Xyro : The Xyleme Robot Architecture

Laurent Mignet, Vincent Aguilera, Sébastien Ailleret, Pierangelo Veltri

INRIA Rocquencourt - BP 105

France

<firstname>.<lastname>@inria.fr

Abstract

In this paper we address the problem of loading data from the web. We present the architecture of Xyro, a crawler designed to fetch data from the Web, and particularly XML data. We describe our experience in designing and implementing this robot, addressing some standard problems in the crawler implementation, and presenting our way to overcome in some of them.

Introduction

With the electronic web revolution all kinds of informations are immediately available through the Web. Nevertheless the number of published documents is growing up in a rapid way, thus that getting accurate informations is often hard.

To overcome this difficulty, search engines appeared very early [24, 18], to help user in founding what exactly he/she is looking for. Search engines as Yahoo, Google or Altavista [27, 12, 11] find web pages using software components called "Robots" or "Crawlers" that cover the web graph and index it. Due to the huge amount of heterogeneous data over the Web, new generation of robots are guided by some navigation policy. A survey of the story of Web crawlers could be found in [15].

In this paper we present Xyro, the crawler designed for the Xyleme system [26]. Xyleme is the result of a research project [26], whose aim was build a dynamic XML data warehouse, to provide high level and new services using the properties of XML, the new format to exchange data over the Web. Thanks to the property of this new standard, Xyleme is able to (i) discovery XML documents over the Web and store locally the documents; (ii) version XML documents; (iii) subscribe users that want to be notified when some document changes (iv) query the Web in a more powerful way than the simple key-word based search engines [2]; (v) classify the documents by semantic using a semi-automatic semantic classification.

In this context, the crawler Xyro, was designed and implemented to find and fetch all the XML pages over the Web associating a discovery/refresh policy to them, to guide the pages acquisition phases (a detailed presentation of the Xyro policy can be find in [19]). The research project lasted from September 1999 to September 2000 and Xyleme [25] is now a start-up company.

In this paper we present the functional architecture of Xyro and we present our experience during the designing. We sketch its high level architecture and give some technical informations that we think can be considered to build a high-performance crawler for a simple PC architecture.

We also describe our approach to implement the navigation policy, spending some time in some aspects for the good designing of a crawler, as (i) the auto-identification of the crawler during the page fetching, (ii) the techniques used to avoid several access to the same site in short time interval (i.e. the "rapid fire" problem) (iii) the politic of following only the allowed path during a crawling phase (i.e. the direction contained in the "robot.txt" file).

Moreover we consider the problem of performance, obtained using a scalable architecture. During the designing phase we consider always this goal in mind.

The paper is organized as follows. Section 1 presents the Xyro's architecture, where Section 2 provides an overview of the different policies used to drive a crawler, and we sketch the policy used in Xyro. We report some experiment results in Section 3 finally we present some possible improvements for current architecture in Section 4.

1 The Xyro Architecture

The whole architecture has been designed following three guide lines: (i) minimize the needed memory size, (ii) prevent I/O blocking, and (iii) use multi-threading strategies. A consequence of multi-threading is that every shared data-structure used by the Xyro components (e.g. FIFOs or Arrays in Figure 2), must be thread-safe.

To reduce the requested memory, we use some standard techniques. Among them, we mention the storage of low priority FIFO data structures in files, using a light buffered mechanism to minimize read/write accesses. This is the case of the URLs FIFO of Figure 2: a data structure containing URLs that can grow up rapidly. Another technique to reduce memory size, is to limit the associated memory to the loaded web pages: an exception is done for XML pages that can use a largest amount of resources.

In this section we present the whole Xyro scalable architecture then we will focus of the functional architecture of one machine.

1.1 Scalability

One of the Xyro specification was scalability. For this purpose we the architecture of the whole system consists of two main parts: (1) a Global Policy Module and (2) a set of crawlers. The first drives the crawlers following some searching policies, where each crawler fetches group of URLs and returns web pages to the external world. The crawlers run on different machines where the Global Policy Module run on one machine; the architecture is given in Figure 1 where M_i indicates a machine. To minimize the communication between the Global Policy Module and the crawlers, a Local Policy Module is plugged inside each crawler to manage local policy and to send to the Global Policy Machine the not treated URLs.

1.2 Functional Architecture.

In this section we present the Xyro functional architecture described in Figure 2. We distinguish between data structure, and threads. Thread processes are represented by ellipses where data structure are represented as arrays of data. For instance, in the

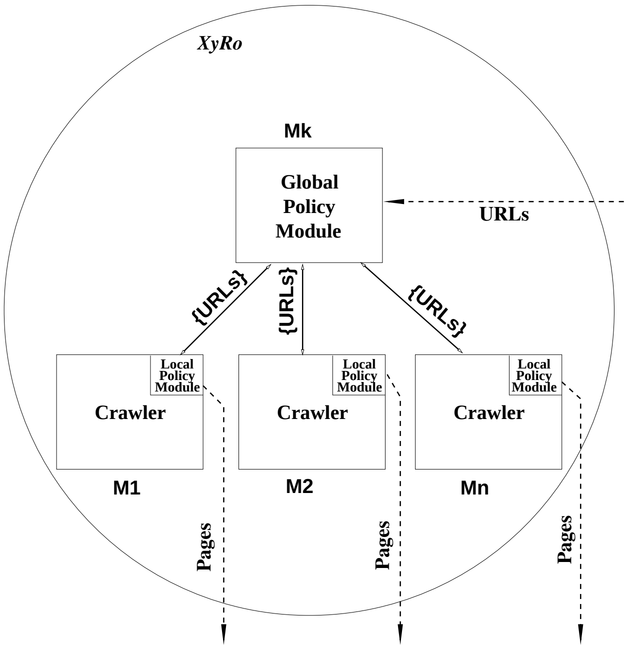

Figure 1: Xyro Scalable architecture

top part of Figure 2 we find two data structures: the URLs Fifo and the Site Array. The former contains a list of URLs, that are the web page identifiers. Each URL is composed of a site name and a path, e.g. in http://www-rocq.inria.fr/˜veltri, the site name is www-rocq.inria.fr and the path is /˜veltri. The Site Array, is an array of site, where a site is a set of URL having the same site name.

During its life cycle, a web page identifier (URL) is added to the URLs FIFO by the Policy Module. Each URL is loaded from the URLs Fifo by the URL Sequencer thread. The Sequencer extracts the site name from the URL and adds the current URL in the corresponding entry in the array of sites (Site Array in Figure 2).

To assign a physical address to each site, the DNS Solver thread translates the site name in the corresponding numerical address (IP address). Xyro is now able to fetch the web page corresponding to this URL. The Connection Assignment thread associates each URL to a physical free socket of the Connection Array to begin a connection. The Fetcher thread sends the loaded web pages to the Local Policy Module that selects relevant links (URLs) in the page using some relevance criteria (see Section 2). Each relevant link is stored in the URLs FIFO and processed.

We give a top down description of some of the Xyro's data structure and threads

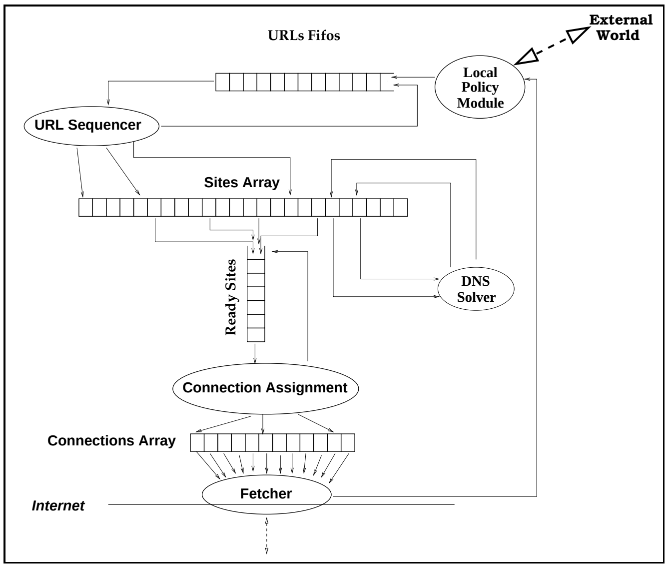

Figure 2: Functional Architecture on One Machine

presented in Figure 2.

URL Sequencer. It is a thread that loads URLs from the URL's FIFO, extracts the site name from the URL, and adds the URL to the site in the site array (Site Array in Figure 2) containing the extracted site name. The Sequencer provides a mapping between URLs and sites. An URL is associated to a site if the corresponding entry contains free place to contain it. The mappings between URLs and sites are implemented using a hash function.

Moreover to prevent memory overflow the system limits the number of URLs inside the whole system. If the amount of URL is full for this site or for the system we push it back in the URLs FIFO.

The acceptance of one URL in a site is also driven by the information contained in the robots.txt summarize.

When an URL is inside a site the URL Sequencer thread assigns a new state to this site. If the site has already its IP address this thread puts it inside the Ready sites FIFO, otherwise the site is registered to the DNS Solver.

Sites Array. Each site contains:

- the list of paths waiting to be fetched. The cardinality of this list is limited to prevent memory overflow;

- a summary of the robots.txt file. The robots.txt file described in [22] is a protocol that allows the Web master to specify which paths must not be followed by automatic crawlers (e.g. cgi scripts). In addition, each HTML pages may contain robot policy informations.
- the current state (e.g. up_to_be_fetched; waiting_for_a_connection; waiting_for_the_DNS_solving_process; waiting_for_the_robots.txt file, etc.);
- the IP number associated to the site name;
- the last date when this site has been fetched;
- the current request associated to the first URL on this site.

DNS Solver. This thread is in charge of associating an IP number to a host name [9]. The simplest way is to use the standard C function *"gethostbyname"* [4]. Nevertheless this is "synchronized" function: each time the system executes it no warranty is given for the function execution time. Moreover, during the *"gethostbyname"* execution no preemption is allowed. In a first version of Xyro, we used the *"gethostbyname"* function, but the system stalled several times.

We thus decided to use the asynchronous functions of the ADNS library [1] . Using this external library, the system submits several queries to the library functions, getting the answers when they are available, with no busy waiting.

To test the system we implemented our own DNS client to obtain some benchmarks: the ADNS library functions was efficient as ours DNS client but more reliable.

Connection Assignment. When the system is able to establish a new connection, The Connection Assignment thread loads a site from the Ready Site queue to assign it to one entry of the connection array.

During this phase, a recurrent risk for the crawler is the "rapid fire" problem, that happens when the crawler fetches several pages on the same web server in a short time interval. To avoid a rapid fire, the Connection Assignment thread decides if the loaded site has to be fetched or not, using the system date and the last visited date stored in each site. If the site can be fetched, the thread sets the last visited time of this site, finds and establishes a free connection. If the site is rejected, it is inserted as last element of the Ready Sites queue.

Connection Array. This structure contains the essential information for a connection: (i) the state of a connection, (ii) the time when this connection has been established, (iii) the request to submit, and (iv) the socket associated to the connection. The different states of a connection are:

1. Connection is free.
2. Connection is establishing.
3. Connection is submitting query to the distant server.
4. Connection is waiting for an answer from the distant server.

During our experiments we set the connection array to 300 entries. Some tests have been performed with 400 entries without big gain in terms of performance. We certainly increase the amount of loaded pages but we drastically increased the number of "time out" sent by the distant web server.

Fetcher Thread. This thread is the interface between the crawler and the Internet. It takes one connection in a round-robin way from the Connection Array, and according to the connection state executes some operations. It manages the sockets associated to the connections. Each socket establishes a link between the system and the remote server. The task of this thread can be summarized in two ones: (i) submitting a request to a distant site and (ii) reading the answer from the distant site. To reduce I/O blocking the system uses a-synchronized I/O functions (for instance the standard C function *"select"*), thus each operation (i.e.submiting a query or reading an answer), is not atomic.

Once a connection is established, each time the Fetcher reads a page portion it sends it to the Local Policy Module that reconstructs the whole page contents. When it finishes to read a page it sends an acknowledge to the Local Policy Module regarding the current connection. If an error appears during the different phases described above, the thread closes the connection and sends an error message to the Local Policy Module. If it fetches a *robots.txt* file it submits the answer, summarized by an appropriate parser, directly to the corresponding site. Moreover a time stamp is assigned to each connection (typically 45 seconds): when it is expired the connection is closed.

The read-write operations depend of the adopted Internet protocol. Xyro is implemented using the **HTTP 1.0** [16] protocol, that is the basic and simplest protocol to query the Web. An HTTP 1.0 request consists in one query followed by an answer. The answer contains an header with informations about the downloaded page (e.g. an error code). From this field the system reads some query states, for instance if the query was correct or well-formed, or if the interesting page was moved etc. A MIME [13] filter can be added during the reading phase to throw unreadable format (e.g. postscript, GIF, jpeg ...).

Due to its importance, we describe the Local Policy Module in a separate section.

2 Local Policy Module

The Local Policy Module decides which links in a page have to be loaded, based on the page content. There are several policies to decide if a link has to be loaded or not. We sketch some of the used policies to chose the relevant links, and we present our choice.

Discovery Policy. The system fetches a page, and the policy module parses it to find the relevant links, using some filters plugged on the URL contents to avoid some useless suffixes as ".jpg, .gif, .gz, ...". This model needs an auxiliary data structure to prevent possible cycles introduced by URL links. For instance, a hash table can be used to store the visited URLs. Each URL is represented using one bit, which is inverted the first time that the URL is visited. Collision on this hash table are acceptable in term of URL missed. The hash table is reinitialized when it is up to x percentage or every n hours.

Another mechanism anticipates the creation of infinite pages limiting the size of one page in main memory.

Discovery Policy using pattern/matching. URLs are accepted only if they match with a specific prefixes or suffixes such as "*.inria.fr/*, *.edu/*, *.index.html, …". This policy is recommended to create a database focused for instance on a set of site or country.

Discovery Policy keyword focus. This policy parses every pages, to find a set of given keyword inside. Weighting could be added with the position of the word inside the document. A basic algorithm decides to follow the page whether its score reaches a limit.

Refresh Policy. This policy is based on some attributes of the page: the importance of the page, last date read, etc. The Policy module using this information stores some meta-data with the page fetched, to decide when to refresh it. Modern Search Engines as [12, 21] uses this policy [5, 7]. Xyleme system uses also this technique as described in [19].

The different policies given are not exhaustive. We may apply some data mining techniques on the page or and the URLs to decide which pages have to read or refresh as given in [20, 14, 6].

Xyro Policy Module. The Xyro crawler was designed for the Xyleme system to find all the XML documents over the Web. The Xyro Policy Module uses a policy mixed between the "Refresh Policy" and the simple "Discovery Policy". The page loaded by crawler machines are parsed and the interesting links are submitted to the Global Policy Module and will distribute them on the appropriate Local Policy Module. Xyleme Global Policy Module (Refresh module) submits URLs to refresh with a higher priority, via CORBA [8] mechanism, as the crawler does (by their Local Policy Module).

3 Experiments

The tests have been performed on a PC with an Intel Pentium II at 450MHz, with 128 MB of main memory, under Linux ReadHat 5.2 distribution. The PC was connected to a local network by a 10 Mb/s Ethernet connection.

We had an available Internet bandwidth of one third of the 20Mb/s total bandwidth available at the INRIA-Rocquencourt network.

The duration of our tests varied between 24 hours and a maximum of ten days. During longest test, Xyro ran for ten days (Actually one of the first test in May 2000) fetching about 25 millions of pages, with an average of 29 pages per second. We could not launch our test indefinitely for two reasons: first of all we could not reserve for long period the INRIA Internet bandwidth, and (ii) we discovered a bug in the operating system that often forced us to stop our test to recuperate the test results. This bug was caused by the *"shutdown"* C function of the Linux kernel. We thus tested a new Xyro version in September 2000, adapted to turn round this system bug. Running Xyro for 24 hours, the performances were notably increased w.r.t the last version.

The current version of the Xyleme crawler is now bug free; actually with Linux system it fetches till 4 millions of pages (46 pages/s) where using Solaris 2.6 it is able to fetch 5 millions of pages (57 pages/s). This difference can be explained by a different multi-thread management present in the two operating systems. When a thread enters on the kernel the Linux software is designed to block the whole process, where under Solaris the thread management is done in hardware.

To have an idea of the order of magnitude of fetched pages by a robot, we can consider that the Mercator [15] robot, working on two 533 MHz Alpha processor with 2 GB of RAM, is able to perform an average of download rate of 112 documents/sec.

Our latest tests were done in March 2001. We launched the scalable version of Xyro using Linux machines during one week. It worked on four machines and fetched an average of 4 millions of pages per day on each computer with an average of 184 pages/s. This performances were possible thanks to the scalability of our system showed in Section 1. Other results are summarized in Table 1.

	After 6 hours	After 8 hours
Pages fetched	1 038 498	1 328 609
Total sized Fetched	88.3 GB	110.4 GB
Errors	197 964	258 840
Number of DNS success	173 261	200 228
Number of unsuccessful DNS	16 056	19 921
Total of sites seen	173 261	210 990
sites with robots.txt	40 028	47 829
of which good robots.txt	36 040	43 025
Memory used (in MB)	91.372	91.392

Table 1: Xyro's statistics

4 Improvements

In this Section we consider possible improvements to the easting system in order to increase global performances.

We present in this section two kinds of the possible add-on : (1) a better integration with the DNS Solver and (2) supporting two others Internet protocols.

DNS Information. The DNS Solver provides not only the IP address for a site but also some useful information attached to this resolution. A short description of the possible informations are given below :

- **Canonical Name.** This field represents the true name of the site. This information can be used to detect basic sites alias and to cluster sites with different names inside the same site data structure as described in Section 1. To use this information we have to change the internal architecture design where the URL Sequencer thread calls before the DNS Solver and next dispatches the URL following its IP address and not the site name.
- **Expiration Time.** This field refers the time interval for which this IP address is valid. If we do not refresh the IP address, some sites could kept indefinitely the same IP number and violates in this way the DNS protocol.
- **List of equivalent IP address.** This list can be used to collect different queries on the same site but related to different connections. These informations can be

used when crawling a specific site. Another way to use this information is to kept the closer or faster address using a *"ping"* evaluation or geographic information.

- **Named daemon.** A limit of the current UNIX DNS solver, called *"named"* provided by the BIND library, is the impossibility to submit several request at the same time. The DNS RFC allows the possibility of serving several requests inside a same query, but current implementation manages only one request by query. This upgrade could increase resolution time avoiding connection establishment for every resolution. We noticed also that the current implementation of the *"named"* daemon is not limited in term of memory size[1]. So we broke one time the INRIA DNS Server, before that they changed the DNS machine configuration.

Internet Protocols. The Internet world is not limited to the HTTP 1.0 protocol. New Web Server implements the next generation of Web protocol called HTTP 1.1. Moreover a very old Internet protocol, denoted by FTP, permits to fetch document on a site. In this paragraph we describe the consequence in your architecture to implement these protocols.

- **HTTP 1.1 [17].** This is an extension of the previous protocol, and thus more sophisticated. It allows the system to submit several queries keeping the same connection instead of restoring it for each query. This protocol has been designed to satisfy the traffic generates by the new page where downloading a page implies very often downloading all the page linked by it (e.g images). Actually, instead of downloading a page every 45 seconds on one site [2], fetching 5 pages with the same connection allows to reconsider this site after for 5 * 45 seconds. Since get a connection is very time consuming, the use of this protocol allows to increase the Fetcher thread performances.

- FTP [10]. On the contrary to the HTTP protocols, that is query/answer protocols, this one is a dialog protocol like SMTP [23] protocol. Page fetching is decomposed in several steps that can be described as follows:

[1]This problem seems to be resolved by newer version of the BIND [4] library
[2]avoiding rapid fire

1. Establish the connection.
2. Submit a login.
3. Submit a password.
4. Navigate on the File System.
5. Submit the query.

Each step could provide several errors. This protocol permits, certainly, to have a bigger vision of the Internet in term of usefulness document accessible. Moreover, some Web master could give you a login and a password to download these documents. So, for each site the system must keep in memory a list of login/password depending of the document path.

5 Conclusion

To conclude this implementation was a success in term of performance. It was a lot of fun and many nights of programming.

Future Works We will continue to provide the first version of Xyro, rename Larbin, under a free license and available on [3]. External users are invited to test it and could ask for someq specials improvements. We plan to use Xyro to follow our experimentations on the Web. One direct consequence of this work is to redefine the notion of site. Currently, a site is defined by its site name (www.inria.fr) but we believe that we will have a better definition of what is a site with a multidimensional factor who could be semantic contents, web graph connectivity, DNS name, ... We plan to upgrade the current version of Xyro with this work. The notion of site will be taken during the parallelization process. The goal is to manage the same site with the same Local Policy Module.

Acknowledgement We want to thank the different responsibles of this work: Serge Abiteboul, Bernd Amann Guy Ferran and Mihai Preda. Special thanks for the INRIA Rocquencourt Web master who supports us when some complaints arrived,

AFS team when we broke their DNS resolvers and Network team who supports our stress network. Finally we thank the whole Xyleme research project and Verso team.

References

[1] GNU ADNS library. http://www.chiark.greenend.org.uk/ ian/adns/.

[2] Vincent Aguiléra, Sophie Cluet, Pierangelo Veltri, Dan Vodislav, and Fanny Wattez. Querying xml documents in xyleme. In *the proceedings of the ACM-SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

[3] Sébastien Ailleret. Larbin. http://www.ailleret.com/.

[4] Berkeley Internet Name Domain. http://www.isc.org/bind.html.

[5] Sergay Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. What can you do with a Web in your Pocket? *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1998.

[6] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[7] Junghoo Cho and Hector Garcia-Molina. Estimating Frequency of Change. Technical report, Stanford University, 2000. http://dbpubs.stanford.edu:8090/pub/1999-22/.

[8] Common Object Request Broker Architecture (O.M.G). `http://www.corba.org/`.

[9] Domain Names Systems – Implementation and Specification. RFC 1035.

[10] File Transfert Protocol. RFC 959.

[11] Altavista incorporation. http://www.altavista.com/.

[12] Google incorporation. http://www.google.com/.

[13] Network Working Group. MIME (Multipurpose Internet Mail Extensions) part one : Mechanism for specifying and describing the format of internet message bodies. RFC 1521.

[14] Taher H. Haveliwala, Aristides Gionis, and Piotr Indyk. Scalabale techniques for clustering the web. In *WebDB 2000*, 2000.

[15] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2, 1999.

[16] Hypertext Transfert Protocol – HTTP/1.0. RFC 1945.

[17] Hypertext Transfert Protocol – HTTP/1.1. RFC 2068.

[18] Internet grow and statistique: Credits and background. http://www.mit.edu/people/mkgray/net/background.html.

[19] Laurent Mignet, Mihai Preda, Serge Abiteboul, Bernd Amman, and Amélie Marian. Acquisition and Maitenance of XML Data from the Web. Technical report, INRIA - VERSO, 2001.

[20] Bamshad Mobasher, Robert Coole, and Jaideep Srivastava. Automatic personalization based on web usage mining. Technical report, DePaul University and University of Minnesota, 2000.

[21] http://www.northernlight.com/.

[22] The robots exclusion protocol. http://info.webcrawler.com/mak/project/robots/exclusion.html.

[23] Simple Mail Transfer Protocol. RFC 1869.

[24] The web robots page. http://info.webcrawler.com/mak/projects/robots/robots.html.

[25] Xyleme S.A. http://www.xyleme.com/.

[26] Xyleme research project. http://www-rocq.inria.fr/verso/Xyleme/.

[27] Yahoo incorporation. http://www.yahoo.com/.