CrawlWave: A Distributed Crawler

Apostolos Kritikopoulos[1], Martha Sideri[1], Kostantinos Stroggilos[1]

[1] Dept. of Computer Science, Athens University of Economics and Business, Patision 76, Athens, T.K.10434, Greece
apostolos@kritikopoulos.info, sideri@aueb.gr, circular@hol.gr

Abstract. A crawler is a program that downloads and stores Web pages. A crawler must revisit pages because they are frequently updated. In this paper we describe the implementation of CrawlWave, a distributed crawler based on Web Services. CrawlWave is written entirely in the .Net platform; it uses XML/SOAP and is therefore extensible, scalable and easily maintained. CrawlWave can use many client and server processors for the collection of data and therefore operates with minimum system requirements. It is robust, has good performance (download rate) and uses small bandwidth. Data updating was one of the main design issues of CrawlWave. We discuss our updating method, some bottleneck issues and present first experimental results.

1 Introduction

The size of the World Wide Web has grown remarkably since its first appearance in the beginning of the 90 s. Because of this rapid increase of the available Web pages, the use of the search engines as the mean for discovering the desired information is continuously becoming more imperative. Search Engines are based on vast collections of Web documents ([2],[4],[11],[16],[17]) which are gathered by Web crawlers. Web crawlers are applications that visit the Web, following the hyperlinks within pages, and store the page contents in a data repository. The pages are then analyzed and organized in a form that allows the fast retrieval of information.

The distributed crawlers that already exist ([1],[3],[7],[8],[15],[20]) communicate via proprietary means (peer to peer, port communication, TCP/IP and HTTP calls), which many times cause problems if the client agent is stopped by firewalls or proxies. In most cases, the clients of these systems are fully dependent with the server and cannot work in an automated way (specific installation process is needed, the client must exist in the same network as the server, the update of the client version is not automated etc.).

We propose a distributed crawler system (CrawlWave) which interacts with its detached clients via SOAP and XML. The interface of the main crawling engine is published via a Web Service. In contrast with the major search engines that use dedicated agents for the crawling process ([7],[12]), our purpose is to distribute the load at multiple client PCs over the Internet and use their processing time and internet connection bandwidth.

The design of the CrawlWave was influenced -but not completely based- on the methodology of the most famous peer-to-peer distributed systems like SETI@home [14]. Our basic goals were to achieve persistence of the connection between the client and the server (HTTP and XML/SOAP), to publish an open interface of the server functionality (Web Services) that anyone could use, to ensure the consecutive and persistent process of the crawling, to automate the upgrading of the client module and to use a Relational Database (Microsoft SQL Server) for the storage of the data. We used state-of-the-art development tools (Microsoft.Net platform, C#, ADO.Net) to achieve great performance. Our initial objective was to use the CrawlWave for the concentration of the Greek Web pages, with aim to create a data depository that could be used for research purposes.

The following paper is organized as follows: Section 2 presents an overview of the functional requirements and the desired architecture of the system. Section 3 analyzes the interface and the implementation of the CrawlWave. Section 4 presents some preliminary experimental results. Finally Section 5 summarizes our conclusion.

2 Functional requirements of the CrawlWave

The structure of the CrawlWave has been designed in such way that it offers efficiency along with easy management and maintenance. A lot of effort has been given to improve the fault tolerance of the system (regarding possible errors of the software and the hardware). In this part we describe the functional requirements that the system must meet. A Web crawler ought to have great performance in order to visit all Greek Web domain (gr) in reasonable time. Based on the fact that the Greek domain has already a size of almost 2400000 URLs [7], and the last 5 years it grows rapidly, is substantial that a Web crawler should be able to visit the Web documents with significant speed (hundreds of pages per minute).

Data Repository Efficiency: The database could be organized in various ways in order to store data with advanced speed. As it is mentioned in [4] the direct storage of entire pages in disk, in the form of a sequence of objects (list or a log) in conjunction with a data structure such as a B-tree index, is the most rapid solution. However this implementation leads to difficulties in maintenance, especially when updating the items of the collection. The required solution is to use a relational Database in order to increase the efficiency and the availability of the stored data.

Download Module: During the past we tried many times to use a single computer to crawl the Greek Web, but we always encountered many problems (power failures, decreasing speed due to the growth of the indices, unhandled exceptions). In most cases the process was cancelled and many crawled data were lost. Based on these facts, we decided to develop a decentralized crawler, which could operate regardless of the errors that might occur, the connection speed, and the efficiency of the database. A server/coordinator has to distribute to the various client modules a list of Web pages to be visited. In that case the bandwidth is not shared between the client modules, but every module could use its own bandwidth. During the visit of different Web pages, the client module will probably meet enough obstacles (for example, many Web servers could be unavailable). To skip this problem a multithreaded approach must be employed; the client has to interact with multiple Web servers simultaneously using different threads. With this approach we isolate any possible problems; in case of a local error, only one thread will freeze for a short amount of time- but all the rest will continue to operate.

User interface: One of the more basic elements that the system supervisor should have access to, is the statistic report of the crawler s progress. The supervisor must be notified immediately of any problems that could delay the crawler s process. A simple Web page that shows the current activity is considered adequate. The client modules must not consume a lot of bandwidth at the local computer. For this reason the client module must be adjusted to function proportionately with the local system's capabilities.

Fault tolerance: Resistibility in faults is one of the elements that characterize the qualitative software. A crawler handles huge amounts of data (tens of GB) and it is extremely essential that the integrity of these data to be independent of any possible errors. It is also very important that if the crawling process stops for some reason, then it should continue from the point that it halted. In the worst case, the data loss should be limited in the loss of some few pages. The system should be compliant with the model ACID (Atomicity, Consistency, Isolation, Durability) for any action that is related with the storage of the data.

Scalability: Greek Web is increasing rapidly. That means that a Web crawler should use all the capabilities of the hardware, in order to cope with the ever increasing load. The system architecture should allow multiple (redundant) instances of its units to operate simultaneously. If one module collapses, then the crawler should be able to continue its operation even with lower speed.

3 Structure of CrawlWave

The basic modules of the CrawlWave are the Data Depository, the DBUpdater, the CrawlWave Server, the CrawlWave Client and the VersionChecker. The following diagram presents the units of the system, and the way that communicate with each other:

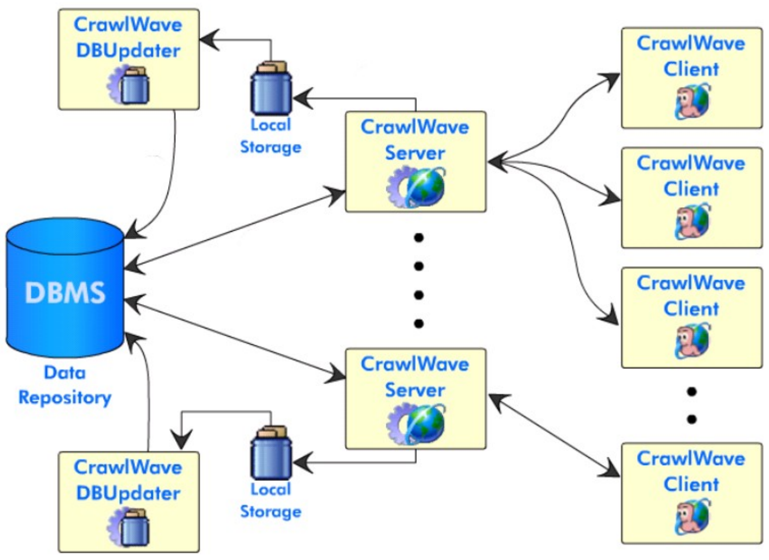

Fig. 1. Architecture of CrawlWave

In the diagram we can see that the only unit that is unique (Singleton) in the system is the data repository. All the other units (Server, DBUpdater and the Client) can exist in many instances. Each CrawlWave Server coexists on the same workstation with the DBUpdater (even though this is not necessary), while the Clients are completely independent. In the next section we analyze each of CrawlWave's main components.

3.1 Data Repository

CrawlWave uses a relational database (Microsoft SQL Server) for saving all the data it collects through the crawling of the Web. A more low-level solution (like using proprietary files based on B-Trees or custom hash tables) could probably be faster. However this approach would be more difficult in the field of administration, report generation and resource management. The database is not only used to store the internal data of each Web page, but also the link graph ([5],[10]) of the hyperlinks. We also store important information about the available CrawlWave Servers, the PCs that the clients operate on, and the history (log) of every function that a client uses.

3.2 Client-Server approach

Many distributed applications ([14]) use internet sockets to communicate with each other (or with their server). This approach is very fast compared to HTTP text based protocol but is extremely error-prone. Many times the Internet Proxy or the Firewall/NAT of the client system restricts the access to any non well-known ports. That means that the communication via ports other than HTTP (port 80) is not always available or reliable. For this reason CrawlWave is based on SOAP (Simple Object Access Protocol), a protocol that allows access to remote objects transparently using XML documents [9]. The following diagram illustrates the communication between client and server with the exchange of SOAP messages over the Web.

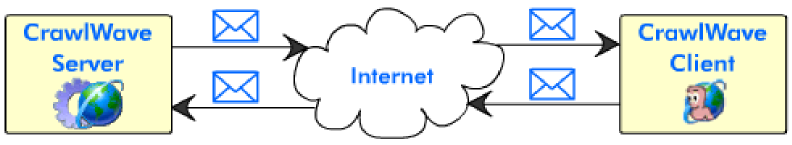

Fig. 2. XML messages between the client and the server of the CrawlWave

For the development of all the modules of the crawler, we used Microsoft Visual Studio .Net 2003, and the C# programming language. In the next section we analyze each of the system s modules.

3.2.1 The common library of the CrawlWave

It was a necessity to use common data structures for almost all modules. We encapsulated the common functionality and the shared objects into a DLL (Dynamic Link Library), in order to avoid the repetition of programming code.

3.3 CrawlWave Server

The main task of the CrawlWave Server is to distribute the URLs that must be visited to all the clients, and to gather all the information that the clients have collected about the requested Web pages. In Fig.3 we can view its basic modules and functionality.

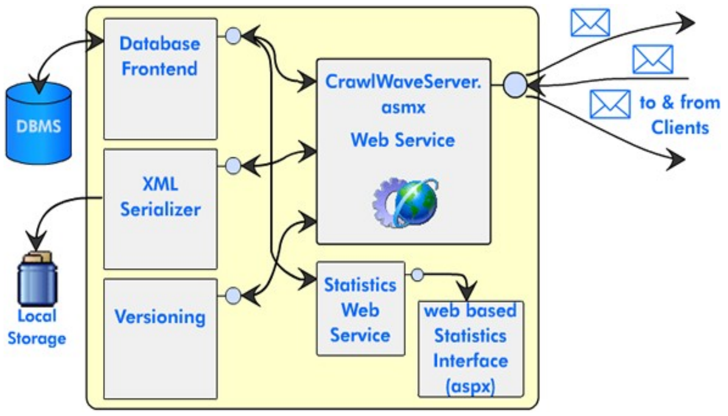

Fig. 3. The architecture of the CrawlWave's server

The server was developed as an ASP.NET XML Web Service [19]. The transactions of the data are based on SOAP protocol and they are performed over HTTP. The public interface of the service is an XML document (WSDL schema) that is available to everyone who has access to the Internet. The specification of the interface XML document is well defined in the Web Services Protocol designed by the World Wide Web Consortium [18], and can be used by any development tool that supports Web Services (such as Microsoft.Net, Java, Delphi etc). Therefore any third development party, who wants to use the methods of the CrawlWave Server, can obtain its interface and use its functionality by just reading the Web Service description file. The CrawlWave Server therefore is simply a public class with well-defined methods that can be called by any remote client PC.

For the naming of the methods, we used a server-initiated naming convention; the prefix send means that the server is sending data to the client, and get means that the server receives information from the client. We briefly describe the functionality of each method of the server object:

– SendFirstClientInfo: This method is called the first time a client is connected with the system. The server creates a unique identifier for the client, and returns it.
– SendServers: It sends a list of the available servers and their version to the client.
– SendUrlsToCrawl: The server creates a list of URLs to be visited, and sends them to the Client. It gives top priority to the URLs that have never been visited. The method used for selecting the URLs is the one of a random walk, though the system can very easily be configured to use a BFS approach.
– GetCrawlResults: Updates the repository with the data returned by a client.
– GetClientComputerInfo: It is used the first time that a client communicates with a server, to get vital client information (type of hardware, connection speed).
– SendCurrentVersion: The client uses this method for version control. If a newer version of the client is available the crawling process stops and the new version is downloaded from the server.
– SendVersionFile: Sends the latest version of the client s executable.
– IsAlive: It is used only to certify that a specific server is in a functional state.

The system also publishes another (supplementary) Web Service, with the name CrawlWaveServerStats. This object contains methods that can be used for statistical reports regarding the topology of the crawled Web, the current crawling speed etc.

3.4 DBUpdater

This module is responsible for the proper update of the Web Pages and the link graph contained in the data depository. The architecture is very simple:

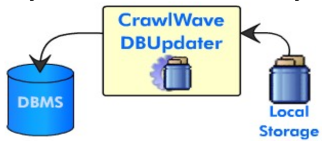

Fig. 4. The functionality of DBUpdater

It can be installed in the same server with the CrawlWave Server, but that is not necessary. Its main function is to read periodically the data returned by the clients, and update the database. The data are stored as XML files from the CrawlWave Server at a local temporary disk. Whenever DBUpdater finds a new XML file, it updates the database with the data it contains and then deletes it. Basically the temporary directory is a queue, and the XML files represent packets of information that must be inserted in the Database. The reason that we used the DBUpdater, and we don t update the database in a synchronous way, is because during our tests the delay of this function was extremely aggravating. It was a necessity to update the database in bulk mode, using an asynchronous method.

3.5 CrawlWave VersionChecker

VersionChecker is one of the three main modules that compose the Client part of the system (the other two are the CrawlWave Client and the CrawlWave Uninstaller). It has three main functionalities:

– Initialization: The first time the client application is launched, a unique identifier must be retrieved, so that the CrawlWave Client can operate.
– Scheduling: It launches the client application at specific time of the day, which is parameterized by the user.
– Versioning: Every time a new version is released, the VersionChecker downloads it from the CrawlWave Server, and replaces the old one at the client PC.

VersionChecker was developed as a Windows Service, because it has to execute continuously, and it has to check the status of the Client at specific time intervals.

3.6 CrawlWave Client

CrawlWave Client is the most important application of the client module, because its main functionality is to crawl the Web pages, analyze their contents and return them back to the CrawlWave Server.

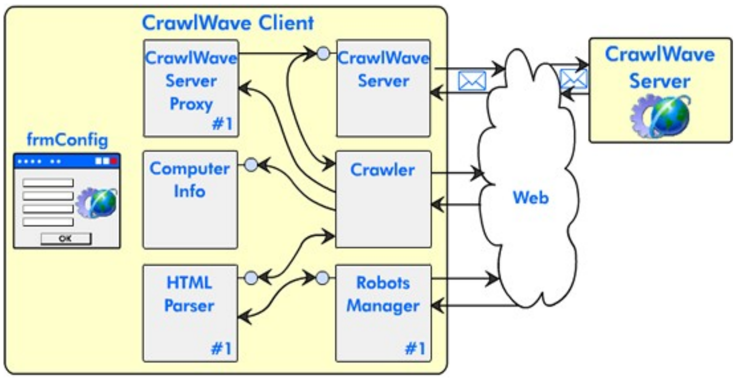

Fig. 5. The modules and the operations of the CrawlWave's client

The only functionality of the client that is irrelevant with the crawling of the URLs, is the GUI (Graphical User Interface) that the user operates to declare important information about the local PC and to adjust the Scheduler of the client. The client after the establishment of the connection with an active server (by calling the server method SendServers), it calls the server method SendUrlsToCrawl to get a list of URLs to visit. After the parsing of the crawled URLs, it returns all the necessary information to the server by using the Web method GetCrawlResults. The parsing of the Web pages and the extraction of the URLs is performed with the use of regular expressions. This way we can extract hyperlinks found in the body of a page, in framesets or form actions. The URLs are then parsed in order to be converted to canonical form and the number and type of their parameters (if any) is examined in order to remove redundant information. The parameters are sorted alphabetically and any session ids are removed using a heuristic function. The types of session ids this function can recognize include hexadecimal strings (traditionally used by languages such as PHP or JSP) and GUIDs. For example the URLs:

http://www.aueb.gr/index.php?sid=1e95ff9dd02161b266b912864a65794d
http:// www.aueb.gr/index.php?sid=bdbed13641ec7a12c2201295598be832
http:// www.aueb.gr/index.php

are dynamic html pages, and they are basically the same, because the first two contain the Session Id of the current connection in hexadecimal form.

The URL extraction and normalization processing speed during our tests varied between 800KB/sec and 3600KB/sec and is a CPU-intensive task. Considering that the average size of an HTML file is less than 20KB, one can conclude that the performance is very adequate. CrawlWave Client also uses the class RobotsManager [13], a Singleton [6] implementation of a FIFO, which is responsible for the manipulation of the robots.txt files. Every time a robots.txt file is visited, the crawler stores its data into RobotsManager. RobotsManager grows up to a specific size, and provides methods to obtain the paths of a domain that must not be crawled. The operational steps of the client are executed continuously in the following order:

1. Examination of the scheduler: The client checks if it is allowed to begin the crawling. If not, then it stops.
2. Request of URLs: It calls the SendUrlsToCrawl Web method to determine the URLs that must be crawled. All of them are gathered in a FIFO queue. For every element in the queue:

– Visit of a Web page: A URL is extracted from the FIFO, and is assigned to an available thread.
– Determine the data to return: If the URL has changed since the last time it was visited (the client examines its checksum) then an *UrlCrawlData* object is created, which contains in compressed form the contents of the page and its out-links.

3. Dispatch of the crawled URLs: When the FIFO is empty, the client gathers all the *UrlCrawlData* objects into a list, and sends them to the server. Afterwards it starts again from the first step.

3.7 CrawlWave Uninstaller

A small application which is executed automatically when the administrator of a workstation tries to uninstall the client module of the CrawlWave.

4. Experimental Results

The time period of the first experimental crawling was from 19 September 2003 to 12 October 2003. Because of the limitations of the academic environment and the technical problems we encountered, the process was executing only for 11 days (during this period, the Internet connection was cut off many times due to problems of the main campus router). We used *only two* clients, and we crawled 526,679 URLs. Finally we collected 496,647 Web pages (almost 1/3 of the Greek Web). Table 1 shows approximate statistics for the crawl:

Table 1. Statistics of the first crawl

Total successful requests	*526,679*	*100%*
200 (OK)	496,647	94.29%
302 (moved temporarily)	84	0.01%
400 (bad request)	24,232	4.60%
403 (forbidden)	1,051	0.19%
404 (not found)	2,943	0.55%
500 (internal server error)	1,338	0.25%
503 (service unavailable)	370	0.07%
Other	14	0.01%

We have also launched a second crawl just to test the speed of the system. We started on 19 October 2003 (12:54) and we stopped it after 19 hours. We used *only one* client executing with 15 threads and the system collected approximately 65,600 Web pages with an average download speed of 0.84 MB/sec (57.54 URLs per minute).

5. Discussion

CrawlWave is a distributed crawler that can operate with minimum system requirements. The crawling of the pages is performed by client PCs which communicate -by using Web Services- with the main server PCs. The operation not only is extremely robust and guarantees the continuous and synchronous function of the crawling, but also is very fast. The collected data are stored in a Relational Database, and by using simple SQL we can easily depict the structure and the link graph of the Web. Any search engine can use the Database of the CrawlWave for performing queries on the Greek Web.

Our preliminary results are quite encouraging, but more tuning is needed to achieve better results. The final target is to be able to support a great number of clients that will be able to crawl the entire Greek web every day.

References

1. S. Abiteboul, G. Cobena, J. Masanes, G. Sedrati: A First Experience in Archiving the French Web. Proceedings of the ECDL 2002.
2. Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, Sriram Raghavan: Searching The Web, ACM Transactions on Internet Technologies, 1(1), June 2001.
3. P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. In Proc. of the 8th Australian World Wide Web Conference, 2002.
4. Sergey Brin and Lawrence Page: The anatomy of a large-scale hypertextual Web search engine. In Proc. of the 7th Intl. World Wide Web Conf., 1998.
5. S. Chakrabarti, B. Dom, R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, David Gibson, J. Kleinberg: Mining the web s link structure. IEEE Computer, 32(8): 60 67, 1999.
6. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns, Elements of Reusable Object Oriented Software. Addison-Wesley, London 1994.
7. Google, http://www.google.com
8. Ankit Jain, Abhishek Singh, Ling Liu: A Scalable Distributed Web Crawler. Technical Report GIT-CC-03-08, College of Computing, Georgia Institute of Technology.
9. D. Jorgensen: Developing Web Services with XML. Syngress Publishing 2002
10. J Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan and Andrew S. Tomkins: The Web as a graph: measurements, models and methods. Proceedings of the 5th International Computing and combinatorics Conference, 1999
11. Apostolos Kritikopoulos, Martha Sideri: The Compass Filter: Search Engine Result Personalization using Web Communities. In Proc. of the 2003 IJCAI Workshop on Intelligent Techniques for Web Personalization (ITWP 2003).
12. Larry Page, Sergey Brin, R. Motwani, T. Winograd: The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford (Santa Barbara, CA 93106, January 1998).
13. The Robots Exclusion Standard, http://www.robotstxt.org/wc/exclusion.html
14. SETI@home, http://setiathome.ssl.berkeley.edu/
15. V. Shkapenyuk, T. Suel: Design and implementation of a high-performance distributed Web crawler. In Proc. of the 18th International Conference on Data Engineering (ICDE 2002), San Jose, 2002
16. SpiderWave, http://195.251.252.44
17. WebBase and related digital library works, Stanford University, http://www-diglib.stanford.edu
18. World Wide Web Consortium, http://www.w3c.org
19. World Wide Web Consortium: Web Services Architecture, Working Draft 08 August 2003, http://www.w3.org/TR/2003/WD-ws-arch-20030808/
20. D. Zeinalipour-Yazti, M. Dikaiakos: Design and Implementation of a Distributed Crawler and Filtering Processor. In Proc. of the Fifth Workshop on Next Generation Information Technologies and Systems (NGITS 2002), Caesarea, Israel, June 2002