

CoBWeb – A Crawler for the Brazilian Web

Altigran S. da Silva* Eveline A. Veloso Paulo B. Golgher

Berthier Ribeiro-Neto Alberto H. F. Laender Nivio Ziviani

Department of Computer Science
Federal University of Minas Gerais
31270-970 Belo Horizonte MG Brazil
{alti,eveline,golgher,berthier,laender,nivio}@dcc.ufmg.br

Abstract

One of the key components of current Web search engines is the document collector. This paper describes CoBWeb, an automatic document collector, whose architecture is distributed and highly scalable. CoBWeb aims at collecting large amounts of documents per time period, while observing operational and ethical limits in the crawling process. CoBWeb is part of the SIAM (Information Systems in Mobile Computing Environments) search engine which is being implemented to support the Brazilian Web. Thus, several results related to the Brazilian Web are presented.

1 Introduction

One of the fundamental components of Web search engines is the *automatic document collector*. The main task performed by this component is the automatic gathering of Web documents which are usually stored locally for further processing. If the application is a search engine, these local document copies can be used to produce index terms, which are required for speeding up the processing task at query time.

In the jargon of Internet application developers (see, for example, [1]), automatic Web document collectors fall in the category of *Web robots* which are applications that systematically traverse the Internet to perform some specific task. More specifically, automatic Web document collectors are commonly referred to as *Web crawlers* and can also be used in tasks such as Web survey, site checking and maintenance, site mirroring, and resource discovery [8].

There are many interesting problems related to Web crawling. Among them, we mention: the scheduling of URLs to be visited [4]; the observation of good behavior agreements, such as the *Robot Exclusion Standard* [7]; the coordination of actions when many collecting agents are used; the correct treatment of exception conditions and errors of many kinds; and the assurance of a good coverage of the set of collected documents with respect to the set of all existing documents.

Even though many applications need efficient and robust Web crawling mechanisms, being the search engines the best example of such applications, this subject has not been extensively discussed in the literature. One of the strongest reasons for this lack of discussion is the strategic interest in Web crawling technology by the main commercial search sites in the Web.

This paper describes *CoBWeb*, a Web crawler developed to serve as the document collecting mechanism of the SIAM search engine currently under development within the *SIAM (Information Systems on Mobile Computing Environments)* project [13]. The main features of CoBWeb are a distributed and scalable architecture, the observation of operational and ethical limits established for the resources it uses, and a genuine mechanism for updating documents already collected. In addition, since CoBWeb currently collects only documents from the “.br” sub-domain, we present some quantitative figures about the Brazilian Web, its documents, and its HTTP servers.

The paper is organized as follows. In Section 2, we discuss the main problems related to crawling and approaches for dealing with them. In Section 3, we describe key operational features of CoBWeb. Section 4 presents some experimental results on the performance of CoBWeb. Information about the Brazilian Web, obtained from the document col-

* On leave from University of Amazonas, Brazil.

lecting process, is presented in Section 5. Finally, Section 6 presents some conclusions and discusses future works.

2 Web Crawling

Given an initial specification of which Web documents to collect, an ideal process of Web crawling would collect all documents satisfying this specification in a suitable period of time. Since it is usually not possible to collect all documents, this requirement is always relaxed to consider the retrieval of a subset of all documents which provides a good document coverage. Further, this must be accomplished while sustaining a *collection rate* (documents/time unit) suitable for the application to which the process will serve. Additionally, the collecting mechanism should observe operational and ethical limits established (or recommended) for the resources it uses (such as the number of visits to a site in the unit of time).

In the case of the SIAM search engine, only indexable documents (e.g., HTML pages, Postscript and PDF files, etc.) stored in HTTP servers within the “.br” top-level domain are collected. This set of documents roughly corresponds to what can be called the “Brazilian Web”. The crawling mechanism is engineered to avoid the overload of the HTTP servers from which it requests documents, as well as of the network infrastructure it traverses.

2.1 Overview

The process of Web crawling can be described in general by the algorithm 1. In this algorithm, the operations

Crawling

```

1 begin
2   Let  $I$  be a list of initial URLs;
3   Let  $F$  be a queue;
4   foreach URL  $i$  in  $I$ 
5      $Enqueue(i, F)$ ;
6   end
7   while  $\neg Empty(F)$ 
8      $u \leftarrow Dequeue(F)$ ;
9      $d \leftarrow Get(u)$ ; /* request document  $d$  pointed by  $u$  */;
10    Store  $d$ ;
11    Extract the hyperlinks from  $d$ ;
12    Let  $U$  the set of URLs cited in these hyperlinks;
13    foreach URL  $u$  in  $U$ 
14       $Enqueue(u, F)$ ;
15    end
16  end
17 end

```

Algorithm 1: Description of a Web Crawling Process.

Enqueue, *Empty*, and *Dequeue* are usual operations over a

FIFO queue F , with the following modifications. Operation *Enqueue* only adds a new URL to the queue if it is not already there. Operation *Dequeue* only marks the URL in front of the queue as “removed”, instead of actually removing it. As a consequence, *Empty* is true only when all URLs in the queue are marked as “removed”. This policy for URL’s queue maintenance will be called *LWF (Longest Wait First)*.

2.2 Distribution of the Crawling Process

For any reasonable implementation of the Algorithm 1, the operation *Get* in line 9 is, in general, the one presenting the greatest cost. The main reason is that it includes the network latency associated with a request which is sent to a remote HTTP server. Longer is the network latency, smaller is the document collecting rate. To reduce the impact of this effect, we propose to distribute the requesting task among many document *collectors*. These document collectors are implemented as distinct processes, which possibly execute in different computers.

This distribution of the collecting task brings, on the other hand, the need for coordination between the collectors. This is needed, for instance, to avoid that two collectors retrieve a same document twice. Thus, if the Algorithm 1 is to be executed by several document collectors, is necessary to consider that the queue F is now accessed concurrently by all the document collectors. The operations of queue maintenance must be such that the cost of their execution and the cost associated with concurrency control are much smaller than the cost of a *Get* operation.

To further reduce the negative effects of the network latency in the document collecting rate, the collectors can be assigned to nearby geographical regions, in such a way that a document collector accesses only HTTP server which are not far away. In this case, to determine how “near” a collector is from a server, metrics such as mean latency and cost could be used. In this approach, adopted in [3] and [12], the direct sharing of a queue F cannot be easily accomplished, because the sets of collectors are typically placed in nodes of a wide area network.

In [3], the manipulation of the queue is assigned to a single “central” node. The data exchange between this node and the other nodes is performed by means of a protocol that includes many techniques such as compression and the use of signatures. This leads to a latency which is much smaller than the typical HTTP latency.

The solution adopted in [12] was simply to partition the set of HTTP servers to be visited and statically allocate each set of collectors to a set of HTTP servers. The partition was based on estimation of network latency. Note that, in this case, it is necessary to know the set of HTTP servers in advance.

2.3 Exception Situations

In practical situations, it is not always possible to successfully execute the *Get* operation. Many exception situations can happen, such as no answers or timeouts from HTTP servers, wrong or no longer existing URLs, etc. In situations like these, depending on the kind of exception, the URL can be returned to the queue, maybe with some penalty (for instance, if a timeout has happened), or it can be marked as “removed” (for instance, in the case the URL has not been found). Thus, Algorithm 1 should be modified to deal with exception situations. These modifications are, however, omitted from our discussion for the sake of brevity.

Besides, there can be URLs that refer to objects that are not of interest to the application being served. For instance, text based search engines have no interest in GIF images and executable programs. The Web crawler can treat these cases by simply not requesting these objects and, if needed, generate a log of these occurrences with proper statistics.

2.4 Operational Limits and Ethics

As we have already said, Web crawlers must observe the operational and ethical limits established or recommended for the use of shared resources. The most important ethical restriction to be observed for crawling the Web is the “Standard for Robot Exclusion” [7]. This standard is not defined or enforced by any official organism. Instead, it is the result of an agreement between developers of Web robots for commercial search engines.

The standard for robot exclusion basically establishes the following rules:

1. Any automatic Web browser, prior to any other request, should ask the HTTP server it visits a file called “robots.txt”. This file, if it indeed exists, will define access constraints to the server’s contents. These access constraints must be observed. The format and syntax used in the file are described in [7];
2. The contents of any HTML page received as the result of a HTTP request must be scanned for special *meta-tags* that indicated whether or not the page can be collected and whether or not it is permitted to extract and to follow the *hyperlinks* contained in it. The description of the format and the codification of these *meta-tags* are described in [7].

Both rules aim at restricting access to sensitive or very volatile information in Web sites. The main distinction between them is that rule 1 is enforced by the site administrator for the entire site, while rule 2 provides common users with the power to protect their own pages, independently of the site’s global policy.

Besides this, it is recommended that crawler developers avoid the so-called *rapid fire* [8], which consists of the sending a series of HTTP requests with little time interval between them. In [8], it is recommended that the interval between requests to a same HTTP server should be of at least 60 seconds. We state this recommendation as the following rule:

3. A request to a HTTP server must only be sent if the server is *free*. A server is considered to be free if at least 60 seconds have passed since the answer for the last query (submitted to this server) has been received.

The enforcement of rule 3 has very strong consequences for the collecting process. First, operation *Dequeue* (in Algorithm 1) must be modified in such a way that a given URL will be selected only if the corresponding HTTP server is free (as stated by rule 3). Moreover, this rule imposes a lower bound in the time interval between requests to a given HTTP server. This lower bound can, in practice, compromise the collection rate when many URLs refer to a same HTTP server.

Although only rules 1 and 2 are actually included in the Standard for Robot Exclusion as described in [7], whenever we mention the standard we also refer to rule 3.

2.5 URL Scheduling

The use of FIFO based policies on URL scheduling, i.e., to determine the next URL whose document will be collected, is not always the best option.

Suppose that a list of URLs could be ranked in terms of their importance to a given application. According to this ranking, a URL, say *a*, is more important than another URL, say *b*, for the application. For such situations, [4] proposes a reordering of the URL queue such that the most important URLs would be visited before than the less important ones, when there are constraints on the time to collect the documents and on the space available for storing local copies of them.

Another situation when the URL scheduling cannot be based entirely on FIFO-like policies is when rule 3 of the Standard for Robot Exclusion must be observed. In this case, the LWF policy should be modified in order to select only URLs referring to free HTTP servers. This modified policy is called *KLWF (Kind LWF)*.

2.6 Updating the Local Copies of Documents

Another important topic related to Web crawling is the freshness of the documents locally stored with regard to the current version of the document in the original server.

The rate with which Web resources change was studied in [6], where analysis of access traces from HTTP servers

of real corporations suggested that the most accessed Web resources are the ones that change more frequently (on a weekly or even on a daily basis). This trend is more accentuated if the documents are HTML pages.

A trivial solution to this problem is to re-collect all pages. However, this might imply in recollecting tens of gigabytes of data. According to data available in [1], the main search engines claim to have a minimum freshness of 3 to 4 weeks for the locally stored documents. Yet, also according to [1], these Web search engines collect between 3 to 10 million pages a day. We conclude that this “brute force” solution (recollect all documents) is currently the most used approach for updating Web documents. Note that, since the Web is growing exponentially, this approach will not stand for too long. Thus, new efficient and scalable approaches are needed to treat adequately this problem.

A simple way of lessening this problem would be to request from the HTTP servers only documents which changed since a given time instant specified. However, this would only avoid the document recollection, remaining unchanged the traversal prescribed by Algorithm 1. Moreover, not all available HTTP servers implement “if-modified-since” HTTP headers, feature needed to perform this kind of request.

A possible non-exhaustive approach would be to require that document producers (or their brokers) themselves communicate changes in the documents they made available. This solution was proposed in the *Harvest* system [3], but requires every document producer to have a specially designed component to communicate changes. This drawback restricts the applicability of this approach to the development of search sites about specific subjects whose contents are generated by well known document producers.

An adoption of this idea to the current Web scenario is to obtain from cache proxy servers [9] *near* to the crawler (e.g., in the same LAN) information about URL requests generated by the users. With this information, the crawler can inspect its local list of URLs and, in case a new document or a new version of a known document has been requested, retrieve the document of interest directly from the cache proxy server. In this case, the freshness of the locally stored document copies would depend on the cache proxy server’s refresh policy. This idea has two potential advantages: (1) it uses the available Web caching technology (mirrors, cache hierarchies, etc.) with low implementation costs (see Section 3 for an example) and (2) through its use, documents with high access demand will have fresher local copies, what is in accordance with the results in [6]. We call this approach *cache parasite*.

A combination of the brute force approach with the cache parasite approach, for the context of Web cache servers was proposed in [5]. In this case, a proxy cache server is responsible for tracking changes in the resources

of a set of Web servers and for propagating these changes to other proxy cache servers it cooperates with.

2.7 Crawling Coverage

One of the most challenging problems in Web crawling is to guarantee a good crawling coverage, i.e., to guarantee that the set D' of documents collected by a collecting process presents a good intersection with the set D of documents that should be collected according to the initial specification.

In first place, depending on the specification, it may not be straightforward to identify the set D . Actually, there is some information related to the set D that could be used. For instance, the set of first, second, and third level DNS domains, the set of Internet hosts responding to an IP checking request, and the set of Web servers found on a Internet traversal could be used to somehow estimate what is D . However, it is not clear how this information is related to the set of pages actually residing in the Web servers.

Moreover, there are cases where more than one URL refers to the same document. For instance, the URLs “http://fua.br/” and “http://fua.br/index.html” lead exactly to the same document and can both appear among the hyperlinks of previously collected pages. In this case, the same document would be collected twice, compromising eventual statistics. Note that while in this case the redundancy can be easily detected, not all cases are straightforward as this.

A generic solution to this problem would be to generate content based signatures to each document (using, for instance, the MD5 algorithm [11]), and for each newly collected document to verify its signature against the signature of the previously collected documents. This verification, if done during the collecting process, would certainly compromise the collection rate. Alternatively, the generation and comparison of documents signature could be done statically for the local stored document copies.

Besides the difficulties discussed so far for estimating the collection coverage, the greatest problem faced here seems to be the occurrence of large URL hierarchies whose “root” URL is not referred to in any document collected. If such hierarchies are common in most Web servers, then this problem, and its cumulative effects, can be disastrous for the collecting process. Obviously, its is not possible to determine, in general, whether this is a common situation or not. One can argue, however, that there is little interest in a document referred to by no other document. For a theoretical point of view on the problem of Web crawling coverage we refer the reader to [10].

In Section 3, we describe how the problems discussed in this section are addressed in the implementation of CoB-Web, considering the application for which it was built (that is, to feed a search engine for the Brazilian Web) and the op-

erational environment where it will run.

3 Main Features of CoBWeb

CoBWeb is a distributed Web crawler whose operation is mainly oriented for efficiency, robustness, and parsimony in shared resources usage. It was developed to serve as the document collecting mechanism of the search engine currently under development as part of the SIAM project [13]. As already mentioned, CoBWeb collects only indexable Web documents (HTML pages, postscript files, etc.) found in HTTP servers whose DNS domains are sub-domains of the “.br” domain. We consider the Brazilian Web as being composed by these set of servers, although there are servers outside this scope that could be also included because they host pages somehow related to Brazil. We consider that all documents that satisfy this specification are of equal importance, given that the search engine will not be tied to any specific subject. Currently, we are not requesting query URLs and password protected documents are handled via a standard HTTP error code.

In a typical collecting session, several collecting processes are run simultaneously, each of them executing an instantiation of Algorithm 1. The operations over the URL queue are isolated by a central *scheduler* which coordinates the operation of the various collecting processes, or simply *collectors*. The general architecture of CoBWeb is illustrated in Figure 1. Notice that the collectors might be distributed over several machines to speed up the processing task.

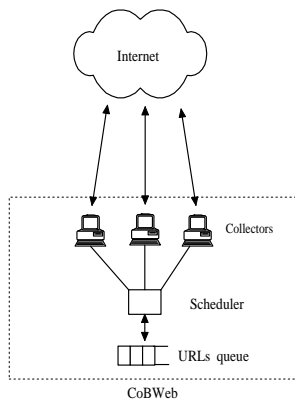


Figure 1. CoBWeb's architecture.

To each one of the collectors, is assigned the task of collecting a specific Web document. Once this document is collected, a local copy of it is created. Additionally, the URLs in the hyperlinks within this document (if any) are extracted and sent to the scheduler to be enqueued. The col-

lector then requests a new URL from the central scheduler and proceeds to collect the corresponding document.

For the manipulation of the queue, the scheduler implements the KLWF policy, i.e., CoBWeb globally follows the Standard for Robot Exclusion, considering all active collectors. Note that, as all collected documents have the same importance, there is no need to reorder the queue as in [4].

CoBWeb also addresses several exception situations, ranging from time-outs in HTTP requests (in this case the URL “returns” to the queue) to URLs that refer to no documents (in this case the URL is “removed” from the queue and the collection does not occur). Every action related to the collecting session, whether successful or not, is logged for further analysis.

During a collecting session, CoBWeb accumulates several pieces of information about collected documents, URLs traversed, and HTTP servers visited. This information includes: number of collected documents, amount of collected bytes, number of visited servers, number of extracted hyperlinks (regardless whether the corresponding documents were collected or not), exception accounting, references to non indexable objects (figures, executables, e-mails, etc.) and the number of references to each URL. Excerpts from this information, gathered through a collecting process, are presented in Section 5.

To treat the problem of updating local document copies, CoBWeb uses the cache parasite approach described in Section 2. The current implementation is specific to the *Squid* [2] Web cache proxy server. It can be easily used with servers already in operation. Figure 2 presents the CoBWeb architecture modified to update document via cache parasite. The updating process works as follows. A *URL redi-*

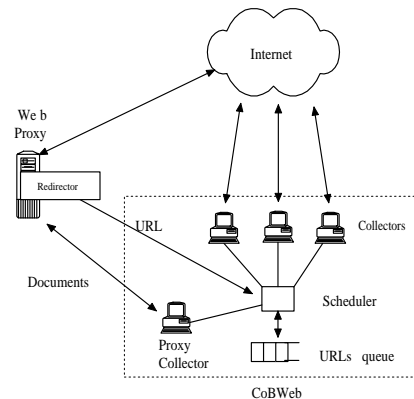


Figure 2. CoBWeb's architecture with cache parasite.

lector is installed in the cache proxy Web server. This URL redirector is a small script (easily incorporated in Squid's

installation) that intercepts every user request to the cache proxy server and sends the requested URL to CoBWeb. The URL is checked to determine whether or not it should be collected. Then, a special collector, the *proxy collector*, requests the URL to the cache proxy server. We recall that this server is running “near” to the collector (e.g. in the same LAN). To be requested from the cache proxy server, a URL must be such that: (1) it satisfies the initial specification of what is to be collected; (2) the document it refers has not been collected yet or it was collected before a pre-defined time limit (currently 24 hours). Note that there must be a delay between the announcement of the URL’s arrival and a request for its associated document, to allow the proxy server to request the URL from the original server, if needed.

4 Performance Results

The exponential growth of the Web is a very well known fact. Therefore, high collection rates and high scalability are essential properties of a Web crawler, so it can cope with the growing amounts of data available. In this section, we present some experimental results that show that the CoBWeb architecture has such qualities.

To assure a higher scalability, CoBWeb is based on a distributed architecture. As we have already discussed, two facts motivate the adoption of such an architecture: (1) the observation of the Standard for Robot Exclusion, that imposes a lower bound in the collecting rate; and (2) the assumption that the HTTP request latency dominates the total crawling cost.

An experiment was carried out to evaluate the efficiency of CoBWeb’s distributed architecture and to verify the assumption about the HTTP request latency. Figure 3 shows the collection rates for 5 different experiments that use 5, 10, 15, 20 and 25 simultaneous collectors. The same set of initial URLs were used in each experiment.

As Figure 3 shows, the collection rates increases linearly with the the number of simultaneous collectors. This occurs due to the dominance of the HTTP request latency over the remaining operations, which results in a high scalability of the scheduler as the number of collectors grows. As it should be clear, the linear behavior observed will flatten out in case the network bandwidth saturates. Table 1 compares the HTTP request latency with the scheduler latency (relative to requests for new URLs).

The results above were generated in a time period during which our local network traffic as well as the Brazilian Internet backbone traffic was small (according to statistics from RNP, Brazil’s largest Internet backbone operator). Further, the set of initial URLs was chosen so that all HTTP servers visited were fully operational during the execution of the experiment. These precautions were taken to minimize possible variations of the HTTP request latency, so

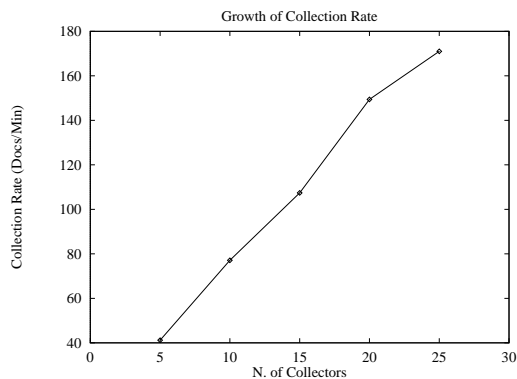


Figure 3. Linear growth of collection rates with the number of simultaneous collectors.

Collectors	Avg. HTTP Latency (secs.)	Avg. Scheduler Latency (secs.)
5	3.42	0.01
10	3.40	0.02
15	3.36	0.02
20	3.39	0.02
25	3.42	0.02
Average	3.39	0.02

Table 1. Averages of HTTP and Scheduler Latencies During the Experiments.

that we could isolate the impact of the number of simultaneous collectors in the collection rate.

The operational conditions in which the collection rates shown in Figure 3 were measured, as described previously, are unreal in practice, given the environment we actually use for the day-to-day operation of CoBWeb. Therefore, to verify the effective collection rates of a typical CoBWeb document collecting process, the mean rates of 5 consecutive days of a real collecting process are shown in Table 2.

Currently, collecting processes are performed using two Intel Pentium II 400 Mhz Xeon machines, with a 256 MB RAM memory each. One of these machines executes the scheduler and 30 collectors and the other executes another 30 collectors. This distribution of collectors was determined empirically through distribution tests previously performed.

Many factors contribute to the worst performance obtained in the real collecting process, compared to the results shown in Figure 3. Besides the smaller number of available machines (only 2), what prevents a better distribution scheme, the collecting rates were measured all day long in normal weekdays, when resources such as the local network, HTTP servers, and the Internet backbone were in full service.

Date	Collection Rates	
	Docs/Min	Docs/Day
25/03	63	72274
26/03	107	99629
27/03	70	89969
28/03	62	52090
29/03	64	71820
Mean	73.2	77169

Table 2. Collection rates in 5 consecutive days in a real collection process.

As discussed in Section 2, another essential property of a Web crawler is its ability to ensure the freshness of the locally stored documents. An experiment was carried out to verify the efficiency of the cache parasite approach used in CoBWeb. Therefore, about nine thousand HTTP requests for HTML pages were submitted to the cache proxy server being used by CoBWeb.

This set of requests was obtained from daily access traces from one of the busiest Brazilian cache proxy servers, namely the one installed at the POP-MG Internet connection provider, which satisfies our criteria of proximity, since it is located at the same LAN as the machine running the scheduler. This set corresponds to about 2% of the total of accesses carried out in the “.br” sub-domain. Thus, we can say that this set of requests fairly represents a typical run of requests to a popular cache proxy server.

This experiment had two main goals: (1) to verify if cache parasite is an useful approach, considering the number of potential updates; and (2) to confirm that the latency of request to a “near” cache proxy server is really lower comparing to those obtained with direct accesses to the HTTP server. Table 3 presents the measured results.

Updates performed	7880 (89%)
Updates discarded	503 (5.7%)
New Documents collected	487 (5.3%)
Total of requests	8870 (100%)

Table 3. Results obtained from a set of requests to the cache proxy server.

In Table 3, the term “Updates discarded” refers to potential updates that were not actually carried out, because the current document copy was collected within a specified time interval (24 hours in our case). It was also verified that 63% of the proxy server requests were answered in less than 1 second. The remaining requests caused exception conditions or were collected directly from the HTTP server, thus raising the latency time.

It is important to note that CoBWeb cache parasite, al-

though fully implemented, is not yet operational in a commercial proxy server. Thus the set of locally stored documents were never updated, fact that favors the update rates obtained in this experiment. However, the results show that this approach is quite effective in maintaining the freshness of stored document copies.

5 Results of the Collecting Process

A very interesting side effect of crawling in the “.br” sub-domain is that it allows for the analysis of several features related to the Brazilian Web. In this section, we describe several data that illustrate the current status of the Internet in Brazil. This data was gathered during several collecting sessions performed by CoBWeb.

Table 4 presents characteristics related to the size of the Brazilian Web. These numbers correspond to a 15 day period of document collecting, which was not sufficient to collect all available documents. However, the total number of collected URLs and URLs present in the queue (which already began to shrink) is a reasonable indication of what is size of the Brazilian Web.

URL	Quantity	Bytes	
		Total	Mean
HTML collected	1,115,288	6,4 G	6.0 K
Other collected	15,707	1,9 G	129.2 K
Total collected	1,130,995	8,3 G	-
URLs to be collected	422,512	-	-

Table 4. Numbers Related to the Brazilian Web Size.

In Table 4, the first line refers to the HTML documents found and collected. These documents were classified according to the corresponding server response. The second line presents data related to other types of documents, which can be indexed, e.g., Postscript and Adobe Acrobat files. Finally, the number of URLs found but not yet collected is shown in the third line of the table.

The collection process also allowed the evaluation of the number of HTTP servers visited during the process. Table 5 shows the total number of server visited and the mean number of collected documents per server.

Visited Servers	35,508
Documents per Server (mean)	32

Table 5. Numbers on HTTP Serves Visited.

Finally, Table 6 summarizes the most frequent errors found during the collection process. The numbers in first column refer to HTTP error codes.

Error	Frequency
400 - Bad request	4,013
403 - Forbidden Access	4,338
404 - Document not found	99,957
Server not found	52,497
Other	802
Total	161,607

Table 6. Most frequent errors found during the crawling process.

6 Conclusions and Future Work

In this paper we presented CoBWeb, a Web crawler whose primary goal is to collect documents from the Brazilian Web to feed the search engine of the SIAM project. CoBWeb has as main features: efficiency, robustness, and kindness in the usage of shared resources. We have described the architecture and the operation of the CoBWeb crawler. We have also discussed the main problems related to Web crawling and our solution to them.

To evaluate CoBWeb's architecture, we presented performance measures taken from experiments and from a real collecting process. These results indicate that by running several collecting processes and by properly distributing (similarly as done in [12]) it is possible to obtain collection rates comparable to that of the crawlers of commercial search engines. We also presented some statistical data gathered from a real collecting process, which reveal some interesting information about the Brazilian Web. Moreover, the document collection which is been gathered by CoBWeb will be made available for further studies.

CoBWeb, as far as we know, is the only large Web crawler target at the Brazilian Web which is currently in operation. Further, a main contribution of our study is an evaluation of the problems related to crawling in the Brazilian Web. Moreover, as an specific contribution, we highlight the good results obtained by the use of the parasite cache approach to update the set of collected documents.

Despite the efficiency of the current implementation of CoBWeb, we have identified some points where it can be further improved. In particular, we are implementing a new scheduling strategy that will allow reducing the cost of queuing operations, but that will keep the enforcement of the Standard for Robot Exclusion. This new design will also allow many schedulers (with their respective set of collectors) to be placed in nodes of WAN and to cooperate to perform an integrated collecting process, improving its efficiency and reliability.

Although search sites form a very important class of services in the current state of the Web, there is few technical literature available on the subject of developing them. This is particularly true in the case of Web crawlers. We

expect that the development of CoBWeb, as well as the development of the search engine to which it will serve, can stimulate more discussion and studies on this subject.

Acknowledgment

The authors would like to thank to Ricardo Costa Rodrigues, Wagner Meira Júnior and Rodrigo Lopes Cançado Fonseca, all of them from DCC/UFMG, for the discussions on the use of cache proxy serves. This work is partially funded by the SIAM (MCT/FINEP/PRONEX grant 76.97.1016.00) and the CYTED (VII.13 AMYRI) projects.

References

- [1] Search engine watch. <http://www.searchenginewatch.com>.
- [2] Squid home page. <http://squid.nlanr.net/Squid>.
- [3] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, 1994.
- [4] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. In *Proceedings of the 7th International WWW Conference*, Brisbane, Australia, 1998.
- [5] F. Dougliis. Resource updates on the web. <http://www.research.att.com/~dougliis/characterization-pos.html>.
- [6] F. Dougliis, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: a live study of the world wide web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.
- [7] C. P. Kollar, J. R. R. Leavitt, and M. Mauldin. Robot exclusion standard revisited. <http://www.kollar.com/robots.html>.
- [8] M. Koster. Guidelines for robot writers. <http://info.webcrawler.com/mak/projects/robots/-guidelines.html>.
- [9] D. Menascé and V. F. Almeida. *Capacity Planning for Web Performance - Metrics, Models & Methods*. Prentice Hall, 1998.
- [10] A. O. Mendelzon and T. Milo. Formal models of Web queries. *Information Systems*, 23(8):615–637, 1998.
- [11] R. Rivest. The MD5 Message-Digest Algorithm. Request for Comments 1321, MIT Laboratory for Computer Science, 1992.
- [12] H. Yamana, K. Tamura, H. Kawano, S. Kamei, M. Harada, H. Nishimura, I. Asai, H. Kusumoto, Y. Shinoda, and Y. Muraoka. Experiments of collecting www information using distributed www robots. In *Proceedings of the 21st International ACM SIGIR Conference*, Melbourne, Australia, 1998.
- [13] N. Ziviani. Information systems on mobile computing environments. Technical Report RT.DCC.001/99 PRONEX.SIAM.001/99, Federal University of Minas Gerais, Computer Science Departament, 1999. <http://www.dcc.ufmg.br/siam>.