SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. (in press)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.825

The Viúva Negra crawler: an experience report

Daniel Gomes*,† and Mário J. Silva

Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, Edificio C6, piso 3, Campo Grande, 1749-016 Lisboa, Portugal

SUMMARY

This paper documents hazardous situations on the Web that crawlers must address. This knowledge was accumulated while developing and operating the Viúva Negra (VN) crawler to feed a search engine and a Web archive for the Portuguese Web for four years. The design, implementation and evaluation of the VN crawler are also presented as a case study of a Web crawler design. The case study tested provides crawling techniques that may be useful for the further development of crawlers. Copyright © 2007 John Wiley & Sons, Ltd.

Received 9 August 2006; Revised 9 February 2007; Accepted 4 March 2007

KEY WORDS: crawler; experience; harvesting; Viúva Negra; Tumba; Tomba

1. INTRODUCTION

Web mining systems are crucial for harnessing the information available on the Web. A *crawler* is a software component that iteratively collects information from the Web, downloading pages and following the linked URLs. There are collections of documents gathered from the Web that can relieve Web miners from the crawling task [1], but they become stale quickly and may not contain the desired information. The search engine industry developed crawlers that download fresh information to update indexes, but their descriptions are superficial owing to the competitive nature of this business [2]. The permanent evolution of the Web and the upcoming of new usage contexts demands continuous research in crawling systems.

Although a crawler is conceptually simple, its development is expensive and time consuming because most problems arise when the crawler leaves the experimental environment and begins

*Correspondence to: Daniel Gomes, Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, Edificio C6, piso 3, Campo Grande, 1749-016 Lisboa, Portugal.
†E-mail: dcg@di.fc.ul.pt

Contract/grant sponsor: FCCN-Fundação para a Computação Científica Nacional and FCT-Fundação para a Ciência e Tecnologia; contract/grant number: SFRH/BD/11062/2002

harvesting the Web. The Web is very heterogeneous and there are hazardous situations associated with Web crawling. Some of them are malicious, while others are caused by malfunctioning Web servers or authors that publish information on the Web without realizing that it will be automatically processed by crawlers. Crawler developers must be aware of these situations in order to design robust crawlers capable of coping with them.

The description of hazardous situations relevant to crawling is scarce in the scientific literature, because most experiments are based on simulations or short-term crawls that do not enable their identification. As a result, much simpler models of the Web are usually assumed than observed in reality. Hence, the detection of hazardous situations on the Web is a recurrent problem that must be addressed by every new system developed to process Web data. Moreover, new hazardous situations arise as the Web evolves, so their monitoring and identification requires a continuous effort.

In this paper we describe hazardous situations relevant to crawling and detail the architecture, implementation and evaluation of the Viúva Negra (VN) Web crawler. The main purpose of this study is to share our experience obtained while developing and operating VN to facilitate the development of crawlers in the future. VN was developed and tested during the past four years to gather information for several projects, including a search engine (www.tumba.pt) and an archive for the Portuguese Web (tomba.tumba.pt). Using the crawled information to feed other applications in a production environment enabled the detection of limitations in the crawler and gradually to improvements. The main contributions of this study are as follows.

- A detailed description of situations that are hazardous to crawling and solutions to mitigate their effects. These hazardous situations were presented in the context of crawling but they affect HTTP clients in general. Therefore, the presented solutions can be used to enhanced other systems that process Web data, such as browsers or proxies.
- A flexible and robust crawler architecture that copes with distinct usage contexts.
- Techniques to save on bandwidth and storage space by avoiding the download of duplicates and invalid URLs.

This paper is organized as follows: in the next section we describe situations on the Web that are hazardous to crawling. In Section 3 we discuss partitioning techniques to divide the URL space across the set of processes that compose a distributed crawler. In Section 4 we present the architecture of the VN crawler, its main features and implementation. In Section 5 we present evaluation results of the VN crawler and share the lessons learned while harvesting the Portuguese Web. In Section 6 we present related work and compare our system with other crawlers. Finally, in Section 7, we draw our conclusions and suggest future work.

2. HAZARDOUS SITUATIONS

In this section we describe hazardous situations found on the Web and discuss solutions to mitigate their effects. First, we present examples of situations that cause unnecessary downloads and degrade the performance of a crawler. We then describe contents that are hard to be automatically processed and frequently prevent crawlers from following their embedded links to other contents. Finally, we present a study of heuristics to detect sites with different names that provide the same contents, causing the crawl of a large number of duplicates.

DOI: 10.1002/spe

2.1. Spider traps

Heydon and Najork defined a spider trap as *a URL or set of URLs that cause a crawler to crawl indefinitely* [3]. We relaxed this definition and consider that situations that significantly degrade the performance of a crawler are also spider traps, although they may not originate infinite crawls. Initially, the pages dynamically generated when a server received a request were pointed as the general cause of spider traps and they were excluded from crawls as a preventive measure [4]. Nowadays, dynamic pages are very popular because they enable the management of information in databases independently from the format used for publication. It was estimated that there are 100 times more dynamic pages than static ones [5]. Thus, preventing a crawler from visiting dynamic pages in order to avoid spider traps would exclude a large section of the Web. Some Webmasters create traps to boost the placement of their sites in search engine results [3], while others use traps to repel crawlers because they consume the resources of Web servers. Spider traps bring disadvantages to their creators. A trap compromises the navigability within the site and human users become frustrated if they try to browse a spider trap. Also, search engines have a key role in the promotion of Web sites, and they ban sites containing traps from their indexes [6,7]. Next, we present some examples of spider traps and discuss how to mitigate their effects.

2.1.1. Misuse of DNS wildcards

A zone administrator can use a DNS wildcard to synthesize resource records in response to queries that otherwise do not match an existing domain [8]. In practice, any site under a domain using a wildcard will have an associated IP address, even if nobody registered it. DNS wildcards are used to make sites more accepting of typographical errors because they redirect any request to a site under a given domain to a default doorway page [9]. The usage of DNS wildcards is hazardous to crawlers because they enable the generation of an infinite number of site names to crawl under one single domain. Moreover, it is not possible to query a DNS server to detect if a given domain is using wildcards. However, a crawler should be able to know that a given site is reached through DNS wildcarding before harvesting it. To achieve this, one could execute DNS lookups for a set of absurd sites names under a domain and check if they are mapped to the same IP address. If they are, the domain is most likely using a DNS wildcard. This way, a black list of domain names that use DNS wildcards could be compiled and used to prevent crawlers from harvesting them. However, many domains that use DNS wildcarding also provide valuable sites. For instance, the domain blogspot.com uses DNS wildcarding but also hosts thousands of valuable sites.

2.1.2. Unavailable services and infinite size contents

Malfunctioning sites are the cause of many spider traps. These traps usually generate a large number of URLs that reference a small set of pages containing default error messages. Thus, they are detectable by the abnormally large number of duplicates within the site. For instance, sites that present highly volatile information, such as online stores, generate their pages from information kept in a database. If the database connection breaks, these pages are replaced by default error messages informing that the database is not available. A crawler can mitigate the effects of this kind of traps by not following links within a site when it reaches a limited number of duplicates. On the other hand, a malfunctioning site may start serving contents with a higher latency. This situation would mean that a crawler would

DOI: 10.1002/spe

take too much time to harvest its contents, delaying the overall progress of the crawl. A crawler should impose a limit on the time to harvest each URL to mitigate this problem.

There are also infinite size contents, such as online radio transmissions, that cause traps if a crawler tries to download them. A crawler may truncate the content if it exceeds a maximum limit size. Notice that contents in HTML format can be partially accessed if they are truncated, but executable files become unusable.

2.1.3. Session identifiers

HTTP is a stateless protocol that does not allow the tracking of user reading patterns by itself. However, this is often required by site developers, for instance to build profiles of typical users. A session identifier embedded in the URLs linked from pages allows tracking a sequence of requests from the same user. According to Web rules, a session identifier should follow a specific syntax beginning with the string 'SID:' [10]. In practice, the session identifiers are embedded by developers in URLs in common with any other HTTP parameters. Session identifiers have lifetimes to prevent the situation where different users are identified as the same one. A session identifier replacement means that the URLs linked from the pages are changed to include the new identifier. If the crawl of a site lasts longer than the lifetime of a session identifier, the crawler could be trapped while harvesting the new URLs generated periodically to include the new session identifiers. The replacement of session identifiers also originates duplicates, because the new generated URLs reference the same contents as those which have been crawled previously [11]. A crawler may avoid becoming trapped by not following links within the site once a limit number of duplicates is reached. This heuristic fails if the pages of the site are permanently changing and the new URLs reference distinct contents. In this case, the insertion of new links within the site should be stopped when a limit number of URLs crawled from the site is reached.

2.1.4. Directory list reordering

Apache Web servers generate pages to present lists of files contained in a directory. This feature is used to easily publish files on the Web. Figure 1 presents a directory list and its embedded links. The directory list contains four links to pages specifying by *Name*, *Last-Modified date*, *Size* and *Description*, in ascending or descending order. Thus, if a crawler follows all the links embedded in a directory list page, it will harvest the same information referenced by eight different URLs. Moreover, a directory list enables browsing a file system and may accidentally expose contents that were not meant to be published on the Web. A crawler could become trapped if harvesting, for instance, temporary files periodically generated in a directory. A crawler could exclude URLs that reference directory listings in order to avoid traps but they are frequently used to publish valuable contents, such as open-source software and documentation.

2.1.5. Growing URLs

A spider trap can be set with a symbolic link from a directory /spider to the directory / and a page /index.html that contains a link to the */spider* directory. Following the links will create an infinite number of URLs (www.site.com/spider/spider/. . .) [12]. Although this example may seem rather academic, these traps exist on the Web. We also found advertisement sites that embedded the history of the URLs followed by a user on the links of their pages. The idea is that when users reach a given page

DOI: 10.1002/spe

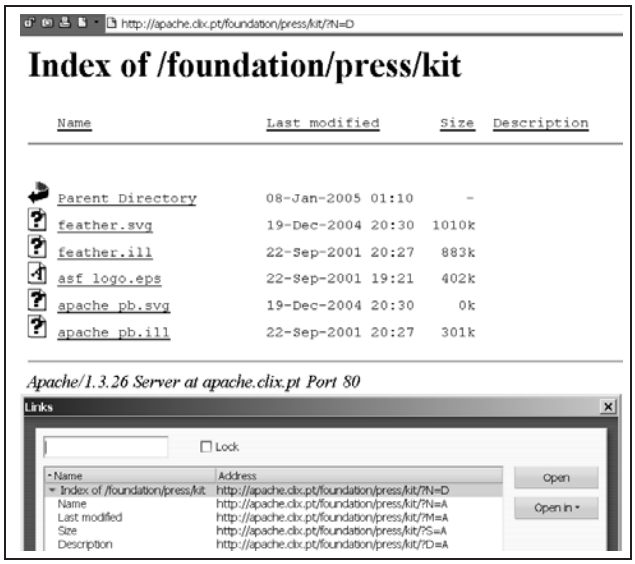

Figure 1. Apache directory list page and the linked URLs.

they stop browsing and the URL that referenced the page contains the history of the URLs previously visited. This information is useful for marketing analysis. The problem is that a crawler never stops 'browsing' and it becomes trapped following the generated links. Hence, a crawler should impose a limit on the length of the URLs harvested.

2.2. Hard to interpret contents

Crawlers interpret the harvested contents to extract valuable data. If a crawler cannot extract the linked URLs from a Web page, it will not be able to iteratively harvest the Web. The extracted texts are important for focused crawlers that use classification algorithms to determine the relevance of the contents. For instance, a focused crawler could be interested in harvesting documents written in a given language or containing a set of words.

However, the extraction of data from Web contents is not straightforward because there are situations on the Web that make contents difficult to interpret.

2.2.1. Wrong identification of media type

The media type of a content is identified through the HTTP header field Content-Type. HTTP clients choose the adequate software to interpret the content according to its media type. For instance, a content with the Content-Type 'application/pdf' is commonly interpreted by the Adobe Acrobat Reader software. However, sometimes the Content-Type values do not correspond to the real media type of the content [13] and a HTTP client may not be able to interpret it correctly. An erroneous Content-Type

DOI: 10.1002/spe

response can be detected through the analysis of the extracted data by probabilistic guessing. A Web page that does not contain any links raises the suspicion that something went wrong. If the text extracted from a content does not contain words from a dictionary or does not contain spaces between sequences of characters, the content may have been incorrectly interpreted. However, the usage of dictionaries may become difficult in a broad context composed by contents written in different languages and using different character encodings. If a crawler identifies an erroneous Content-Type response it may try to identify the correct type to enable the correct interpretation of the content. The format of a content is commonly related to the file name extension of the URL that references it. This information can be used to automatically identify the real media type of the content. However, the usage of file name extensions is not mandatory within URLs and the same file name extension may be used to identify more than one format. For example, the extension .rtf identifies documents in the application/rtf and text/richtext media types. The media type can also be guessed through the analysis of the content. For instance, if the content begins with the string <html> and ends with the string </html> it is most likely an HTML document (text/html media type). This approach requires specific heuristics for each of the many media types available on the Web and identifying the media type of a binary file is unattainable.

2.2.2. *Malformed pages*

A malformed content does not comply with its media type format specification, which may prevent its correct interpretation. Malformed HTML contents are prevalent on the Web. One reason for this fact is that authors commonly validate their pages through visualization on browsers, which tolerate format errors to enable the presentation of pages to humans without visible errors. As a result, the HTML interpreter used by a crawler should be tolerant to common syntax errors, such as unmatched tags [14,15].

2.2.3. *Cloaking*

A cloaking Web server provides different content to crawlers than to other clients. This may be advantageous if the content served is a more crawler-friendly representation of the original. For instance, a Web server can serve a Macromedia Shockwave Flash Movie to a browser and an alternative HTML representation of the content to a crawler. However, spammers use cloaking to deceive search engines without inconveniencing human visitors.

Cloaking may be unintentional. There are Web servers that, when in the presence of an unrecognized user-agent, return a page informing that the client's browser does not support the technology used in the site and suggest the usage of an alternative browser. A crawler may identify itself as a popular browser to avoid suffering from this cloaking situation. However, this solution violates the principles of politeness and webmasters could confuse the consecutive visits of a crawler with an attack to their sites.

2.2.4. *JavaScript-intensive pages*

JavaScript is a programming language created to write functions, embedded in HTML pages that enable the generation of presentations that were not possible using HTML alone. The Asynchronous JavaScript And XML (AJAX) libraries contributed to the widespread usage of this language in

DOI: 10.1002/spe

Web pages [16]. It is now increasingly common to find pages where normal links are JavaScript programs activated through clicking on pictures or selecting options from a drop-down list [17].

A JavaScript program may build a link or a text to be accessed through a series of computational steps. However, writing an interpreter to understand what a JavaScript program is doing is extremely complex and computationally heavy. As consequence, the extraction of data from Web pages written using JavaScript is hard.

2.3. Duplicate hosts

Duplicate hosts (duphosts) are sites with different names that simultaneously serve the same content. Duphosts have been identified as the single largest source of duplicates on the Web [18]. Technically, duphosts can be created through the replication of contents among several machines, the usage of virtual hosts or the creation of DNS wildcards. There are several situations that originate duphosts.

Mirroring. The same contents are published on several sites to backup data, reduce the load on the original site or to be quickly accessible to some users.

Domain squatting. Domain squatters buy domain names desirable to specific businesses, in order to make a profit on their resale. The requests to these domains are redirected to a site that presents a sale proposal. On their turn, companies also register multiple domain names related to their trade marks and point them to the company's site to protect against squatters.

Temporary sites. Web designers buy domains for their customers and point them temporally to the designer's site or to a default 'under construction' page. When the customer's site is deployed the domain starts referencing it.

The detection of duphosts within a Web data set can be used to improve Information Retrieval algorithms. For instance, search engines can avoid presenting the same information published in duphosts as different search results. Crawlers should avoid crawling duphosts to save on bandwidth and storage space. Previous works have presented algorithms to detect duphosts within a set of documents harvested from the Web [19,20]. However, preventing a crawler from harvesting duphosts is more difficult than detecting them on a static data set because a list of duphosts extracted from a previously compiled Web collection may not reflect the current state of the Web. Meanwhile, sites identified as duphosts may have disappeared or start presenting distinct contents.

Next, we present heuristics to identify duphosts within a data set and evaluate their application in Web crawling. The experimental data set was composed by 3.3 million pages crawled from 66 370 sites. We compared the intersection of content fingerprint signatures between sites to derive a list of pairs of duphosts, where the first site is considered a *replica* of the second one, nominated as the *original*. The election of the original within a pair of duphosts is arbitrary because they both provide the same contents. We analyzed three heuristics to detect if two sites were duphosts.

SameHome. Both sites present equal home pages. As the home page describes the content of a site then if two sites have the same home page they probably present the same contents. However, there are home pages that permanently change their content, for instance to include advertisements, and two home pages in the data set may be different although the remaining contents of the sites are equal. On the other hand, there are sites within the data set composed by a single transient 'under construction' home page, that in a short notice after the data set was built, begin presenting distinct and independent contents.

DOI: 10.1002/spe

Table I. Results from the five approaches to detect duphosts.

Heuristic	Invalid IP (in %)	Percentage of duphosts	Precision (in %)	Relative coverage
SameHome	6.7	4.8	68	6.6
SameHomeAnd1Doc	4.4	3.9	92	2.1
Dups60	4	3.7	90	2.5
Dups80	4.1	2.2	92	2.0
Dups100	4.7	0.4	94	1.0

SameHomeAnd1Doc. Both sites present equal home pages and at least one other equal content. This approach follows the same intuition as SameHome for the home pages but tries to overcome the problem of transient duphosts composed by a single page.

DupsP. Both sites present a minimum percentage (P) of equal content and have at least two equal contents. Between the crawl of the duphosts to build the data set, some pages may change, including the home page. This approach assumes that if the majority of the content is equal between two sites, they are duphosts. We considered a minimum of two equal documents to reduce the presence of sites under construction.

We extracted five lists of duphosts following the heuristics SameHome, SameHomeAnd1Doc and DupsP considering levels of duplication of 60%, 80% and 100%. The proposed heuristics were evaluated by simulating their application on a crawl executed 98 days after the creation of the data set. Table I summarizes the obtained results. We executed a DNS lookup for each site on the duphosts lists and excluded those that did not have an associated IP address because a crawler would not be able to harvest them. On average 4.8% of the pairs were no longer valid because one of the duphosts did not have an associated IP address (Invalid IP column). The third column of Table I presents the percentage of the total number of sites in the data set that were replicas identified through each heuristic after the IP check. On average, 1841 replicas were identified, which represents 2.8% of the sites found within the data set. In order to measure the precision of each heuristic we randomly chose 50 pairs of duphosts from each list and visited them simultaneously to verify if they still presented the same contents (Precision column). We used the lowest number of pairs detected (Dups100) as a baseline to compare coverage (Relative coverage column). The SameHome heuristic achieved the maximum relative coverage (6.6) but the lowest precision value (68%). When we imposed that at least one content besides the home page must exist on both sites (SameHomeAnd1Doc), the relative coverage decreased to 2.1 but the precision improved to 92%. The Dups100 heuristic detected sites that shared all the contents and it achieved the highest precision of 94%. The remaining 6% of the pairs referenced sites that were no longer online, although they still had an associated IP address. As we decreased the threshold of duplication we identified more replicas, maintaining the precision over 90%, as we can see in the third and fourth lines of Table I.

The SameHome heuristic is an inexpensive way to detect duphosts because it requires the comparison of just one page per site. However, it is the most prone to identify transient duphosts originated by single-page sites under construction. The detection of duphosts imposes an overhead

DOI: 10.1002/spe

Table II. Consequences of the normalization of the usage of the WWW prefix in site names.

Domain level	Percentage duphosts avoid	Percentage sites lost
2nd level	30	4
3nd level	17	16

on the crawler and avoiding the crawl of a duphost containing one single page may not pay-off. The SameHomeAnd1Doc overcomes this problem at the cost of comparing more contents per site. The number of duphosts decreases as the threshold of duplicates required between sites increases. At the same time, precision is improved. Owing to the permanent changes that occur on the Web, we believe that the effectiveness of the proposed heuristics to avoid the crawl of duphosts depends on the age of the data set used to extract the list of duphosts.

2.3.1. The WWW prefix

The most common reason for duphosts lies on site names that just differ on the prefix 'www.'. This corresponds to 51% of the names of the duphosts detected on the previous experiments. It is recommended that World Wide Web site names begin with the prefix 'www.' [21]. Therefore one way to avoid the crawl of duphosts is to normalize the URLs to visit by appending the 'www.' prefix when it is not present. However, there are site names that use a different prefix and this change could generate invalid site names, excluding valid URLs from the crawl.

We ran an experiment to evaluate the application of this heuristic to the prevention of duphosts. The experimental data set was composed by two lists of second-level domains (e.g. domain.pt) and third-level domains (e.g. subdomain.domain.pt) from the official registries. We generated a list of home page URLs referencing the domain names and the domain names with the prefix 'www.' and crawled it. Table II presents the obtained results. The normalization heuristic applied to the second-level domains avoided the crawl of 30% of duphosts and 4% of the sites were excluded because they were not available with a name containing the 'www.' prefix. For the third-level domains, just 17% of the sites were duphosts due to the usage of the 'www.' prefix and 16% were lost owing to the normalization process. The results suggest that the success of appending the prefix 'www.' to avoid duphosts depends on the domain level of the site name.

3. WEB PARTITIONING STRATEGIES

Designing a crawler to harvest a small set of well-defined URLs is simple. However, the required information is usually spread across millions of pages and most crawlers adopt distributed architectures that enable parallel crawling to cope with the large size of the Web. Although there are several types of crawlers [3,22,23], they all share ethical principles. A crawler must be polite, robust, configurable, scalable and flexible [24].

DOI: 10.1002/spe

A crawler is composed by a *Frontier*, which manages the URLs, and *Crawling Processes* (CPs) that iteratively harvest documents from the Web for local storage. A CP iteratively obtains a seed from the Frontier, downloads the referenced document, parses it, extracts the linked URLs and inserts them in the Frontier to be harvested. A crawl finishes when there are no URLs left to visit or a limit date is reached. A simple crawler can be assembled from only one CP and one Frontier, and it might be suitable for storing local copies of sites by individual users. However, the download rate provided by this architecture is not scalable. Large-scale crawlers parallelize Web crawling using several CPs at the cost of increased complexity. The URL space must be partitioned to enable parallel harvesting and the CPs must be synchronized to prevent multiple harvests of the same URLs. The Frontier is the central data structure of a crawler. Some URLs are linked from many different pages. Thus, every time a CP extracts an URL from a link embedded in a page, it must verify if the URL already existed in the Frontier to prevent overlapping. This verification is known as the *URL-seen test* and demands permanent access to the Frontier [3].

The URL space of the Web may be partitioned for parallel crawling. A partitioning function maps an URL to a class of a partition. The partitioning strategy has implications for the operation of the crawler [25]. The main objective of partitioning the URL space is to distribute the workload among the CPs creating classes for groups of URLs that can be harvested independently. After partitioning, each CP is responsible for harvesting exclusively one class at a time. In general, the following partitioning strategies may be followed.

IP partitioning. Each partition class contains the URLs hosted on a given IP address.

Site partitioning. Each partition class contains the URLs of a site, considering that a site is composed by the URLs that share the same site name. This partitioning schema differs from the above, because the same Web server may host several sites on the same IP address (virtual hosts) and each will be crawled separately.

Page partitioning. Each partition class contains a fixed number of URLs independently from their physical location. A partition may contain URLs hosted on different sites and IP addresses. Page partitioning is suitable for harvesting a selected set of pages spread on the Web.

A CP may exhaust its resources while trying to harvest a partition class containing an abnormally large number of URLs. Thus, the number of the URLs contained in a partition class should be ideally constant to facilitate load balancing. The page partitioning strategy is the one that best meets this criterion. The IP partitioning tends to create some extremely large classes due to servers that host thousands of sites, such as Geocities (www.geocities.com) or Blogger (www.blogger.com). The site partitioning strategy is more likely to create partition classes containing a single URL, due to sites under construction or presenting an error message. The efficiency of the IP and site partitioning strategies depend on the characteristics of the portion of the Web to crawl. However, these characteristics may be unknown, which makes it difficult to predict their impact on the performance of the crawler. Table III summarizes the relative merits of each strategy, which are characterized by the following determinants.

DNS caching. A CP executes a DNS lookup to map the site name contained in an URL into an IP address, establishes a TCP connection to the correspondent Web server and then downloads the content. The DNS lookups are responsible for 33% of the time spent to download content [26]. Hence, caching a DNS response and using it to download several documents from the same site optimizes Web crawling. A CP does not execute any DNS lookup during the crawl when harvesting an IP partition class because

DOI: 10.1002/spe

Table III. Comparison of the partitioning schemes.

Partitioning strategy	DNS caching	Use keep-alive connections	Avoid server overloading	Reuse site meta-data	Independency
IP	++	++	++	+	−
Site	+	+	+	++	++
Page	−	−	−	−	++

all the URLs are hosted on the IP address that identifies the partition. A site partition class requires one DNS lookup to be harvested because all its URLs have the same site name. A page partition contains URLs from several different sites, so a CP would not benefit from caching DNS responses.

Use of keep-alive connections. Establishing a TCP connection to a Web server takes on average 23% of the time spent to download a content [26]. However, HTTP keep-alive connections enable the download of several documents reusing the same TCP connection to a server [27]. A page partition class contains URLs hosted on different servers, so a CP does not benefit from using keep-alive connections. On the other hand, with IP partitioning an entire server can be crawled through one single keep-alive connection. When a crawler uses a site partitioning strategy, a single keep-alive connection can be used to crawl a site. However, the same Web server may be configured to host several virtual hosts. Then, each site will be crawled through a new connection.

Server overloading. In general, a crawler should respect a minimum interval of time between consecutive requests to the same Web server to avoid overloading it. This is called the courtesy pause. The page partitioning strategy is not suitable to guarantee courtesy pauses, because the URLs of a server are in general spread among several partition classes. Thus, if no further synchronization mechanism is available, the CPs may crawl the same server simultaneously, disrespecting the courtesy pause. The page partitioning strategy requires that each CP keeps track of the requests executed by other CPs in order to respect the courtesy pause. With the IP partitioning strategy, it is easier to respect the courtesy pause because each CP harvests exclusively the URLs of a Web server and simply needs to register the time of the last executed request to respect the courtesy pause. A crawler using a site partitioning strategy respects at first sight a minimum interval of time between requests to the same site but a server containing virtual hosts may be overloaded with requests from CPs that harvest its sites in parallel. On the other hand, a Web server containing virtual hosts should be designed to support parallel visits to its sites performed by human users. Hence, it should not become overloaded with the parallel visits executed by the CPs.

Reuse of site meta-data. Sites contain meta-data, such as the Robots Exclusion file [28], which influence crawling. The page partitioning strategy is not suitable to reuse the meta-data of a site because the URLs of a site are spread across several partitions. With the IP partitioning strategy, the meta-data of a site can be reused, but it requires additional data structures to keep the correspondence between the sites and the meta-data. Notice, however, that this data structure can grow considerably, because there are IP partition classes that contain thousands of different sites generated through virtual hosting.

DOI: 10.1002/spe

On its turn, when a crawler is harvesting a site partition class, the site's meta-data is reused and the crawler just needs to manage the meta-data of a single site.

Independency. The site and page partitioning strategies enable the assignment of an URL to a partition class independently from external resources. The IP partitioning depends on the DNS servers to retrieve the IP address of an URL and it cannot be applied if the DNS server becomes unavailable. If the site of an URL is relocated to a different IP address during a crawl, two invocations of the function for the same URL would return different partitions. In this case, the URLs hosted on the same server would be harvested by different CPs.

Initially, each partition class contains only a set of seeds. A partition class is assigned to a CP that becomes responsible for harvesting the correspondent URLs. The assignment process can be *static* or *dynamic* [29]. In the static assignment, the partitions are assigned before the beginning of the crawl. Each CP knows its partition class and assigns the URLs extracted from Web pages to the partition classes responsible for their crawl. The static assignment imposes that the number of CPs is constant during the crawl to guarantee that all the partition classes are harvested. The partition classes assigned to a CP would not be harvested if it failed and could not be recovered. Moreover, one cannot increase the number of CPs to accelerate a crawl, because all the partitions were mapped to the initial set of CPs. In the dynamic assignment, a central coordinator assigns the partition classes during the crawl. This approach supports having a variable number of CPs. The performance of the system degrades if a CP fails but this does not compromise the coverage of the crawl.

4. THE VN CRAWLER

This section details the design and implementation of the VN crawler. VN was conceived to be used in research projects requiring the harvesting of Web data, fulfilling the requirements of different applications. VN has a hybrid Frontier, adopts a site partitioning strategy and dynamic-pull assignment.

Hybrid frontier. The Frontier is distributed among several Local Frontiers that periodically synchronize with a central Global Frontier. This approach does not concentrate the load of the URL-seen test on a single component. It does not either require frequent synchronization among the Local Frontiers. Each CP has an associated Local Frontier where it stores the meta-data generated during the crawl of a partition. The meta-data on the seeds and crawled URLs is stored on the Global Frontier.

Site partitioning strategy. Besides the previously discussed advantages (Section 3), three additional reasons lead us to adopt the site partitioning strategy. First, a CP frequently accesses the Local Frontier to execute the URL-seen test. As sites are typically small [30], the Local Frontier can be maintained in memory during the crawl of the site to optimize the execution of the URL-seen test. Second, Web servers are configured to support access patterns typical of human browsing. The crawling of one site at a time enables the reproduction of the behavior of browsers, so that the actions of the crawler do not disturb the normal operation of Web servers. Third, site partitioning facilitates the implementation of robust measures against spider traps.

Dynamic-pull assignment. The Global Frontier assigns partition classes to CPs as they pull them. The Global Frontier guarantees that a partition class is never harvested simultaneously by two CPs. The coordinator sends partition classes to the CPs.

DOI: 10.1002/spe

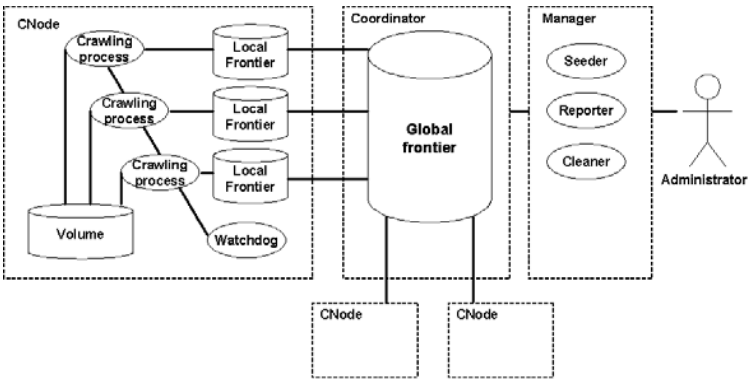

Figure 2. Architecture overview.

Figure 2 describes VN's architecture. It comprises a Global Frontier, a *Manager* that provides tools to execute administrative tasks, such as monitoring the progress of a crawl, and several *Crawling Nodes* (CNodes) that host the CPs, Local Frontiers, *Volumes* that store the harvested data and a *Watchdog* that restarts the CPs if they are considered dead (inactive for a given period of time). The scheduling of the execution of the CPs within a CNode is delegated to the operating system. We assume that when a CP is blocked, for instance while executing IO operations, another CP is executed. The *Seeder* generates seeds to a new crawl from user submissions, DNS listings and home pages of previously crawled sites and inserts them in the Global Frontier. The *Reporter* obtains statistics on the state of the system and emails them to a human *Administrator*. The *Cleaner* releases resources acquired by faulty Crawling Processes.

4.1. Crawling algorithm

The CPs harvest information from the Web visiting one site at a time in a breadth-first mode. Figure 3 illustrates this process. A CP begins the crawl of a new site partition class by transferring a seed from the Global Frontier to its Local Frontier. We say that the partition class was *checked-out*. The Global Frontier identifies each partition class with the site's hash and manages three lists: (i) partition classes to crawl; (ii) partition classes being crawled; and (iii) partition classes crawled. When a CP checks out a partition class it is moved from the first to the second list. The check-in moves the partition from the second to the third list.

After the check-out, the CP downloads the 'robots.txt' file (Robots Exclusion Protocol) and then iteratively harvests one URL at a time from the site until there are no URLs to visit (*loop*). The CP launches a *Collector* thread that downloads and processes information referenced by an URL. The Collector requests the HTTP headers of the URL to check if the content should be downloaded. For instance, if the content is an MP3 file and the selection criteria defines that only HTML pages should be harvested, the content is not downloaded. Then, the content is parsed to extract various meta-data, such as links to external pages. The extraction and storage of meta-data from the contents during the crawl, while they are stored in memory, avoids redundant processing by the applications

DOI: 10.1002/spe

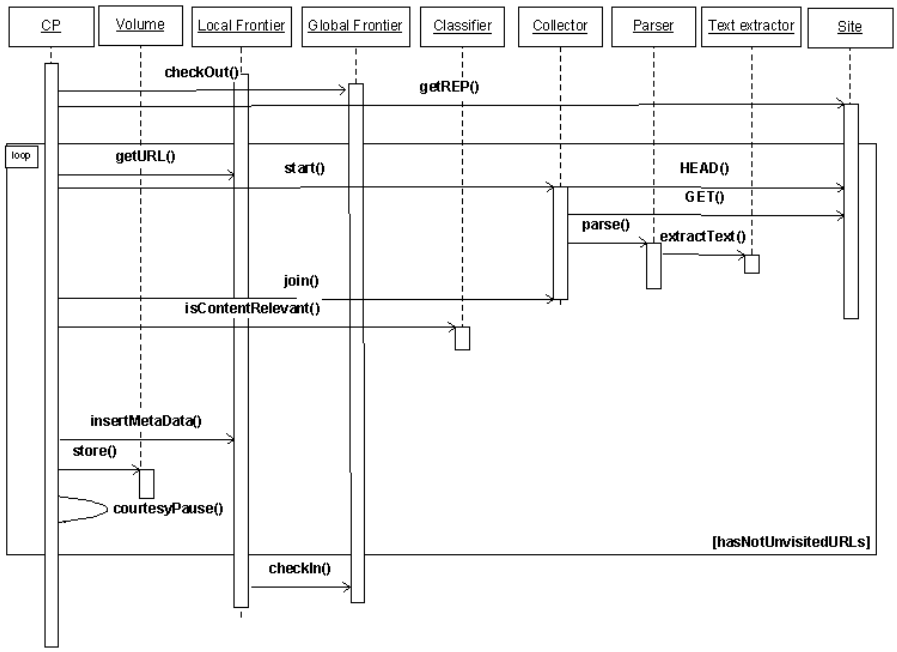

Figure 3. Sequence diagram: crawling a site.

that will latter process the Web data. Finally, the Collector returns the content and extracted meta-data to the CP. This information is analyzed by a *Classifier* that checks if the content matches the selection criteria. If the content is considered relevant, it is stored in a *Volume* and the meta-data is inserted in the Local Frontier, otherwise it is discarded. The CP sleeps after crawling each URL to execute a courtesy pause.

The crawl of the site finishes when all its URLs were visited. Then, the correspondent meta-data is transferred to the Global Frontier (check-in). A CP successively checks-out a seed, harvests the partition class and checks-in the resultant meta-data, until there are no unvisited seeds in the Global Frontier.

4.2. Fault management

VN was designed to tolerate faults at different levels in its components without jeopardizing the progress of the crawl. This way, it is able to face hazardous situations while crawling the Web and possible hardware problems on the underlying cluster of machines.

A CP launches an independent thread (Collector) to execute sensitive tasks, such as the download, meta-data extraction and parsing of a content. The CP terminates the execution of the Collector after a limited time. Hence, a fault caused by the harvest and processing of a single content does not

DOI: 10.1002/spe

compromise the crawl of the remaining contents at the site. However, this imposes the overhead of launching a new thread to crawl each URL.

The CPs are independent from each other and the failure of one of them does not influence the execution of the rest. However, they depend on the Global Frontier for synchronization. A CP operates independently from the Global Frontier between the check-out and the check-in events. A fault of the Global Frontier causes a gradual degradation of the system because the CPs will continue harvesting until the check-in is attempted. As a result, there is an interval of time when it is possible to recover the Global Frontier without stopping the crawl. For instance, in our environment the network cable was once disconnected from the machine hosting the Global Frontier and the incident was solved without influencing the progress of the crawl.

A CP acquires an exclusive lock on the check-out of a partition to prevent simultaneous harvests of the same site. If a CP fails before the check-in, the site that was being visited remains locked, preventing other CPs from visiting it. The site's meta-data kept in the Local Frontier (in-memory) is lost and the contents stored in the Volume become orphans because the references to them disappear. The VN Administrator can unlock the sites checked-out by the faulty CPs so that they can be crawled again the orphan contents can be deleted using the Cleaner. The sites are unlocked on-demand to avoid repeated crawls of problematic sites that cause failures on the crawler.

In practice, we first crawl all the sites and then we revisit those that could not be completely harvested because there was a CP failure. This way, a page is downloaded twice from the Web only if a CP visited it on the first run but died before the check-in of the site.

The contents can be stored locally on the same machine that hosts the CP or remotely on any other machine that hosts a Volume. Therefore, if the Volume used by a CP fails, for instance because it exhausted its storage capacity, the CP can be quickly set to store contents on remote Volumes, without requiring any data migration.

4.3. URL extraction

URL extraction is an important task because a crawler finds contents by following URLs extracted from links embedded in Web pages. However, there are URLs extracted from Web pages that should be processed before being inserted in the Frontier to improve the crawler's performance. There are invalid URLs that reference contents that cannot be downloaded. A crawler will waste resources trying to crawl these URLs, so their presence in the Frontier should be minimized. Invalid URLs were pruned using the following strategies.

Discarding malformed URLs. A malformed URL is syntactically incorrect [31]. For instance, an URL containing white spaces. However, there are Web servers that return contents in response to malformed URLs and cause the existence of valid URLs on the Web that are syntactically incorrect.

Discarding URLs that reference unregistered sites. The site name of an URL must be registered in the DNS. Otherwise, the crawler would not be able to map the domain name into an IP address to establish a connection to the server and download the content. Thus, an URL referencing an unregistered site name is invalid. However, testing if the site names of the URLs are registered before inserting them into the Frontier imposed an additional overhead on the DNS servers.

Duplicates occur when two or more different URLs reference the same bytewise equal content. A crawler should avoid harvesting duplicates to save on processing, bandwidth and storage space.

DOI: 10.1002/spe

The crawling of duplicates can be avoided through the normalization of URLs. We added the following rules to the standard URL normalization rules [31].

1. *Add trailing '/' when the path is empty*. The HTTP specification states that if the path name is not present in the URL, it must be given as '/' when used as a request for a resource [27]. Hence, the transformation must be done by the client before sending a request. This rule of normalization prevents URLs, such as www.site.com and www.site.com/, originating duplicates.
2. *Remove trailing anchors*. Anchors are used to reference a part of a page (e.g. www.site.com/file#anchor). However, the crawling of URLs that differ only on the anchors would result in repeated downloads of the same page.
3. *Add prefix 'www.' to site names that are second-level domains*. We observed that most of the sites named with a second-level domain are also available under the site name with the prefix 'www.' (see Section 2.3.1).
4. *Remove well-known trailing file names*. Two URLs that are equal except for a well-known trailing file name such as 'index.html', 'index.htm', 'index.shtml', 'default.html' or 'default.htm', usually reference the same content. The results obtained in our experiments showed that removing these trailing file names reduced the number of duplicates by 36%. However, it is technically possible that the URLs with and without the trailing file reference different contents. We did not find any situation of this kind in our experiments, so we assumed that this heuristic does not reduce the coverage of a crawler noticeably.

A request to an URL may result in a redirect response, (3** HTTP response code), to a different one named the *target URL*. For instance, the requests to URLs such as www.somesite.com/dir, where dir is a directory, commonly result in a redirect response (301 Moved Permanently) to the URL www.somesite.com/dir/. Browsers follow the redirects automatically, so they are not detected by the users. VN can also automatically follow a redirect response to download the content referenced by the target URL. However, both the redirect and the correspondent target URLs reference the same content. If they are linked from Web pages, the same content will be downloaded twice. We observed that automatically following redirects during a Web crawl increased the number of duplicates by 26%. On the other hand, when a crawler does not follow redirects, it considers that a redirect is equivalent to a link to a URL. The crawler inserts both the redirect and target URLs in the Frontier: the former is marked as a redirect and the target URL is visited to download the content.

4.4. URL-seen test

The URL-seen test is executed in two steps: first, when the URLs are inserted in the Local Frontier and upon the check-in to the Global Frontier. 81.5% of the links embedded in Web pages reference URLs internal to its site [32]. The URL-seen test for internal URLs is done locally because all the seen URLs belonging to the site are covered by the Local Frontier. So, when the CP finishes crawling the site it can check-in the internal URLs to the Global Frontier, without further testing. However, the URL-seen test for the external URLs must be executed against all the URLs in the Global Frontier during check-in, because the URLs may have been inserted there by another CP in the intervening period. Thus, the URL-seen test for external URLs is an expensive operation and the number of external URLs to check-in should be minimized. Nonetheless, the external URLs are important because they are potential seeds to newly found sites. There are three policies for the insertion of external URLs in the Frontier.

DOI: 10.1002/spe

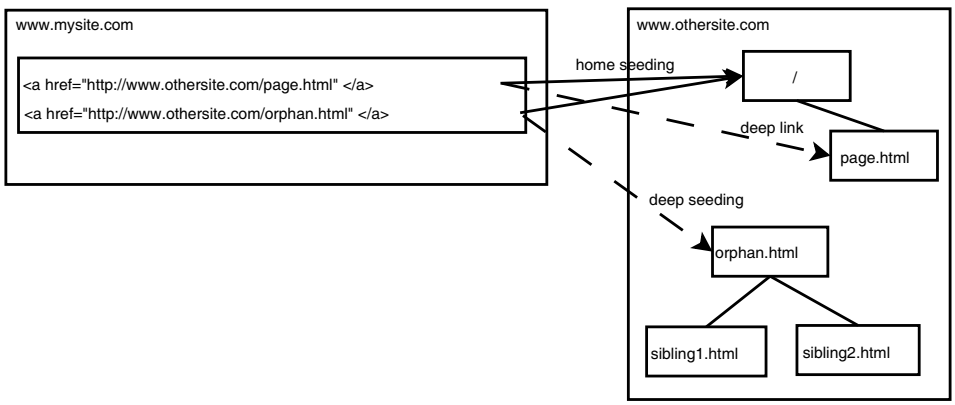

Figure 4. Deep versus home page policy.

Home page. The home page policy assumes that all the contents within a site are accessible through a link path from its home page. Hence, a CP replaces every external URL by its site home page before inserting it in the Local Frontier (Figure 4). The home page policy reduces the number of external URLs to check-in. However, if a CP cannot follow links from the home page, the remaining pages of the site will not be harvested.

Deep link. A deep link references an external URL that is different from the home page. The deep link policy assumes that there are pages that are not accessible through a link path from the home page of the site. The CP inserts the external URLs without any change in the Local Frontier to maximize the coverage of the crawl. For instance, in Figure 4 the URL www.othersite.com/orphan.html is not accessible from the home page of the site but it is linked from the site www.mysite.com. However, if the external URL references a page without links, such as a PDF document, the crawl of the site would be limited to this document. Some authors believe they make pages unavailable by removing the internal links to them, forgetting that external pages may maintain links to these pages. The deep link policy enables the crawling of these supposedly unavailable pages and may expose them in search engine results.

Combined. The policy follows deep links but always visits the home page of the sites. Even if a deep link references a content without links, the remainder of the site accessible through a link path from the home page will be harvested. According to Cothey [33], crawlers that follow this policy can be categorized as *augmented crawlers* because they expand one referenced URL to several URLs to maximize coverage.

VN supports the home page and combined policies. As an optimization, when VN is configured to follow the home page policy, it discards the external URLs hosted on the sites contained in the seeds of the crawl, because they were already inserted in the Global Frontier by the Seeder. Discarding external URLs contained in the initial seed list breaks the deep link and combined policies, because a link may reference a page from a site contained in the initial seed list that is not accessible from the home page.

DOI: 10.1002/spe

In general, the adoption of the combined policy gives a marginal gain of coverage against the home page policy, because just 29.5% of the external links are deep links [30] and pages are usually accessible through a link path from its site home page. Hence, the home page policy is suitable for most crawling contexts, while the combined policy should be used when coverage needs to be maximized (e.g. to execute exhaustive crawls of corporate intranets).

4.5. Addressing hazardous situations

VN avoids getting trapped by being polite, integrating human intervention and using Web characterizations to detect abnormal behaviors that suggest the presence of spider traps. A crawler can be prevented from visiting a site through the Robots Exclusion Protocol (REP) but spammers that use crawlers to gather email addresses from Web pages usually ignore these restrictions. Hence, some webmasters create traps to punish crawlers that do not respect the imposed access rules. For instance, a webmaster creates a directory containing a trap and forbids all the robots from visiting it. If a crawler ignores this restriction, it will get trapped while trying to crawl the contents of the directory. On the other hand, the REP is also used to prevent the crawl of sites under construction or containing infinite contents, such as online calendars. Thus, respecting the REP prevents crawlers from getting trapped, besides being a politeness requirement. According to the REP, the robots.txt file should follow a strict structure to enable its automatic processing. However, we observed that 64.3% of the REP files were not compliant with the specification [34]. Thus, VN tolerates common syntax errors in the robots.txt file, to respect the restrictions imposed by the webmasters. When the robots.txt file cannot be interpreted, the site is crawled without restrictions. The indications provided by the meta-tag ROBOTS contained in pages are also respected [28]. VN identifies itself through the user-agent header field contained in the HTTP requests, providing its name, software version and the URL of a Web page that exposes the purpose of the crawler.

The creation of traps that look like ordinary sites combines technologies, such as virtual hosts and dynamic pages, that require human intervention to be detected. VN receives a black list of URLs, domains and IP addresses not to be visited during the crawl because human experts detected they contained traps. It does not visit sites hosted on IP addresses that are the target of top-level domains that use DNS wildcarding. The black list can be augmented after the beginning of the crawl. However, excluding a list of URLs from a site being crawled, for instance to respond to an angry site administrator, requires killing the CP that is harvesting the site. VN receives a list of duphosts and during the crawl replaces the replicas in the URLs by the originals.

Spider traps are characterized by presenting abnormally large values for some Web characterization metrics. For instance, a site containing a spider trap usually exposes a large number of URLs [3]. If VN detects that the site being crawled contains a trap, it stops following the links within the site. The content harvested prior to the detection of the trap are kept, because they might have valuable information. A spider trap is detected if the site exceeds configurable thresholds for the number of URL and duplicates. The thresholds that denounce the existence of a spider trap must be carefully determined according to the characteristics of the portion of the Web being harvested and periodically updated to reflect its evolution. A malfunction on a site may mean that the contents start being served with a high latency. This situation would mean that a crawler would take too much time to harvest its contents, delaying the overall progress of the crawl. Hence, VN imposes a limit time to harvest each URL. VN truncates contents if they exceed a maximum limit size. It does not try to correct erroneous

DOI: 10.1002/spe

Content-Type answers, instead it discards contents that originated errors on the interpretation software. VN excludes strings longer than that 30 characters that did not contain at least one space from the extracted texts to prevent classification errors but tolerates common format errors in HTML documents to extract meta-data, such as unmatched tags [14].

VN trims URLs of Apache directory reorderings at the last slash. For instance, the URL apache.clix.pt/foundation/press/kit/?N=D is converted to apache.clix.pt/foundation/press/kit/. Given the heterogeneity of Web server software available on the Web, this conversion may seem too specific, but according to the Netcraft survey executed in November 2005, 70% of the sites use Apache Web servers [35]. The results obtained from a crawl of 1.2 million contents showed that 0.5% of its URLs referenced directory list reorderings. VN also avoids URLs that grow incrementally by discarding those that exceed a maximum length.

When using a previous version of VN, we observed that 27% of the URLs visited in a crawl of 7 million pages contained well-known session identifier parameter names (phpsessid, sid, sessionid). Interestingly, 95% of them were hosted in sites developed with PHP engines. We visited some of these sites and found that the links of the pages were changed by the Web server to contain session identifiers when the HTTP client did not accept cookies. A cookie is a piece of data sent by the site that is stored in the client and enables tracking user sessions without URL changes. We enhanced VN to accept cookies and observed that the percentage of URLs containing session identifiers dropped to 3.5%. This had the noteworthy effect of reducing the average URL length from 74 to 62 characters, which saved space on the data structures in the Frontiers.

4.6. Implementation

The VN Web crawler integrates components developed within our research group and external software. It was mainly written in Java using jdk1.4.2, but it also includes software components implemented in native code. The CPs consist of approximately 5000 lines of Java code. They use hash tables to keep the list of duphosts and the DNS cache. The Parser was based on WebCAT, a Java package for extracting and mining meta-data from Web documents [14]. The Classifier used to harvest the Portuguese Web includes a language identifier [36]. The Robots Exclusion file interpreter was generated using Jlex [37]. The Seeder and the Cleaner are Java applications. The Reporter and Watchdog were implemented using shell scripts that invoke operating system commands (e.g. ps, iostat).

Versus is a distributed Web repository developed within our research group to manage the meta-data kept in the Frontiers [38]. The information crawled and related meta-data becomes immediately available for processing by other applications after it is checked-in to Versus. It was implemented using relational databases: the Local Frontier uses the HypersonicSQL in-memory Java database engine [39] and the Global Frontier is based on Oracle 9i [40]. Versus provides access methods to its clients through a Java library API that hides the underlying technologies.

The Volumes were implemented using *Webstore*, a distributed content manager composed by several volume servers written in Java, that eliminates duplicates at storage level [41].

5. EVALUATION

This section analyzes the results gathered from crawls of the Portuguese Web executed during June and July 2005, with the purpose of evaluating the performance of our Web crawler on a real usage scenario.

DOI: 10.1002/spe

Table IV. Hardware used to implement VN.

Configuration	Number of machines	CPU (GHz)	Memory (GB)	Disk speed (rpm)	Storage (GB)
1	1	2 × P4-2.4	4	SCSI 10 000	5 × 73
2	4	1 × P4-2.4	1.5	IDE 7 200	2 × 180
3	2	2 × P4-2.4	2	IDE 7 200	5 × 250
4	1	2 × P3-1.26	1	SCSI 15 000	2 × 18
5	1	2 × P3-1.26	4	SCSI 10 000	5 × 73

We performed experiments to detect bottlenecks, malfunctions and tune its configuration according to the characteristics of the harvested portion of the Web.

Table IV summarizes the hardware configurations of the nine machines used to support VN in our experiments. The machines used the RedHat Linux operating system with kernel 2.4. The machine with Configuration 5 hosts the Global Frontier and the remaining host CNodes. The machines were interconnected through a 100 Mbps Ethernet and accessed the Internet through a 34 Mbps ATM connection shared with other customers of the data center where they were hosted.

VN was configured to use the home page policy and gather contents from several media types convertible to text. It collected statistics for Web characterization, stored original contents for archival and extracted texts for indexing. A content was considered as part of the Portuguese Web if it was hosted on a site under the .PT domain or written in the Portuguese language hosted in other domains, linked from .PT [42]. The thresholds that prevent VN against hazardous situations were determined based on the results of previous works. At most 5000 URLs can be crawled per site because most will have a smaller number of documents [42,43]. We considered that the level of depth of an URL is the number of links followed in breadth-first search order from the seed of the site until the URL is reached by the crawler. We limited to URL depth to five [44]. The maximum size allowed for the contents used in Mercator crawler was 1 MB [3]. We limited the size of the contents to 2 MB, because the size of the content has been growing in past years [42]. We configured a limit number of 10 duplicates per site, 60 s to download each content and 200 characters for the URL length. The CPs were considered dead and restarted by the Watchdogs if they remained inactive for more than 5 min. VN was configured to respect a courtesy pause of 2 s between requests to the same site. Users frequently visit subsequent pages with a time gap of 1 s [45] and a page contains on average 20 embedded images that must also be downloaded [46–48]. Assuming that a browser executes these downloads sequentially, the time interval between its requests to the server is just 0.05 s. Based on this result, the defined courtesy pause of 2 s for VN imposes less load on the Web servers than human browsing.

5.1. Bottlenecks

The Global Frontier is the central point of coordination of VN. Therefore, it is a potential bottleneck that may compromise the scalability of the crawler. We tested the scalability of the VN's architecture by measuring the number of downloads and amount of data crawled within 24 hours, with an increasing number of CPs spread across several machines. We added a new machine hosting 20 CPs to the cluster daily. Figure 5 presents the results obtained. VN scaled up to 160 CPs executing 2.169 million

DOI: 10.1002/spe

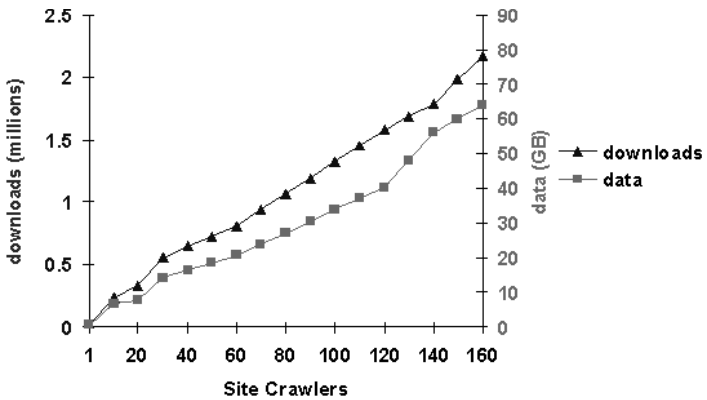

Figure 5. Scalability of the download rate with the addition of new CNodes.

downloads per day (63 GB). This shows that VN's architecture is scalable and the Global Frontier is not a bottleneck within the observed range.

We also measured the duration of the main operations executed during the crawl and the load they imposed on the underlying operating system and hardware. The objective of these experiments was to detect bottlenecks in the components of VN. The experimental setup was composed by four machines with Configuration 2. Each one hosted an increasing number of CPs.

Figure 6(a) presents the amount of data crawled within 24 h by each set of CPs and the corresponding average duration of the operations executed by them. The download rate increased from 1 to 10 CPs. The system tools indicated that the CPU of the machine was exhausted at this point. However, the number of downloads still increased up to 20 CPs. For 30 CPs there is a clear performance degradation. We monitored the Watchdogs to verify if there were CPs considered dead (inactive for more than 5 min) due to starvation caused by the operating system scheduler but we found no relation. The *StoreOriginal* and *StoreText* series of Figure 6(b) present the time spent to store the contents and the corresponding extracted texts in the Volume. The *Processing* series includes parsing of pages, check-operations, interpretation of the REP and meta-data extraction. The *Download* series includes the establishment of a connection to a Web server, the download of the header and content. Before executing a request to a site, the CP verifies if the configured courtesy pause of 2 s was respected. If it was not, the CP sleeps for the remaining time, yielding its execution. The *CourtesyPause* series represents the time elapsed since the CP decided to sleep until it was executed again. The results show that with one CP running on each machine, most of the time is spent on Download operations and executing the CourtesyPause. The percentage of time spent in the storage operations was 1.2%. However, the load of the storage operations increased with the number of CPs. For 30 CPs, the storage operations spent 64.8% of the execution time. Table V presents an analysis of the disk accesses. It shows that the disk throughput could not cope with the load imposed by the CPs. The size of the queue of the requests waiting to be served (third column) grew as the number of requests issued to the device increased (second column). Surprisingly, we observed that the time spent in the CourtesyPause increased to 40.8% for 20 CPs and

DOI: 10.1002/spe

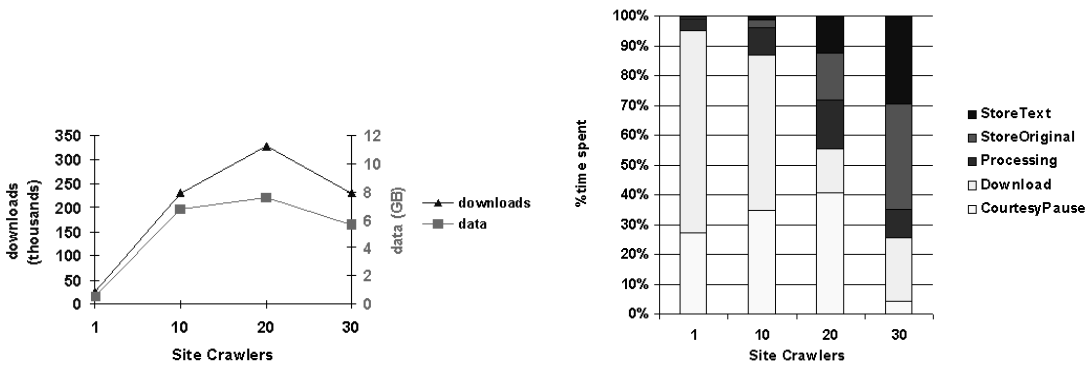

Figure 6. Bottleneck analysis: (a) data downloaded versus the number of CPs per host; (b) duration of the operations.

Table V. Disk I/O analysis of the Volume using iostat. The name of the column on the iostat report is presented in parenthesis.

Number of CPs	Number of WR requests per second (w s^{-1})	Average queue length of the requests (avgqu-sz)	Average time (ms) for requests to be served (await)	Number of sectors written per second (w s^{-1})
1	0.5	0.2	36.2	17.2
10	22.6	10.9	410.4	409.4
20	39.7	49.7	811.2	667.7
30	48.0	92.9	1299.8	821.3

then dropped to 4.6% for 30 CPs. The reason for this phenomenon is that when a machine hosts just one CP, the operating system executes the CP immediately after the CourtesyPause is reached. However, the time that a CP waited to be executed after performing the CourtesyPause increased with the load on the machine, because there were other processes scheduled to run before it. For 30 CPs, the average time for requests to be served was 1.2998 s, a value close to the configured courtesy pause of 2 s. Hence, the time elapsed between two requests resulting from disk latency to the same site frequently achieved 2 s and the CPs did not need to execute courtesy pauses.

The *Processing* operations use mainly the CPU and access data kept on memory without requiring disk accesses. Hence, they were not significantly affected by the load imposed on the machines. The obtained results show that disk access became a bottleneck in the crawler before bandwidth was exhausted. We conclude that the design of a crawler to be deployed on cheap hardware requires the additional concern of optimizing disk accesses. We believe that the storage throughput could be

DOI: 10.1002/spe

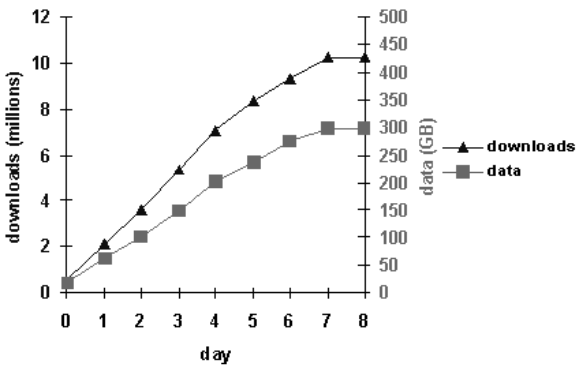

Figure 7. Evolution of a crawl of the Portuguese Web.

improved by using an alternative implementation of the Volumes that would cache larger blocks of data in memory and write them sequentially to disk. However, this approach would require the usage of additional memory.

5.2. Robustness

The robustness of a crawler is measured by the number of pages downloaded over a large period of time. A crawler may achieve an outstanding download rate in one day and may not crawl a single content on the next day because of a crash due to some unexpected problem. We set VN to execute a complete crawl of the Portuguese Web to evaluate its robustness. The crawl was bootstrapped with 152 000 seeds generated from a previous crawl. We used 140 CPs hosted across seven machines. Figure 7 represents the evolution of the crawl. VN collected a total of 10.2 million documents totalling 299.3 GB in eight days. During the crawl several problems occurred on the machines such as operating system crashes that required reboots or disks that ran out of space. However, the crawl never stopped. It crawled on average 1.140 million documents per day (37.2 GB) and the peak download rate achieved was 1.7 million documents within one day (53.2 GB). Figure 7 shows that the download rate started to decrease on day 7. The reason for this fact is that VN was configured to first crawl the sites referenced by the seeds and then begin the *expansion phase*, where it harvests new sites found through URL extraction to expand the boundaries of the Portuguese Web. The number of pages matching the selection criteria falls sharply once the crawler enters the expansion phase because most of the pages visited outside the .PT domain were not written in the Portuguese language (79%).

VN presented a steady download rate before the expansion phase, which shows that it is robust to hazardous situations on the Web as well as operational problems that occurred on the underlying system setup.

DOI: 10.1002/spe

Table VI. Crawling thresholds.

Parameter	Limit	Percentage sites	Percentage URLs
Number of duplicates	10 duplicates	1.6	—
Number of URLs	5000 URLs	0.8	—
URL depth	5	11.3	—
Download time	60 s	—	0.4
Size	2 MB	—	0.1
URL length	200 characters	—	1

5.3. Tuning thresholds

A portion of the Web presents specific characteristics and a crawler should be tuned to improve its performance according to them. On the previous experiments VN was configured with limit values for several parameters to avoid hazardous situations. Periodically, these thresholds must be reviewed and updated to reflect the evolution of the Web. However, the decision to update requires human reasoning. If the limits were achieved as a result of hazardous situations they should not be updated.

Table VI describes the limits imposed and the percentage of URLs or sites whose crawl was stopped by reaching one of the limits. The maximum number of duplicates was overcome by 1.6% of the sites. We observed 10 of these sites and they all contained spider traps. The number of sites that contained more than 5000 URLs was just 0.8%, which confirms previous findings [43]. The maximum depth was achieved in 11.3% of the sites. We visited a sample of these sites and concluded that the limit depth was not related to any hazardous situation. Only 0.4% of the URLs took longer than 60 s to be downloaded and processed to extract meta-data. The results obtained by Liu [49] suggested that the timeout of a Web client should not be longer than 10 s. Hence, we believe that the performance of the crawler could be improved by reducing the timeout value. Only 0.1% of the contents achieved the maximum size of 2 MB and just 1% of the URLs were longer than 200 characters.

In summary, we concluded that the limits for most of the parameters were adequate, except for the maximum URL depth, which should be increased on subsequent crawls of the Portuguese Web.

6. RELATED WORK

In this section we present a performance comparison based on the results published in related work. Table VII summarizes these results. This comparison must be taken with a grain of salt because the experiments were run using different setups over the last decade. Older results require different interpretation to more recent results because the Web is permanently evolving. For instance, the Googlebot harvested 34 pages per second in 1998 and VN harvested 25 pages per second in 2005. However, the size of Web pages has grown and 34 pages in 1998 corresponded to 200 KB of data, while 25 documents in 2005 corresponded to 768 KB. The usage of simulations that can be reproduced for different crawlers is too restrictive, because they cannot reproduce the upcoming hazardous situations

DOI: 10.1002/spe

Table VII. Performance comparison between crawlers.

Crawler name	Number of machines	Internet (Mbps)	Number of downloads per second	Downloaded data (KB s^{-1})	Percentage duplicate download	Downloads /URLs	Simulation results
Googlebot [2]	5	?	34	200	?	31%	No
Kspider [50]	1	8	15	360	?	92%	No
Mercator [3]	4	100	112	1682	8.5%	87%	No
Polybot [24]	4	45	77	1030	13%	85%	No
Ubicrawler [25]	5	116	?	?	?	?	Yes
VN	9	34	25	738	7%	80%	No
WebBase [51]	1	1000	6000	111 000	?	?	Yes
Webgather [52]	4	?	7	?	?	?	No

that degrade the crawler's performance when used on the Web. Simulations cannot realistically test the robustness of the crawlers or if their actions are incommodious to Web servers. The download rate of a crawler while harvesting the real Web tends do be significantly lower than that obtained on simulations. The developers of the Googlebot estimated that it could execute 100 downloads per second (600 KB s^{-1}) but in practice it did not surpass 34 downloads per second. The performance of the Kspider was initially measured by crawling a fixed set of 400 000 URLs and the obtained results suggested that the crawler could download 618 pages per second (6 MB s^{-1}) using four machines. However, the results obtained from a crawl of 8 million pages from the Thai Web suggest that the download rate would have been just 232 downloads per second using four machines. The WebBase crawler achieved an outstanding performance of 6000 downloads per second. However, the harvested documents were not written to disk and the paper suggests that the pages were harvested from sites accessible through a 1 Gbps LAN.

Nonetheless, we compared VN's performance while crawling the Portuguese Web with the results presented in previous studies. Its performance is close to that presented by other crawlers but it was hosted on a larger number of machines (nine). However, VN had the additional overhead of extracting and storing meta-data during the crawl. The speed of the connection to the Internet must be considered to analyze crawling performance. The third, fourth and fifth columns of Table VII show that the most performant crawlers used the fastest connections to the Internet. This might also be a reason why VN presented a lower download rate than the most performant crawlers.

The performance of a crawler is usually synonymous to its download rate, but there are other features that should be considered. In the sixth column of Table VII, we present the percentage of duplicates harvested by the crawlers. The results show that our efforts to minimize the download of duplicates, saving on bandwidth and storage space, yielded good results in practice. VN crawled the smallest percentage of duplicates (23% of the contents were duplicates in its first release). A crawler should also minimize the number of visits to URLs that do not reference a downloadable content. The downloads/URLs column of Table VII presents the ratio between the number of downloads and the URLs visited. VN was configured as a focused crawler of the Portuguese Web and discarded contents considered irrelevant. However, the ratio of *downloads/URLs* is close to that achieved by the remaining crawlers, which did not discard any contents.

DOI: 10.1002/spe

7. CONCLUSIONS

In this paper we shared our experience obtained during the design and operation of the VN crawler. VN has been used in a production environment during the past four years to feed a search engine and populate an archive of the Portuguese Web, having crawled over 54 million documents (1.5 TB). During this period of time we received just two complaints from our crawling activities, which shows that our politeness measures were successful. The results obtained also showed that our crawler is robust and scalable. An extended version of this paper can be found in a technical report [53].

We have shown that there are important crawler design choices that must be made according to the characteristics of the Web. We proposed solutions to mitigate the effects of situations that are hazardous to crawling. These hazardous situations were presented in the context of crawling, but they affect HTTP clients in general. Therefore, the presented solutions can be used to enhance other systems that process Web data, such as browsers or proxies.

We observed that respecting the REP is not just a matter of politeness, it also prevents a crawler from falling into spider traps. The obtained results showed that crawlers able to process cookies are less prone to falling in spider traps and save space in the crawler's data structures. Spider traps can also be avoided through crawling constraints based on Web characterizations. However, the characterization of the Web must be part of the crawling procedure to reflect the evolution of the Web in the configuration of the crawler. Nonetheless, there are spider traps that cannot be detected without human intervention.

Sites with different names that simultaneously serve the same content (duphosts) can be detected within a Web data set by intersection of contents of sites or by the analysis of the site names. However, duphosts are transient, making the applicability of this knowledge in avoiding the download of duplicates questionable.

We observed that the standard URL normalization process is insufficient to avoid the download of duplicates and can be improved with additional rules adequate to Web crawling. We proposed several algorithms to avoid the download of duplicates and invalid URLs.

In future work we plan to execute broader crawls of the Web to test VN's performance on larger harvests. The increasing capacity disks at low prices has helped crawlers to extend their storage capacity. However, the storage throughput did not follow the pace of the disk's capacity and latency is a potential bottleneck that must be carefully addressed in the design of crawlers. The storage throughput of the harvested contents is the main performance problem for the current version of VN. We will improve this aspect by studying alternative storage techniques.

ACKNOWLEDGEMENTS

We thank Bruno Martins for his comments and collaboration in the implementation of VN. This study was partially supported by the FCCN-Fundação para a Computação Científica Nacional and FCT-Fundação para a Ciência e Tecnologia, under grant SFRH/BD/11062/2002 (scholarship).

REFERENCES

1. Hawking D, Craswell N. Very large scale retrieval and Web search. *The TREC Book*, Voorhees E, Harman D (eds.). MIT Press: Cambridge, MA, 2004.

DOI: 10.1002/spe

2. Brin S, Page L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* 1998; **30**(1–7):107–117.
3. Heydon A, Najork M. Mercator: A scalable, extensible Web crawler. *World Wide Web* 1999; **2**(4):219–229.
4. Cho J, Garcia-Molina H, Page L. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems* 1998; **30**(1–7):161–172.
5. Handschuh S, Staab S, Volz R. On deep annotation. *Proceedings of the 12th International Conference on World Wide Web*. ACM Press: New York, 2003; 431–438.
6. Cho J, Roy S. Impact of search engines on page popularity. *Proceedings of the 13th International Conference on World Wide Web*. ACM Press: New York, 2004; 20–29.
7. Olsen S. Does search engine's power threaten Web's independence? *CNET News.com*, October 2002.
8. Mockapetris PV. Domain names—concepts and facilities. *Request for Comments 1035*, Network Working Group, 1987.
9. ICANN. ICANN—Verisign's wildcard service deployment. http://www.icann.org/topics/wildcard-history.html [November 2004].
10. Hallam-Baker PM, Connolly D. Session identification URI. http://www.w3.org/TR/WD-session-id.html [November 2005].
11. Douglis F, Feldmann A, Krishnamurthy B, Mogul JC. Rate of change and other metrics: A live study of the World Wide Web. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, 1997. USENIX Association: Berkeley, CA, 1997.
12. Jahn. Spider traps – an upcoming arms race. http://www.jahns-home.de/rentmei/html/sptraps.html [November 2004].
13. Gomes D, Freitas S, Silva MJ. Design and selection criteria for a national Web archive. *Proceedings of the 10th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)* (*Lecture Notes in Computer Science*, vol. 4172), Gonzalo J, Thanos C, Verdejo MF, Carrasco RC (eds.). Springer: Berlin, 2006.
14. Martins B, Silva MJ. The WebCAT framework: Automatic generation of meta-data for Web resources. *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, September 2005.
15. Woodruff A, Aoki PM, Brewer E, Gauthier P. An investigation of documents from the World Wide Web. *Proceedings of the 5th International World Wide Web Conference on Computer Networks and ISDN Systems*. Elsevier Science Publishers B. V.: Amsterdam, The Netherlands, 1996; 963–980.
16. Paulson LD. Building rich Web applications with AJAX. *Computer* 2005; **38**(10):14–17.
17. Thelwall M. A free database of university Web links: Data collection issues. *International Journal of Scientometrics, Informetrics and Bibliometrics* 2002; **6/7**(1).
18. Henzinger MR, Motwani R, Silverstein C. Challenges in Web search engines. *SIGIR Forum* 2002; **36**(2):11–22.
19. Bharat K, Broder AZ, Dean J, Henzinger MR. A comparison of techniques to find mirrored hosts on the WWW. *Journal of the American Society of Information Science* 2000; **51**(12):1114–1122.
20. da Costa Carvalho AL, de Souza Bezerra AJ, de Moura ES, da Silva AS, Peres PS. Detecção de réplicas utilizando conteúdo e estrutura. *Simpósio Brasileiro de Banco de Dados*, Uberlândia, Brazil, 2005. ACM Press: New York, 2005; 25–39.
21. Barr D. *Common DNS Operational and Configuration Errors*, February 1996.
22. Chakrabarti S, van den Berg M, Dom B. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks* 1999; **31**(11–16):1623–1640.
23. Raghavan S, Garcia-Molina H. Crawling the hidden Web. *Proceedings of the 27th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc.: San Francisco, CA, 2001; 129–138.
24. Shkapenyuk V, Suel T. Design and implementation of a high-performance distributed Web crawler. *Proceedings of the 18th International Conference on Data Engineering*. IEEE Computer Society Press: Los Alamitos, CA, 2002; 357.
25. Boldi P, Codenotti B, Santini M, Vigna S. Ubicrawler: A scalable fully distributed Web crawler. *Software—Practice and Experience* 2004; **34**(8):711–726.
26. Habib MA, Abrams M. Analysis of sources of latency in downloading Web pages. *WebNet*, San Antonio, TX, November 2000; 227–232.
27. Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T. *Hypertext Transfer Protocol – HTTP/1.1*, June 1999.
28. The Web Robots Pages. HTML author's guide to the ROBOTS meta-tag. http://www.robotstxt.org/wc/meta-user.html [March 2005].
29. Cho J, Garcia-Molina H. Parallel crawlers. *Proceedings of the 11th International Conference on World Wide Web*. ACM Press: New York, 2002; 124–135.
30. Eiron N, McCurley KS, Tomlin JA. Ranking the Web frontier. *Proceedings of the 13th International Conference on World Wide Web*. ACM Press: New York, 2004; 309–318.
31. Berners-Lee T, Fielding R, Masinter L. *Uniform Resource Identifier (URI): Generic Syntax*, January 2005.
32. Broder AZ, Najork M, Wiener JL. Efficient URL caching for World Wide Web crawling. *Proceedings of the 12th International Conference on World Wide Web*. ACM Press: New York, 2003; 679–689.
33. Cothey V. Web-crawling reliability. *Journal of the American Society for Information Science and Technology* 2004; **55**(14):1228–1238.

DOI: 10.1002/spe

34. Koster M. A standard for robot exclusion. http://www.robotstxt.org/wc/norobots.html [June 1994].
35. Ltd N. Netcraft: Web server survey archives. http://news.netcraft.com/archives/web_server_survey.html [November 2005].
36. Martins B, Silva MJ. Language identification in Web pages. *Proceedings of the 20th Annual ACM Symposium on Applied Computing (ACM-SAC-05)*, Santa Fe, New Mexico, March 2005. ACM Press: New York, 2005.
37. Berk E, Ananian CS. JLex: A lexical analyzer generator for Java(TM). http://www.cs.princeton.edu/∼appel/modern/java/JLex/ [February 2005].
38. Campos J. Versus: A Web repository. *Master Thesis*, Faculdade de ciências, Universidade de Lisboa, 2003.
39. HypersonicSQL. http://sourceforge.net/projects/hsqldb/ [July 2006].
40. Oracle Corporation. Oracle9i. http://www.oracle.com/ip/index.html?content.html [July 2006].
41. Gomes D, Santos AL, Silva MJ. Managing duplicates in a Web archive. *Proceedings of the 21st Annual ACM Symposium on Applied Computing (ACM-SAC-06)*, Dijon, France, April 2006, Liebrock LM (ed.). ACM Press: New York, 2006.
42. Gomes D, Silva MJ. Characterizing a national community Web. *ACM Transactions on Internet Technology* 2005; **5**(3):508–531.
43. Brandman O, Cho J, Garcia-Molina H, Shivakumar N. Crawler-friendly Web servers. *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*, Santa Clara, CA, 2000. ACM Press: New York, 2000.
44. Baeza-Yates R, Castillo C. Crawling the infinite Web: Five levels are enough. *Proceedings of the 3rd Workshop on Web Graphs (WAW)*, Rome, Italy, October 2004 (*Lecture Notes in Computer Science*, vol. 3243). Springer: Berlin, 2004.
45. Cockburn A, McKenzie B. What do Web users do? An empirical analysis of Web use. *International Journal Human-Computer Studies* 2001; **54**(6):903–922.
46. Jaimes A, del Solar JR, Verschae R, Yaksic D, Baeza-Yates R, Davis E, Castillo C. On the image content of the Chilean Web. *Proceedings of the 1st Conference on Latin American Web Congress (LA-WEB '03)*. IEEE Computer Society Press: Los Alamitos, CA, 2003.
47. Marshak M, Levy H. Evaluating Web user perceived latency using server side measurements. *Computer Communications* 2003; **26**(8):872–887.
48. Wills CE, Mikhailov M. Examining the cacheability of user-requested Web resources. *Proceedings of the 4th International Web Caching Workshop*, San Diego, CA, 1999.
49. Liu B. Characterizing Web response time. *Master's Thesis*, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, VI, April 1998.
50. Koht-Arsa K. High performance cluster-based Web spiders. *Master's Thesis*, Graduate School, Kasetsart University, March 2003.
51. Cho J, Garcia-Molina H, Haveliwala T, Lam W, Paepcke A, Raghavan S, Wesley G. Stanford WebBase components and applications. *Technical Report*, Stanford Database Group, July 2004.
52. Yan H, Wang J, Li X, Guo L. Architectural design and evaluation of an efficient Web-crawling system. *The Journal of Systems and Software* 2002; **60**(3):185–193.
53. Gomes D, Silva MJ. The Viúva Negra Web crawler. *Technical Report DI/FCUL TR 06–21*, Department of Informatics, University of Lisbon, November 2006.

DOI: 10.1002/spe