed in Section 2. The focus is on crawling effectiveness rather than on crawling efficiency. Evaluations are made according to the amount of resources consumed during the course of a crawl, and the usefulness of the crawling result. The crawler proposed in the paper is named LiDi Crawl, and is described in some detail in Section 3. Some experimental results are presented and analyzed in Section 4. Finally, Section 5 offers a conclusion.

2 Crawling challenges

The unregulated nature of the Internet has introduced a number of challenges for web crawling. Some of these challenges are well-known, such as leniency for HTML syntax, bad responses from domain hosting server, etc. This section addresses some of the less frequently discussed challenges encountered during a crawl, and proposes some novel solutions which help overcome the problem.
Non-terminating crawling loop: Non-terminating crawling loop can be caused by symbolic links, server-side

scripts, web spam and a virtually unlimited number of hostnames per domain. This is a serious issue, as retrieving multiple copies of the same page can significantly waste network resources, and processing time.

The most commonly adopted solution is to restrict the directory depth (i.e. [1, 4, 5]), or to limit the number of web pages retrieved from a site (see e.g. [6]). Although these approaches prevent crawling in a non-terminating loop, they do not prevent the crawler from retrieving multiple copies of a page. The solution proposed in this paper is to detect duplicated web pages by firstly comparing directory structure and then find the degree of similarity in the content for suspected pages. Pages with more than 95% of content similarity would then be labeled as duplicated page. If multiple web pages from a domain or a directory have been identified as duplicated copies, the site (or directory) is labeled as a mirror, in which case the crawler would not crawl further into the domain or directory. It will be shown that this approach is very effective.

DNS lookup time consumption: It is important to minimize DNS lookup to allow optimal crawling result. Tests show that the average time taken for the retrieval of an IP address from a local database is ≈ 0.00158 seconds, whereas the average retrieval time of an IP address from the DNS server is on an average 1.661 seconds [1]. Given that the large majority of URLs require a DNS lookup, the speed impact is significant. The proposed approach stores the lookup results in a central location, and a DNS lookup is only conducted when an unseen domain is encountered. Note that the approach is more efficient than DNS caching by an operating system. This is due to the fact that the approach exploits the working of the proposed crawler which engages a depth-first crawling approach; it crawls into a domain before crawling other domains. It can be assumed that the DNS rule is unlikely to change during the crawl of a domain, and hence, the proposed approach is not required to regularly check for updates of a DNS entry.

File type misinterpretation: File extension, or the http response header provide basic but important information about the content of a page, especially when efficiency and appropriate resource allocation are the major crawling concerns. For example, the ".html" extension, or the file header content description "html/text" indicates a static web page most likely to be constructed by ASCII characters, whereas the ".exe" extension, or associated file header content description indicates an executable file of binary content. These file type information help crawlers to determine whether a file is worth crawling before its actual retrieval. In recent years, as part of the dynamics of the Internet, leniency is introduced to web hosting servers in order to accommodate the rapidly increasing variety of file types. As a result, file extensions and file header description provide a less accurate indication of the file type, and therefore older versions of crawler can fail more often due to file type mis-interpretation. A proposed solution is to monitor data contained in a page while it is loaded to confirm the file type. The loading of the file is interrupted immediately if retrieved data do not match the indicated file type. This approach has been particularly effective with domains which create content dynamically.

A crawler designed to address these issues, and to provide an effective and low cost solution to crawling is described in Section 3.

3 The proposed LiDi Crawl crawler

LiDi Crawl (short for **Li**ghtwight **Di**stributed **Crawl**er) is a centralized distributed crawling application, and comprises of a central node (the master) that manages the individual crawling components (the slaves). LiDi Crawl incorporates the approaches mentioned in Section 2, therefore is addressing its goal on effectiveness. The distributed design described in the following reduces dependency on expensive resources (hardware and network).

LiDi Crawl can be executed either locally on a single machine, or as a distributed crawling application with machines located globally. For both of these configurations, there are a number of crawling options that can be used.

URL only method. The crawling component retrieves a set of pages for a given set of URL provided by the master. This is a common strategy for distributed crawlers but the communication overhead is usually high for this method.

Rule-based method. The proposed crawler assigns a set of seed pages belonging to one domain to an available slave component. Efforts are made to allocate domains to a slave which is the closest physically located relative to the slave. This reduces network traffic and costs by aiming at crawling locally available sources (causing costs for local or national rather than international internet traffic). The slave will then crawl all pages in that domain which are reachable from the given set of seeds. This approach can greatly reduce the network overhead and costs due to (a) the master is only required to send a relatively small set of seed pages, and the IP address of the associated domain, (b) the slave crawls pages from a domain which is located physically close. In the best case, the crawler and affected web server both reside on the same local network. The slaves compress the crawled data using the well-known z-lib library before sending it back to the master, and hence, the traffic at the master node is by a magnitude smaller when compared to stand-alone crawlers. This approach provides a clear partitioning of the Internet into domains which eases the assignment task to the crawlers, which can crawl independently once the set of seeds is received. Note that the slaves extract hyperlink information embedded in the pages retrieved. These are compressed and returned to the master so as to allow for the maintenance of a central database containing complete information about the current knowledge of the Internet at any given time.

[1]Different hardware might have different figures for these retrievals of IP addresses. But the relative speedup as shown remains valid.

The master maintains information about known domains including domain name, IP address, and status (either scheduled, currently being crawled, or crawled). The master is the only location at which DNS look-up is performed, and the result of the DNS look-up is cached and passed to the crawling components, so that no crawling components need to perform a DNS look-up. The central node is also the only node with knowledge of all the URLs, as this information is required for the allocation of seed URLs to the participating slaves.

The independent crawlers (slaves) have several features that are important for an effective crawling process. These include **compression** of data on the fly to decrease storage requirement, **loop detection** to reduce wasting resources on duplicated data. The latter allows crawling to an **unlimited directory depth**. To the best of our knowledge, there is no other general purpose crawler armed with such a feature. **State safety** allows slaves to halt crawling at any time then continue from where it left off, and **fault tolerance** to prevent loss of data in cases of communication interruption with the master.

The proposed crawler is scalable in that many slaves can be added to participate in crawling for the retrieval of a large amount of data. Assuming that the slaves reside on separate standard machines with access to acceptable network bandwidth, the overall performance is expected to be as many times faster as the number of slaves, even with a large number of slaves, because the communication with the central node (master) is minimal. "Light weight" here refers to the impact of the crawling application on resources, where dedicated super-computer and high bandwidth are not necessities, in contrast to e.g. [6].

4 Experiments

During the development of the crawler, several experiments were conducted to observe its performance. In particular, experiments were conducted to observe the amount of improvement in crawling efficiency under a distributed setting, and the degree of crawling effectiveness using content based depth restriction.

Impact on network and hardware resources: It is commonly believed that a distributed system is more efficient than a single sequentially executed crawling process because a distributed system allows multiple processes to run simultaneously over a number of hardware locations. However, a distributed system also has drawbacks. The management and communication between the processes may result in a complex system that requires additional tasks that it is no longer efficient. The tests carried out aim to identify the difference in crawling performance between a single process and a distributed crawling application. An experiment aimed to identify the threshold point where it is no longer beneficial to crawl in parallel.

The crawling slaves have a "polite" feature to avoid overloading servers with requests, which means, there are delays

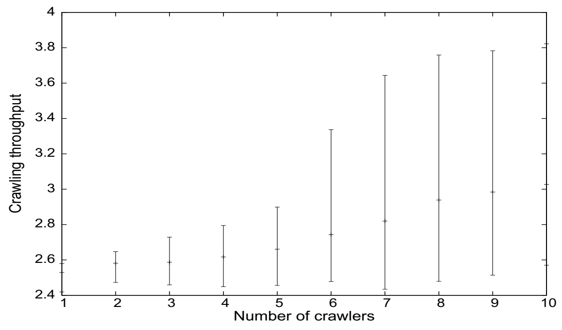

Figure 1. Crawling throughput versus the number of slaves crawling the same domain.

in between the crawling of each page. Since it was not possible to access a domain host server's performance statistics while crawling was taking place, an attempt was made to increase the number of total processes that is crawling the same domain, to observe the likelihood of the crawling task congesting the network for the hosting domain. For this experiment, a number of tests were run on different days of the week and at different times of the day, and for various numbers of simultaneous crawlers. Figure 1 shows the min, max, and average throughput values for this test.

Figure 1 shows that in the best case (max throughput is obtained when server is not busy serving other users during a test) there is no significant decrease in crawling efficiency. There is only a slight decrease when the number of simultaneous crawlers reaches 9 and 10. However, in the worst case (i.e. server is busy) the throughput can decrease quite dramatically with an increase of simultaneous crawler. Thus, the impact of the crawler on the hosting domain is heavily dependent on the network condition at the time.

Furthermore, in the graph, a trend of gradual decrease of the average throughput seems to form with the increasing number of crawlers, this decrease is more apparent for 8 or more crawlers. Considering the scale of the throughput in the graph, this decrease in the average throughput is not significant, and shows that even with 10 crawlers all accessing the same hosting domain simultaneously, it would not cause significant network congestion.

For a second test, a set of domains is crawled using a varying number of slaves running on the same machine. The domains chosen are located either local, domestic, of international. This is to investigate the impact of the crawlers on resources available to a (slave) machine, and to investigate the scalability when considering a single machine only. The resources monitored are CPU load, and network bandwidth. The result is shown in Table 1.

Note that the slaves engage a "server friendly" crawling technique by pausing 1.8 seconds between the retrieval of pages. Thus, 1.8 seconds can be deducted from the times indicated in Table 1 in order to obtain the actual time requirement for loading, hyperlink extraction, and data com-

Table 1. Throughput vs. number of slaves.

Locality	1 slave	2 slaves	3 slaves	4 slaves
Local	2.507 sec/pg	2.549 sec/pg	2.571 sec/pg	2.578 sec/pg
Domestic	2.815 sec/pg	2.761 sec/pg	2.800 sec/pg	2.848 sec/pg
Interntl.	3.248 sec/pg	3.086 sec/pg	3.116 sec/pg	3.131 sec/pg

pression. It was observed that during data compression that the CPU is fully utilized albeit for a very short time. On average, the CPU load was observed to be about 15% on a 2GHz intel based CPU. It can be observed from Table 1 that when using a single slave in crawling web sites within the same locality (country) or internationally is slower than using 2 slaves in a distributed system.

Effectiveness: Page redirections, domain mirroring, symbolic links within the domain structure can all result in redundant web pages. The amount of resource saved by avoiding crawling duplicated web pages is dependent on the directory structure of each domain. This focus of retrieving the complete set of unique web pages within a domain during crawling is for achieving effective crawling. In a test conducted on www.crown.net - a domain with hidden redirections demonstrated the extent of redundancy reduction that could be achieved using the recursion detection feature in crawling. During the test, crawling was conducted twice, once with the fixed-depth set to 10, another with the recursion detection feature proposed in Section 2. The fixed depth crawling took a little over 34 hours, crawling 26,442 pages, whereas the proposed recursion detection crawling only took about 8 hours to crawl 8,180 pages, eliminating crawling of 18,262 redundant web pages. Therefore, the reduction that can be obtained using the proposed recursion detection is significant.

In another test, the effectiveness of crawling was examined from a different perspective by focusing on crawling coverage. This is to assess the proposed recursion detection approach. For this experiment, we considered the domain www.popularmechanics.com which is found to use deep directory structures extensively. In the test, 37,165 web pages were crawled, and 24,775 of them have URLs with a depth of more than 10. This finding shows that common fixed-depth crawlers would miss all these unique web pages.

Most crawlers use a fixed-depth method of crawling, where depth is set to 10 or 11 (eg. Google's crawling agent). This renders such crawlers clueless to whether the web page belongs to a recursive directory, a mirrored directory, dynamically created redundant content, or genuine content. One may argue that the amount of unnecessary data obtained through crawling recursive directories, up to a fixed depth, is insignificant. In a dataset crawled in December 2005 using a fixed-depth approach where depth restriction was set to 10, it was found that 8.95% of crawled pages were duplicates. The proposed approach of loop (and mirror) detection eliminates almost entirely the problem at a cost of 2 milliseconds of CPU time per page on average.

Comparisons: The crawler's effectiveness is examined using *precision*. Crawling precision is calculated by dividing the number of unique pages by the total number of web pages retrieved by the crawler. It has been noted that there is a trade-off among crawling speed, precision and recall.

A recently proposed crawler [6] reported a crawling precision of 0.16. This low precision may be due to its optimization towards crawling at maximum speed. A system proposed in [1] produced a precision of 0.2569. The LiDi Crawl is able to achieve a precision of 0.8371. This demonstrates the effect of the loop and duplication detection, and file type misinterpretation meassures proposed in this paper, and was achieved after having crawled over 27 million pages. Although duplicated data is avoided, the precision still cannot achieve the perfect 1. This is because suspected pages, although not stored, still need to be retrieved in order to verify its uniqueness. An entire mirroring site or replicated directory will only be avoided once 3 duplicated web pages are identified. Nevertheless, it is observed that the crawling precision achieved by LiDi Crawl is significant.

5 Conclusion

The crawler implemented for the research project focuses on crawling effectiveness for the minimization of resource consumption. The result is a scalable, effective and reasonably efficient crawling application. As the experimental results show, the features in LiDi Crawl significantly improves the completeness and accuracy of crawling by minimizing wastage on redundant crawling. The recursion detection feature in LiDi Crawl is an important addition that does not exist in other known crawlers. It was shown that recursion detection method outperforms its fixed-depth counterpart in 3 major aspects: effectiveness, redundancy reduction, and retrieval coverage.

References

[1] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. In *AusWeb '02: Proceedings of the 8th Australian World Wide Web Conference*, Sunshine Coast, Queensland, July 2002.

[2] J. Cho and H. Garcia-Molina. Parallel crawlers. In *WWW '02: Proceedings of the 11th International Conference on World Wide Web*, Hawaii, USA, May 2002.

[3] J. Edwards, K. McCurley, and J. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *WWW '01: Proceedings of the 10th International Conference on World Wide Web*, Hong Kong, May 2001.

[4] Google Inc. Google webmaster central blog: Official news on crawling and indexing sites for the google index. [online] http://googlewebmastercentral.blogspot.com/-search/label/crawling and indexing, 2008.

[5] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1978.

[6] H.-T. Lee, D. Leonard, X. Wang, and D. Loguinov. Irlbot: Scaling to 6 billion pages and beyond. In *WWW '08: Proceedings of the 17th International Conference on World Wide Web*, pages 427–436, Beijing, china, April 2008.