The Viuva Negra crawler

Daniel Gomes
Mário J. Silva

DI–FCUL TR–2006–06-21

November 2006

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

The Viuva Negra crawler

Daniel Gomes, Mário J. Silva
Departamento de Informática
Faculdade de Ciências, Universidade de Lisboa
1749-016 Lisboa, Portugal

{dcg, mjs}@di.fc.ul.pt

November 2006

Abstract

This report discusses architectural aspects of web crawlers and details the design, implementation and evaluation of the Viuva Negra (VN) crawler. VN has been used for 4 years, feeding a search engine and an archive of the Portuguese web. In our experiments it crawled over 2 million documents per day, correspondent to 63 GB of data. We describe hazardous situations to crawling found on the web and the adopted solutions to mitigate their effects. The gathered information was integrated in a web warehouse that provides support for its automatic processing by text mining applications.

1 Introduction

Web mining systems are crucial to harness 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 [41], but they become quickly stale 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 due to the competitive nature of this business [12]. The permanent evolution of the web and the upcoming of new usage contexts demands continuous research in crawling systems. Brewster Kahle, the founder of the Internet Archive, revealed that their commercial crawler is rewritten every 12–18 months to reflect changes in the structure of the web [46]. 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 harvesting the web. The description of hazardous situations to crawling is scarce among scientific literature, because most experiments are based on simulations or short term crawls that do not enable their identification. Hazardous situations are commonly ignored in academic studies because the scientific hypotheses being tested assume a much simpler model of the web 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 discuss the design of crawling systems and detail the architecture, implementation and evaluation of the Viuva Negra (VN) web crawler. VN was developed and tested during the past 4 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 improve it. The main of this study contributions are:

- An analysis of techniques to partition the URL space among the processes of a parallel crawler;
- A flexible and robust crawler architecture that copes with distinct usage contexts and it is able to follow the pace of the evolution of the web;
- A detailed description of hazardous situations 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. So, the presented solutions can be used to enhanced other systems that process web data, such as browsers or proxies;
- 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 present the requirements for a crawler. In Section 3 we discuss the architectural options for designing a crawler. The following Section discusses situations on the web that are hazardous to crawling. Section 5 presents the architecture of the VN crawler, its main features and implementation. In Section 6, we present evaluation results of the VN crawler and share the lessons learned while harvesting the Portuguese web. In Section 7, we present related work and compare our system with other crawlers. Finally, in Section 8, we draw our conclusions and suggest future work.

2 Requirements

Crawlers can be classified in four major classes according to their harvesting strategies: i) *Broad crawlers* collect the largest amount of information possible within a limited time interval [55]; ii) *Incremental crawlers* revisit previously fetched pages looking for changes [27]; iii) *Focused crawlers* harvest information relevant to a specific topic from the web, usually with the help of a classification algorithm, to filter irrelevant documents [16]; and iv) *Deep* or *hidden web crawlers* also harvest information relevant to a specific topic but, unlike focused crawlers, have the capacity of filling forms in web pages and collect the returned pages [58, 64]. Although each type of crawler has specific requirements, they all share ethical principles and address common problems. A crawler must be:

Polite. A crawler should not overload web servers. Ideally, the load imposed while crawling should be equivalent to that of a human while browsing. A crawler should expose the purposes of its actions and not impersonate a browser, so that webmasters can track and report inconvenient actions. A crawler must respect exclusion mechanisms and avoid visits to sites where it is not welcome. The Robots Exclusion Protocol (REP) makes the definition of access rules on a file named robots.txt that is automatically interpreted by crawlers [48]. An author of an individual page can also indicate if it should be indexed and if the links should be followed by a crawler through the `ROBOTS HTML` meta-tag [67];

Robust. The publication of information on the web is uncontrolled. A crawler must be robust against hazardous situations that may affect its performance or cause its mal-functioning;

Fault tolerant. Even a small portion of the web is composed by a large number of contents, which may take several days to be harvested. Crawlers frequently present a distributed architecture comprising multiple components hosted on different machines. A crawler must be fault tolerant so that its performance may degrade gracefully if one of its components fails, without compromising the progress of the crawl on the remaining machines;

Integrable. The information harvested by a crawler would have little use if it could not be processed by other applications. Thus, a crawler should be designed to operate as a component of broader systems;

Able to collect meta-data. A *content* results from a download (e.g. an HTML file). *Meta-data* is information that describes a content (e.g. its size). There is meta-data temporarily available only during the crawl (e.g. date of crawl). A crawler should keep these meta-data because it is often needed in the future. For instance, the Content-Type HTTP header field identifies the media type of a content. If this meta-data element is lost, the content type must be guessed latter;

Configurable. A crawler should be highly configurable to suit the requirements of different applications without suffering major changes;

Scalable. The crawl of a portion of the web must be completed within a limited time and the download rate of a crawler must be adequate to the requirements of the application that will process the harvested data. A search engine that requires weekly updates to its index can not use a crawler that takes months to harvest the data from the web. The download rate of the crawler is always limited by the underlying resources, such as the number of machines. However, a crawler must be designed to scale its performance proportionally to available resources;

Flexible. The web is permanently evolving and it is difficult to predict which direction it will take in the future. A crawler must be flexible to enable quick adaption to new publishing technologies and formats used on the web as they become available;

Economic. A crawler should be parsimonious with the use of external resources, such as bandwidth, because they are outside of its control. A crawler may connect the Internet through a large bandwidth link but many of the visited web servers do not;

Manageable. A crawler must include management tools that enable the quick detection of its faults or failures. On the other hand, the actions of a crawler may be deemed unacceptable to some webmasters. So, it is important to keep track of the actions executed by the crawler for latter identification and correction of undesirable behaviors.

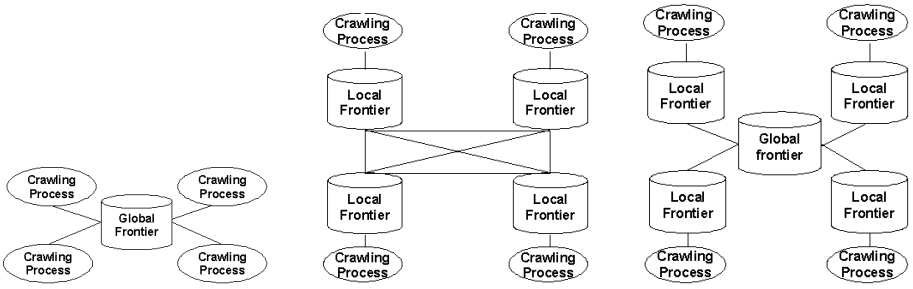
3 Designing a crawler

Designing a crawler to harvest a small set of well-defined URLs is simple. However, harvesting information spread across millions of pages requires adequate selection criteria and system architectures. Most crawlers adopt distributed architectures that enable parallel crawling to cope with the large size of the web.

In this Section we analyze sources of URLs to bootstrap a new crawl. Then, we present architectural options to design a crawler and discuss strategies to divide the URL space to harvest among several crawling processes. Finally, we present heuristics to avoid the crawl of invalid URLs and duplicates.

3.1 Bootstrapping

The selection criteria delimits the boundaries of the *portion* of the web to crawl and it is defined according to the requirements of the application that will process the harvested data. For instance, a search engine is interested on the portion of the web composed by the documents that can be indexed. Hence, its crawler selects textual contents from the web. A crawl of the web is bootstrapped with a list of URLs, called the *seeds*, which are the access nodes to the portion of the web to crawl. For instance, to crawl a portion of the web containing all the documents hosted in



a) Centralized. b) Distributed. c) Hybrid.

Figure 1: Types of Frontiers.

the .gov domain, we should use URLs from that domain as seeds. These should be carefully chosen to prevent the crawler from wasting resources visiting URLs that do not reference accessible or relevant contents. The seeds can be gathered from different sources:

User submissions. The seeds are posted by the users of a given service. This approach is common among search engines that invite users to submit the home page of their sites to be indexed. However, URLs containing typographical errors or referencing sites under construction are common;

Previous crawls. The seeds are extracted from a previous crawl. The main problem of this source of seeds is that URLs have short lives and an old crawl could supply many invalid seeds. For instance, Ntoulas et al. witnessed that only 20% of the URLs that they crawled one year before were still valid [57];

Domain Name System listings. The seeds are generated from domain names. However, the domains reference servers on the Internet and not all are web servers. So, the generated seeds may not be valid. Another problem is that the lists of the second-level domains belonging to the web portion to be crawled are usually not publicly available.

3.2 Architectural options

A crawler is composed by a *Frontier* that manages the URLs and the *Crawling Processes* that iteratively harvest documents from the web and store them locally. A Crawling Process (CP) iteratively gets a seed from the Frontier, downloads the document, parses it, extracts the linked URLs and inserts them in the Frontier to be harvested. A crawl finishes when there are no seeds left to visit or a limit date is reached. A simple crawler can be assembled from only one CP and one Frontier. This is what is known as an off-line browser. Simple crawlers like this are suitable for storing local copies of web 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 increasing its 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 Crawling Process 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 [43]. There are 3 approaches for organizing the Frontier (see Figure 1):

Centralized. In this organization, the Crawling Processes share a single *Global Frontier*. The URL-seen test is permanently being executed on the Global Frontier by all the CPs. This

Partition. function	DNS caching	use keep-alive connections	avoid server overloading	reuse site meta-data	independency
IP	++	++	++	+	–
Site	+	+	+	++	++
Page	–	–	–	–	++

Table 1: Comparison of the partitioning schemes.

architecture is conceptually simple but the Frontier becomes a potential hot-spot of the system;

Distributed. The Frontier is a cooperative set of *Local Frontiers*. There is not a central point of congestion because the URL-seen are distributed by the Local Frontiers. However, the URL-seen test imposes frequent synchronization among the Local Frontiers, which may become a bottleneck;

Hybrid. 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. However, the design of the crawler is more complex than with previous approaches.

3.3 Web partitioning and assignment

The URL space of the web may be partitioned for parallel crawling. A partitioning function maps an URL to its partition. The partitioning strategy has implications on the operation of the crawler. The main objective of partitioning the URL space is to distribute the workload among the CPs creating groups of URLs that can be harvested independently. After partitioning, each Crawling Process is responsible for harvesting exclusively one partition at a time. We assume that the Crawling Processes do not keep state between the processing of partitions and do not communicate directly with each other. In general, the following partitioning strategies may be followed:

IP partitioning. Each partition contains the URLs hosted on a given IP address. We assume that each web server has a single IP address;

Site partitioning. Each partition 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 and each will be crawled separately (virtual hosts);

Page partitioning Each partition 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 to harvest a selected set of pages spread on the web.

The number of the URLs contained in a partition should be ideally constant to facilitate load balancing. A CP may exhaust its resources while trying to harvest a partition containing an abnormally large number of URLs.

The page partitioning is the most adequate according to this criterion. The IP partitioning tends to create some extremely large partitions due to servers that host thousands of sites, such as Geocities (`www.geocities.com`) or Blogger (`www.blogger.com`). The site partitioning is more likely to create partitions containing a single URL, due to sites under construction or presenting an error message. The efficiency of the IP and site partitioning depends 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 1 summarizes the relative merits of each strategy, which are characterized by the following determinants:

DNS caching. A Crawling Process 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 a content [37]. 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 because all the URLs are hosted on the IP address that identifies the partition. A site partition 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 [37]. However, HTTP keep-alive connections enable the download of several documents reusing the same TCP connection to a server [30]. A page partition 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 site partitioning, 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. Page partitioning is not suitable to guarantee courtesy pauses, because the URLs of a server are spread among several partitions. Thus, if no further synchronization mechanism is available, the Crawling Processes may crawl the same server simultaneously, disrespecting the courtesy pause. The page partitioning requires that each Crawling Process keeps track of the requests executed by the other ones to respect the courtesy pause. With IP partitioning, it is easier to respect the courtesy pause because each Crawling Process 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 site partitioning 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 Crawling Processes 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 Crawling Processes;

Reuse of site meta-data. Sites contain meta-data, such as the Robots Exclusion file, that influences crawling. The page partitioning strategy is not suitable to reuse the site's meta-data because the URLs of a site are spread across several partitions. With the IP partitioning, the site's meta-data 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 partitions that contain thousands of different sites generated through virtual hosting. On its turn, when a crawler is harvesting a site partition, 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 enable the assignment of an URL to a partition independently from external resources. The IP partitioning depends on the DNS servers to retrieve the IP address of an URL and it can not 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 Crawling Processes.

Initially each partition contains only a set of seeds. A partition is assigned to a CP that becomes responsible for harvesting the correspondent URLs. The assignment process can be *static* or *dynamic* [19]. In the static assignment, the partitions are assigned before the beginning of the crawl. Each CP knows its partition and assigns the URLs extracted from web pages to the partitions responsible for their crawl. The static assignment imposes that the number of CPs is constant during the crawl to guarantee that all the partitions are harvested. The partitions assigned to a CP would not be harvested if it failed and could not be recovered. Moreover, one can not 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 partitions 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. There are two strategies for dynamic assignment according to the behavior of the coordinator:

Push. The coordinator sends partitions to the Crawling Processes. It must keep an accurate state of the system to balance the load efficiently. The coordinator is responsible for monitoring the set of active Crawling Processes and contact them to assign partitions. So, it has the overhead of establishing connections, detecting and managing possible failures of the Crawling Processes. This approach allows to concentrate the management of the crawler on a single component which facilitates administration tasks;

Pull. The coordinator waits for requests of partitions to crawl by CPs. It does not have to permanently monitor the system because the Crawling Processes demand work on a need basis. The number of Crawling Processes may be variable without imposing any overhead on the coordinator because it simply responds to requests for uncrawled partitions.

3.4 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 can not be downloaded. A crawler will waste resources trying to crawl these URLs, so their presence in the Frontier should be minimized. Invalid URLs can be pruned using the following strategies:

Discarding malformed URLs. A malformed URL is syntactically incorrect [6]. Malformed URLs are most likely caused by a typing errors. For instance, an URL containing white spaces is syntactically incorrect. However, there are web servers that enable the usage of malformed URLs;

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 imposes an additional overhead on the DNS servers.

Duplicates occur when two or more different URLs reference the same content. A crawler should avoid harvesting duplicates to save on processing, bandwidth and storage space. The crawling of duplicates can be avoided through the normalization of URLs:

1. *Case normalization*: the hexadecimal digits within a percent-encoding triplet (e.g., "%3a" versus "%3A") are case-insensitive and therefore should be normalized to use uppercase letters for the digits A-F;

2. *Percent-Encoding Normalization*: decode any percent-encoded octet that corresponds to an unreserved character;

3. *Convert site name to lower case*: the domain names are case insensitive thus, the URLs `www.site.com/` and `WWW.SITE.COM/` reference the same content;

4. *Convert relative to absolute file paths*: For instance, `www.site.com/dir/../index.html` to `www.site.com/index.html`;

5. *Remove identification of the HTTP default port 80*: For instance, change `www.site.com:80/index.html` to `www.site.com/index.html`;

6. *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 [30]. Hence, the transformation must be done by the client before sending a request. This rule of normalization prevents that URLs, such as `www.site.com` and `www.site.com/`, originate duplicates;

7. *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;

8. *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 4.3.1);

9. *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 like `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. A crawler 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 an 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. The number of URLs inserted in the Frontier increases, approximately by 5% [15, 34, 43], but duplicates are avoided.

4 Hazardous situations

The web is very heterogeneous and there are hazardous situations to web crawling. Some of them are malicious, while others are caused by mal functioning 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 to design robust crawlers capable of coping with them.

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. Then, we 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.

4.1 Spider traps

Heydon and Najork defined a spider trap as *an URL or set of URLs that cause a crawler to crawl indefinitely* [43]. We relaxed this definition and consider that situations that significantly degrade the performance of the 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 [22]. 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 [40]. Thus, preventing a crawler from visiting dynamic pages to avoid spider traps would exclude a large parcel of the web. Some webmasters create traps to boost the placement of their sites in search engine results [43], while others use traps to repel crawlers because they spend the resources of the web servers. Spider traps bring disadvantages to their creators. A trap compromises the navigability within the site and human users get frustrated if they try to browse a spider trap. Plus, search engines have a key role in the promotion of web sites, and they ban sites containing traps from their indexes [23, 59]. Next, we present some examples of spider traps and discuss how to mitigate their effects:

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 [54]. 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 [56]. The usage of DNS wildcards is hazardous to crawlers because they enable the generation of an infinite number of sites 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 would 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.

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 tops a number of duplicates. A malfunctioning site may start serving contents with a higher latency. This situation would cause that a crawler would take too much time to harvest its contents, delaying the overall progress of the crawl. To prevent this, a crawler should impose a limit on the time to harvest each URL.

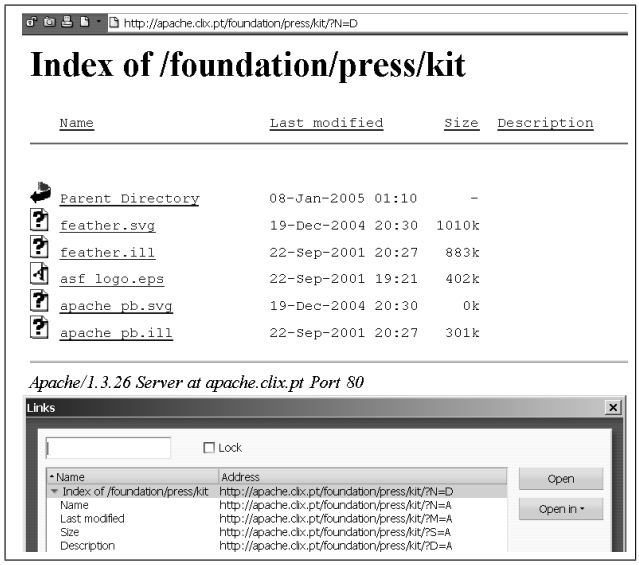


Figure 2: Apache directory list page and the linked URLs.

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.

Session identifiers and cookies. HTTP is a stateless protocol that does not allow tracking of user reading patterns by itself. However, this is often required by site developers, for instance, to build profiles of typical users. According to web rules, session identifier embedded in the URLs linked from pages allows maintaining state about a sequence of requests from the same user. A session identifier should follow a specific syntax beginning with the string "SID:" [39]. In practice, the session identifiers are embedded by developers in URLs as any other parameters. Session identifiers have lifetimes to prevent that different users are identified as the same one. A session identifier replacement causes 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 get trapped 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 previously crawled [26]. A crawler may avoid getting trapped by stop following links within the site when a limit number of duplicates is achieved. 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 achieved.

Cookies have been replacing session identifiers embedded in URLs [4]. 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. A crawler able to process cookies is less prone to fall in traps caused by session identifiers embedded in URLs.

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 2 presents a directory list and its embedded links. The directory list contains 4 links to pages that present it reordered by *Name*, *Last-Modified date*, *Size* and *Description*, in ascendent or

descendent order. Thus, if a crawler follows all the links embedded in a directory list page, it will harvest the same information referenced by 8 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 get trapped if harvesting, for instance, temporary files periodically generated in a directory. A crawler could exclude URLs that reference directory listings to avoid traps but they are frequently used to publish valuable contents, such as open-source software and documentation.

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/...`) [69]. 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 an user on the links of their pages. The idea was that when users reach a given page 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 gets trapped following the generated links. Hence, a crawler should impose a limit on the length of the URLs harvested.

4.2 Difficult interpretation contents

Crawlers interpret the harvested contents to extract valuable data. If a crawler can not 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:

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 software. However, sometimes the Content-Type values do not correspond to the real media type of the content [32] and a HTTP client may not be able to interpret it correctly. An erroneous Content-Type response can be detected through the analysis of the extracted data. 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 white spaces between sequences of characters, the content may have been incorrectly interpreted. 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 1 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). However, this approach requires specific heuristics to each of the many media types available on the web and identifying the media type of a binary file is a complex task;

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 [53, 71];

Cloaking. A cloaking web server provides different contents 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 XML 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;

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 AJAX (Asynchronous JavaScript And XML) libraries contributed to the widespread usage of this language in web pages [62]. 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 [68].

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 and crawlers usually identify the embedded URLs using pattern matching. For instance, they identify an URL embedded in a JavaScript program if it begins with the string "http://".

4.3 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 [42]. 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, to make profit on their resale. The requests to these domains are redirected to a site that presents a sale proposal. To protect against squatters, companies also register multiple domain names related to their trade marks and point them to the company's site;

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

Heuristic	Invalid IP	% of duphosts	Precision	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

Table 2: Results from the 5 approaches to detect duphosts.

and storage space. Previous works presented algorithms to detect duphosts within a set of documents harvested from the web [7, 24]. 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. Sites identified as duphosts may have meanwhile 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 3 heuristics to detect if two sites were duphosts:

SameHome. Both sites present equal home pages. The home page describes the content of a site. So, 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;

SameHomeAnd1Doc. Both sites present equal home pages and at least one other equal content. This approach follows the same intuition than the 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 contents 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 contents are 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 5 lists of duphosts following the heuristics SameHome, SameHomeAnd1Doc, and DupsP considering levels of duplication of 60%, 80% and 100%. We evaluated the proposed heuristics by simulating their application on a crawl executed 98 days after the creation of the data set. Table 2 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 (column *Invalid IP*). The 3^{rd} column of Table 2 presents percentage of the total number of sites in the data set that were replicas identified through each heuristic after the IP check. On average, 1,841 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

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

Table 3: Consequences of the normalization of the usage of the WWW prefix in site names.

of pairs detected (Dups100) as 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 1 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 2^{nd} and 3^{rd} lines of Table 2.

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 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. Due 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.

4.3.1 The WWW prefix

The most common reason for duphosts is the existence of site names that just differ on the prefix "www.". 51% of the names of the duphosts detected on the previous experiments differed just on the prefix "www.". It is recommended that World Wide Web site names begin with the prefix "www." [3]. So, 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 3 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 due 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.

5 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. Hence, it must be highly configurable, fulfilling the requirements of different applications. The hardware is permanently evolving and the number of machines required to perform crawls of the web is considerable. VN was designed to

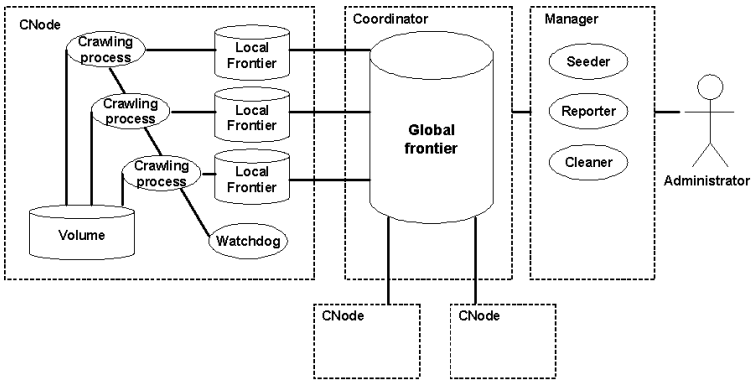


Figure 3: Architecture overview.

be easily deployed on an heterogenous cluster of inexpensive machines and enable the scalability of its download rate with the addition of new machines to the cluster. The hazardous situations that occur on the web frequently cause mal-functions on crawlers. Plus, system crashes among a cluster of machines that use cheap hardware are frequent. Hence, VN must be fault tolerant.

VN has a hybrid Frontier, uses site partitioning and dynamic-pull assignment:

Hybrid frontier. Each Crawling Process 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. A Crawling Process begins the crawl of a new partition by transferring a seed from the Global to its Local Frontier. We say that the partition was *checked-out*. Then, the URLs that match the partition are harvested by the Crawling Process. When the crawl of the partition is finished, the correspondent meta-data is transferred to the Global Frontier (*check-in*). A Crawling Process successively checks-out a seed, harvests the partition and checks-in the resultant meta-data, until there are no unvisited seeds in the Global Frontier;

Site partitioning. Besides the advantages previously discussed (Section 3.3), three additional reasons lead us to adopt the site partitioning strategy. First, a Crawling Process frequently accesses the Local Frontier to execute the URL-seen test. As sites are typically small [28], 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 designed 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 partitions to CPs as they pull them. The Global Frontier guarantees that a partition is never harvested simultaneously by two CPs. The Global Frontier identifies each partition with the site's hash and manages 3 lists: i) partitions to crawl; ii) partitions being crawled and; iii) partitions crawled. When a Crawling Process checks-out a partition, it is moved from the first to the second list. The checks-in moves the partition from the second to the third list.

Figure 3 describes VN's architecture. It is composed by 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 *Crawling Processes*, *Local Frontiers*, *Volumes* that store the harvested data and a *Watchdog* that restarts the CPs if they are considered dead (inactive

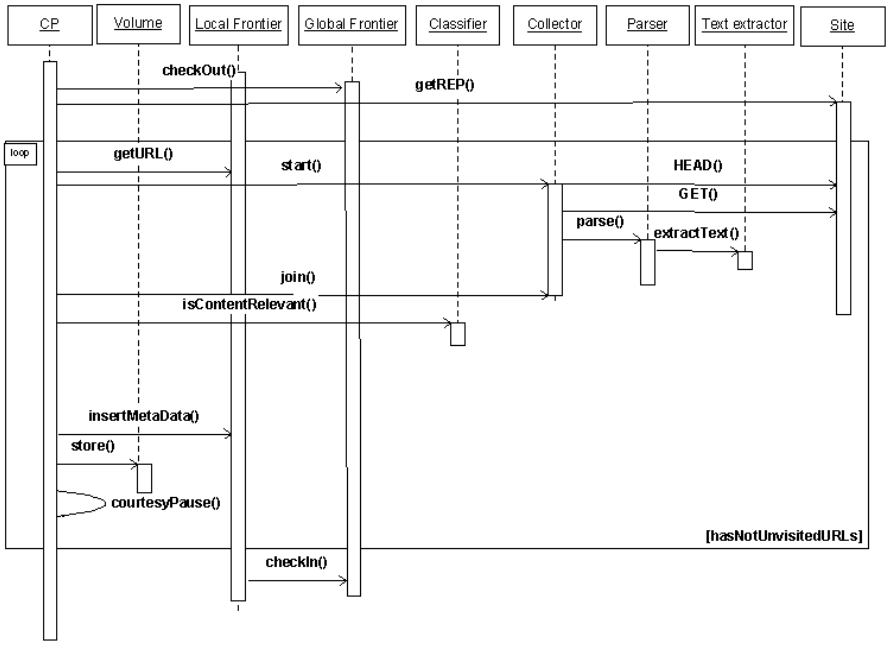


Figure 4: Sequence diagram: crawling a site.

for a given period of time). The scheduling of the execution of the Crawling Processes 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* gets statistics on the state of the system and emails them to a human *Administrator*. The Cleaner allows to release resources acquired by faulty CPs.

5.1 Crawling algorithm

The Crawling Processes harvest information from the web visiting one site at a time in a breadth-first mode. Figure 4 shows this process. The crawl of a new site begins when the CP *checks-out* a seed. The Crawling Process 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 on memory avoids redundant processing by the applications 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. When the CP finishes visiting the site, it checks-in the partition.

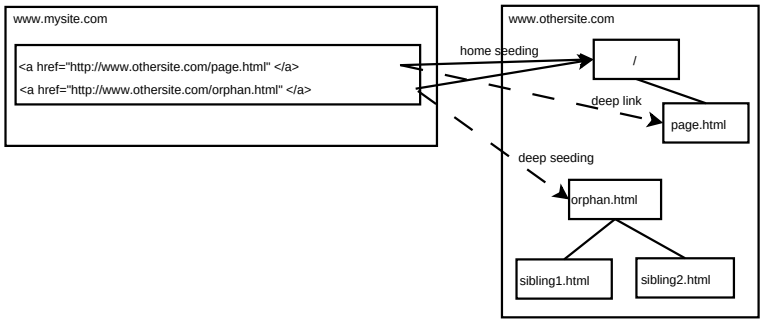


Figure 5: Deep vs. home page policy.

5.2 Fault management

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

A Crawling Process 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 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 remaining. However, they depend on the Global Frontier for synchronization. A Crawling Process 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 tempted. 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 Crawling Process 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 and delete the orphan contents 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 harvested on the first run.

The contents can be stored locally on the same machine that hosts the Crawling Process 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. In the current VN operating environment the storage of contents in remote Volumes is 52% slower than on the local ones (connected through a 100 Mbps Ethernet).

5.3 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 [13]. 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 Crawling Process 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 meanwhile by another CP. 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 3 policies for the insertion of external URLs in the Frontier:

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 Crawling Process replaces every external URL by its site home page before inserting it in the Local Frontier (Figure 5). The home page policy reduces the number of external URLs to check-in. However, if a Crawling Process can not 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 different than the home page. The deep link policy assumes that there are pages not accessible through a link path from the home page of the site. The Crawling Process inserts the external URLs without any change in the Local Frontier to maximize the coverage of the crawl. For instance, in Figure 5 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. Follows deep links but always visits the home page of the sites. This policy is intended to maximize coverage. Even if a deep link references a content without links, the remaining site accessible through a link path from the home page will be harvested.

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. 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 [28] 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, such as to execute exhaustive crawls of corporate intranets.

5.4 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 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 [66]. 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 [48]. Thus, VN tolerates common syntax errors in the robots.txt file, to respect the restrictions imposed by the webmasters. When the robots.txt file can not be interpreted, the site is crawled without restrictions. The indications provided by the meta-tag ROBOTS contained in pages are also respected [67]. 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. 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 [43]. If VN detects that the site being crawled contains a trap, it stops following the links within the site. The contents harvested until the trap was detected are kept, because they might have valuable information. A spider trap is detected if the site exceeds configurable thresholds for the number of URLs, duplicates or level of depth. 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 malfunctioning on a site may cause that the contents start being served with a high latency. This situation would cause 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 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 contained at least 1 whitespace from the extracted texts to prevent classification errors but tolerates common format errors in HTML documents to extract meta-data, such as unmatched tags [53].

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 [50]. 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. 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.

5.5 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 Crawling Processes consist of approximately 5,000 lines of Java code. They use hash tables to keep the list of duphosts and the DNS cache. The Parser was

Configuration	#machines	CPU (GHz)	Mem. (GB)	Disk speed (r.p.m)	Storage (GB)
1	1	2 x P4-2.4	4	SCSI 10,000	5 x 73
2	4	1 x P4-2.4	1.5	IDE 7,200	2 x 180
3	2	2 x P4-2.4	2	IDE 7,200	5 x 250
4	1	2 x P3-1.26	1	SCSI 15,000	2 x 18
5	1	2 x P3-1.26	4	SCSI 10,000	5 x 73

Table 4: Hardware used to implement VN.

based on WebCAT, a Java package for extracting and mining meta-data from web documents [53]. The Classifier used to harvest the Portuguese web includes a language identifier [52]. The Robots Exclusion file interpreter was generated using Jlex [5]. 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 [14]. The information crawled 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 [44] and the Global Frontier is based on Oracle 9i [60]. 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 [33].

6 Evaluation

In this section we analyze 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. We performed experiments to detect bottlenecks, mal-functions and tune its configuration according to the characteristics of the harvested portion of the web.

Table 4 summarizes the hardware configurations of the 9 machines used to support VN in our experiments. For instance, a machine with configuration *1* had 2 Pentium 4 processors running at *2.4* GHz, *4* GB of memory and *5* SCSI disks running at *10,000* rotations per minute, each one with *73* GB of storage space. All the machines use 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 inter-connected 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 [34]. 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 less documents [11, 34]. We imposed a limit to URL depth of 5, because Baeza-Yates and Castillo showed that most of the pages visited by users are located at a lower or equal depth [2]. The maximum size allowed for the contents used in Mercator crawler was 1 MB [43]. We limited the size of the contents to 2 MBs, because the size of the contents has be growing in the past years [34]. We configured a limit number of 10 duplicates per site, 60 seconds to download each content and 200 characters for the URL length. The Crawling Processes were considered dead and restarted by the Watchdogs if they remained inactive for more than 5

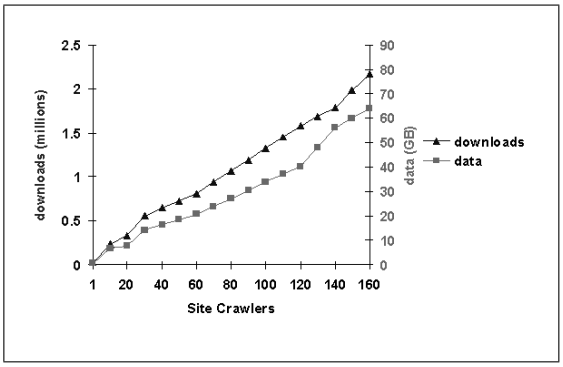
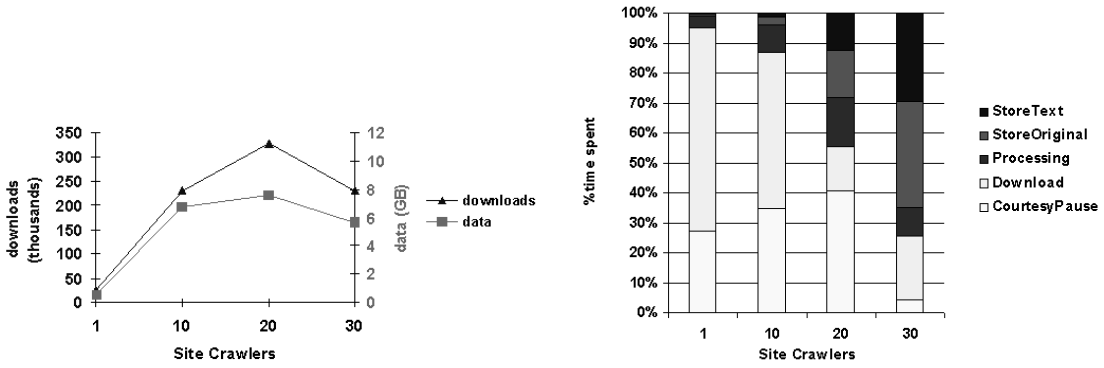


Figure 6: Scalabillity of the download rate with the addition of new CNodes.



a) Data downloaded vs. # CPs per host.

b) Duration of the operations.

Figure 7: Bottleneck analysis.

minutes. VN was configured to respect a courtesy pause of 2 seconds between requests to the same site. Users frequently visit subsequent pages with a time gap of 1 second [1] and a page contains on average 20 embedded images that must also be downloaded [45, 51, 70]. Assuming that a browser executes these downloads sequentially, the time interval between its requests to the server is just 0.05 seconds. Based on this results, the defined courtesy pause for VN imposes less load on the web servers than human browsing.

6.1 Bottlenecks

The Global Frontier is the central point of coordination of VN. So, 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 Crawling Processes spread across several machines. We added a new machine hosting 20 CPs to the cluster daily. Figure 6 presents the obtained results. VN scaled until 160 Crawling Processes executing 2,169,831 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 4

#CPs	#WR requests p/second (w/s)	avg. queue length of the requests (avgqu-sz)	avg. time (ms) for requests to be served (await)	#sectors written p/sec. (wsec/s)
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

Table 5: Disk I/O analysis of the Volume using iostat. The name of the column on the iostat report is presented in parenthesis.

machines with configuration 2. Each one hosted an increasing number of Crawling Processes.

Figure 7a) presents the amount of data crawled within 24 hours 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 until *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 minutes) due to starvation caused by the operating system scheduler but we found no relation. The *StoreOriginal* and *StoreText* series of Figure 7b) 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 seconds 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 1 CP running on a 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 along with the number of Crawling Processes. For 30 CPs, the storage operations spent *64.8*% of the execution time. Table 5 presents an analysis of the disk accesses. It shows that the disk throughput could not cope with the load imposed by the Crawling Processes. The size of the queue of the requests waiting to be served (3^{rd} column) grew as the number of requests issued to the device increased (2^{nd} column). Surprisingly, we observed that the time spent in the CourtesyPause increased to *40.8*% until 20 CPs and then dropped to *4.6*% for 30 CPs. The reason for this phenomenon is that when a machine hosts just 1 CP, the operating system executes the CP immediately after the courtesy pause is reached. However, the time that a CP waited to be executed after performing the courtesy pause 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* seconds, a value close to the configured courtesy pause of 2 seconds. Hence, the time elapsed between two requests due to disk latency to the same site frequently achieved 2 seconds 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 designing a crawler to be deployed on cheap hardware requires additional concern in optimizing disk accesses. We believe that the storage throughput could be improved by using an alternative implementation of the Volumes that would cache larger blocks of data in memory and wrote them sequentially to disk. However, this approach would require the usage of additional memory.

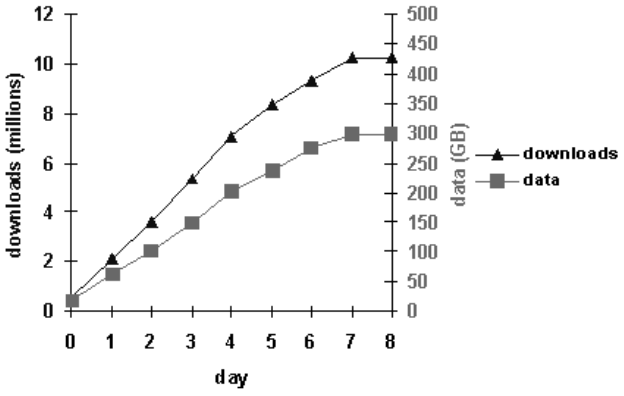


Figure 8: Evolution of a crawl of the Portuguese web.

6.2 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 1 day and do not crawl a single content on the next one because it crashed due to some unexpected problem. To evaluate VN's robustness, we set it up to execute a complete crawl of the Portuguese web. The crawl was bootstrapped with 152,000 seeds generated from a previous crawl. We used 140 Crawling Processes hosted across 7 machines. Figure 8 represents the evolution of the crawl. VN collected a total of 10.262 million documents totalling 299.334 GB in 8 days. During the crawl several problems occurred on the machines like 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 (37.248 GB) and the peak download rate achieved was 1,734,573 documents within 1 day (53.244 GB). Figure 8 shows that the download rate started to decrease on day 7. The reason for this fact is that VN was configured to crawl first 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.

6.3 Text extraction

When crawling the Portuguese web, VN extracts texts from the harvested contents and classifies them to determine if they match the selection criteria (written in the Portuguese language). However, VN could not extract text from 37% of the contents that were not in the HTML format. The objective of this experiment was to measure the effectiveness of the tools used to extract text from the contents and detect the reasons for their unsuccess.

Table 6 describes the file extension of the contents, the tool used to extract text from them, the percentage of contents successfully converted to text and the identified causes of failure. The text extraction process mostly failed for the contents in proprietary formats. Only *19%* of the Microsoft Powerpoint presentations (.ppt, .pps) and Microsoft Excel worksheets (.xls) were successfully converted to text. One reason for this is that the owners of the formats frequently release new versions for commercial purposes and the conversion tools used by our crawler could not extract text from

file extension	tool	ok	conversion	timeout error	max. size	type not allowed	404	other
.ppt, .pps	xlhtml	19%	54%	1%	15%	1%	9%	2%
.xls	xlhtml	19%	60%	13%	1%	0%	6%	1%
.rtf	unrtf	25%	33%	2%	1%	0%	38%	1%
.swf	webcat	36%	53%	1%	0%	5%	4%	1%
.doc	antiword	54%	33%	2%	1%	0%	8%	2%
.ps	ghostscript	59%	6%	25%	3%	0%	5%	2%
.pdf	xpdf	74%	8%	4%	4%	1%	7%	2%
.txt	-	90%	0%	5%	0%	0%	4%	1%
.html, .htm	webcat	94%	0%	0%	0%	0%	4%	2%

Table 6: Analysis of text extraction efficiency.

Parameter	Limit	% Sites	% URLs
# duplicates	10 duplicates	1.6%	–
# URLs	5000 URLs	0.8%	–
URL depth	5	11.3%	–
download time	60 seconds	–	0.4%
size	2 MB	–	0.1%
URL length	200 characters	–	1%

Table 7: Crawling thresholds.

the most recent versions. The column *timeout* shows that *25*% of the postscript documents (.ps) and *13*% of the Microsoft Excel worksheets (.xls) could not be converted to text within 1 minute. The column *max. size* presents the percentage of the contents that were bigger than the configured limit of *2* MB. Most of the files were smaller than this limit but *15*% of the Powerpoint presentations were bigger. These files are big because they usually contain many illustrations. In the *type not allowed* column we can observe that *5*% of the files with extension .swf were not identified with the expected media type "application/x-shockwave-flash", 32% of these files did not return any type on the Content-Type header field, 56% returned the media type "application/octet-stream" and the remaining 12% presented other media types. The MIME specification states that the media type "application/octet-stream" is to be used in the case of uninterpreted binary data, in which case the simplest recommended action is to offer to write the information into a file for the user [31]. Hence, this media type does not seem adequate for flash movies, which must be interpreted by a specific software and are written following a proprietary format.

We conclude that VN requires more efficient tools for extracting text from contents in proprietary formats. However, there are contents that can not be converted to text because their web servers provide erroneous identifications for their media types. When a crawler is setup to execute an exhaustive crawl of contents with different media types, the limit thresholds should be configured according to each media type. For instance, the maximum size allowed for Powerpoint presentations should be higher than for HTML pages.

6.4 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 due to hazardous situations they should not be updated.

Table 7 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 [11]. 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 seconds to be downloaded and processed to extract meta-data. The results obtained by Liu et al. suggested that the timeout of a web client should not be longer than 10 seconds [49]. 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 MBs and just 1% of the URLs were longer than 200 characters.

In a nutshell, 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.

7 Related work

Web characterization is a research area intimately related to crawling. The structural and technological features of the web are determinant to the design of efficient crawlers. On their turn, crawlers are used to generate web characterizations. Najork and Heydon presented a detailed description of the architecture and implementation of a broad crawler named Mercator [43]. Later, Broder et al. investigated URL caching techniques to improve its detection of visited URLs [13]. Mercator was used to study the content of the web, detect hazardous situations and study the evolution of web pages [29]. Boldi et al. presented the Ubicrawler, giving special attention to its fault tolerance and scalability features [9]. It was used to determine the structural properties of the African web [8]. Castillo's thesis discusses effective web crawling techniques, presenting a crawler's implementation and the web characterization results obtained in several experiments performed while harvesting the Chilean and Greek webs [15]. The Kspider is a cluster-based web crawler [47]. Its authors presented technical optimizations and studied the partitioning of URLs among the Crawling Processes to ensure load balancing and avoid overloading the visited web servers. The Kspider contributed to characterize the Thai web [63].

The Google crawler (Googlebot) is superficially described in a research paper about the anatomy of a large-scale search engine [12]. Silva et al. described the CobWeb crawler, one of the components of a search engine for the Brazilian web that used proxy servers to reduce implementation costs and save network bandwidth when updating a set of documents [25].

Shkapenyuk and Suel presented the design and implementation of a high performance distributed crawler named Polybot [65]. Yan et al. presented the architectural design and evaluation of the Webgather crawler aimed to collect pages hosted in Chinese IP addresses [72]. The Dominos web crawler introduced an innovative and totally dynamic architecture, providing high availability and fault tolerance [38]. The Internet Archive introduced an open-source web crawler named Heritrix, specially designed to collect and archive large collections of documents from the web [35].

Cho and Garcia-Molina studied how to order a fixed set of URLs so that the most important pages could be crawled first [22], implemented an incremental crawler [18], proposed and evaluated several architectural alternatives [19] and derived mathematical models to estimate the frequency of change of web pages [20]. Most of their research was applied in the Standford Webase project, where they studied how a crawler should select and refresh the pages kept in a large web data repository [17]. IBM developed a similar project called Webfountain for large-scale text analytics and presented a high level description of an incremental crawler [36].

Monika Henzinger exposed challenges for web search engines [42] and Anna Patterson presented the main difficulties of developing a new search engine [61]. Both researchers described problems

Crawler name	Frontier	Partitioning	Assignment	Meta-data	Focused crawls
Googlebot	centralized	?	?	no	no
Kspider	distributed	page	static	no	no
Mercator	distributed	site	static	yes	no
Polybot	distributed	page	dynamic-push	no	no
Ubicrawler	distributed	site	dynamic-?	no	no
VN	hybrid	site	dynamic-pull	yes	yes
Webase	hybrid	site	dynamic-push	no	no
Webgather	distributed	IP	static	no	no

Table 8: Comparison of design options adopted in several crawlers.

related to crawling. The International Internet Preservation Consortium elaborated a test bed taxonomy listing issues that a crawler should address [10]. Brandman et al. proposed simple modifications in web servers to make them more crawler-friendly by exporting meta-data [11].

7.1 Design comparison

Table 8 compares the design of some of the crawlers described in previous works. In the Googlebot, the Frontier is centralized on a single URL server that somehow distributes them among a set of Crawling Processes.

The Kpsider is composed by a cluster of spiders that collaboratively maintain the URL Frontier and harvest the web. The URL space is partitioned uniformly among the spiders through the application of a hash function on the URLs. However, each spider groups the URLs hosted on the same site to enable the reuse of connections. When a spider finds an URL that does not belong to its partition, it sends it to the correspondent spider using UDP packets.

The Mercator crawler distributes the URL Frontier among several Crawling Processes that communicate among each other using TCP. The URL space is divided using a site partitioning strategy.

In the Polybot crawler, the Frontier is distributed among a set of inter-communicating Crawling Applications and the URL space is partitioned uniformly through a hash function. The Crawling Applications send packets of URLs to the Crawl Manager that assigns each one of them to a Downloader that crawls it. The Crawler Manager ensures that courtesy pauses are respected, caches DNS lookups and robots exclusion files. The communication among the processes is done through a NFS-mounted file system.

The Ubicrawler is composed by inter-communicating agents responsible for managing the Frontier and crawling the web. The assignment is dynamic but there is not a central node of coordination because it is achieve through consistent hashing. Inside each agent the URLs are partitioned per site among the threads, each one dedicated to visit a single site. The number of agents can vary during the crawl which makes Ubicrawler robust to failures and allows to increase download rate by incrementing the number of agents. We could not determine if the agents pull URLs to crawl or if they are pushed from other agents.

The Webgather is composed by a static set of Main Controllers that communicate among each other to jointly manage the Frontier. The URL space is partitioned among the Main Controllers in the beginning of the crawl through the application of a hash function to the IP addresses of the machines to visit (IP partitioning strategy). Each Main Controller has an associated Gatherer that is responsible for crawling the web.

VN presents architectural similarities with the Webase crawler [21]. In the Webase crawler, the URL space is partitioned by site and the Crawling Processes pull seeds from a central point of coordination named the Seed-URL Dispenser. The management of the Frontier is hybrid: the

Crawler name	#machines	Internet (Mbps)	#downloads /second	KB /sec	% dups.	downloads /URLs	Simulation results
Googlebot	5	?	34	200	?	31%	No
Kspider	1	8	15	360	?	92%	No
Mercator	4	100	112	1,682	8.5%	87%	No
Polybot	4	45	77	1,030	13%	85%	No
Ubicrawler	5	116	?	?	?	?	Yes
VN	9	34	25	738	7%	80%	No
Webase	1	1,000	6,000	111,000	?	?	Yes
Webgather	4	?	7	?	?	?	No

Table 9: Performance comparison between crawlers.

Seed-URL Dispenser manages the seeds and each Crawling Processes manages the URLs extracted from the harvested pages. However, the Webase crawler differs from VN in 3 main aspects. First, each crawling process harvests several sites simultaneously. Second, the DNS resolution bottleneck was addressed by solving all the names before the start of the crawl and keeping the associated IP addresses along with the seeds. This approach was time-consuming, lasting 3 days to solve 310,000 domain names and imposed an heavy continuous load on the DNS servers. Third, the Webase crawler harvests only the sites contained in the seeds list. The links to new sites found during the crawl are not harvested. Instead, they are added as seeds of the following crawl. This approach simplifies the crawling task because there is no exchange of URLs between the Crawling Processes, thus, no need for synchronization between the parts of the Frontier managed by them. However, it requires an exhaustive list of seeds that covers completely the portion of the web to harvest and the new sites created since the generation of the seed list are not harvested.

The presented crawlers support both broad and incremental crawls of the web. However, only VN also supports focused crawls. VN extracts meta-data from the contents and stores it during the crawl to prevent redundant processing by the applications that will use the harvested data. Only Mercator presented a similar feature storing statistics on the usage of HTML tags and GIF images.

7.2 Performance comparison

Table 9 presents a performance comparison between the previously analyzed crawlers extracted from published results. This comparison must be taken with a grain of salt because the experiments were run using different setups and in different periods of time. Results gathered on different periods of time require different interpretations because the web is permanently evolving. For instance, the Googlebot in 1998 harvested 34 pages per second 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 on the web that degrade the crawler's performance in practice. 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 the one obtained on simulations. The developers of the Googlebot estimated that it could execute 100 downloads per second (600 KB/s) 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 (6MB/sec.) using 4 machines. However, the results obtained from a crawl of 8 million pages from the Thai web suggest that the download rate would have been of just 232 downloads/sec using 4 machines. The Webase 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 the one presented by other crawlers but it was hosted on a larger number of machines (9). 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 3^{rd}, 4^{th} and 5^{th} columns of Table 9 show that the most performant crawlers used the fastest connections to the Internet. So, this might be also a reason why VN presented a lower download rate than the most performant crawlers.

The performance of a crawler is usually synonymous of its download rate, but there are other features that should be considered. In the *%dups.* column of Table 9, 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, yield 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 9 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 the one achieved by the remaining crawlers, which did not discard any contents.

8 Conclusions

In this paper we shared our experience obtained during the design and operation of the Viuva Negra (VN) crawler. We discussed partitioning techniques to divide the URL space among the processes that compose a parallel crawler and describe thoroughly the architecture of VN. We have shown that there are important architectural choices that must be made, and the decisions are influenced by the crawling application requirements and characteristics of the web. 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. We described situations on the web hazardous to crawling and discussed solutions to mitigate their effects. These hazardous situations were presented in the context of crawling, but they affect HTTP clients in general. So, the presented solutions can be used to enhance other systems that process web data, such as browsers or proxies. We observed that respecting the Robots Exclusion Protocol is not just a politeness measure, 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 can not be detected without human intervention. Sites with different names that simultaneously serve the same content (duphosts) can be detected within a web data set through the intersection of contents between sites or through the analysis of the site names. However, the duphosts are transient, which jeopardizes the usage of this knowledge in a crawler to avoid the download of duplicates.

VN has been used in a production environment during the past 4 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 2 complaints from our crawling activities, which shows that our politeness measures were successful. The obtained results showed that our crawler is robust and scalable. The upcoming of large capacity disks at low prices helped crawlers extending its 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.

VN can be used to execute broad and focused crawls based on content classification. However, as focused crawler that classifies web contents, it must cope with erroneous meta-data available on the web that may prevent the correct interpretation of the harvested contents.

In future work we plan to execute broader crawls of the web to compare VN's performance on

larger harvests. The storage throughput of the harvested contents and the meta-data extraction from documents in proprietary formats are the main problems of the current version of VN. We will improve these aspects by studying alternative storage techniques and conversion tools. We intend to study how web crawlers could be adapted to the Semantic Web.

9 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] What do web users do? an empirical analysis of web use. *Int. J. Hum.-Comput. Stud.*, 54(6):903–922, 2001.

[2] R. Baeza-Yates and C. Castillo. Crawling the infinite Web: five levels are enough. In *Proceedings of the third Workshop on Web Graphs (WAW)*, Rome, Italy, October 2004. Springer LNCS.

[3] D. Barr. *Common DNS Operational and Configuration Errors*, February 1996.

[4] L. Bent, M. Rabinovich, G. M. Voelker, and Z. Xiao. Characterization of a large web site population with implications for content delivery. In *Proceedings of the 13th international conference on World Wide Web*, pages 522–533. ACM Press, 2004.

[5] E. Berk and C. S. Ananian. Jlex: A lexical analyzer generator for java(tm). http://www.cs.princeton.edu/ appel/modern/java/JLex/, February 2005.

[6] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*, January 2005.

[7] K. Bharat, A. Z. Broder, J. Dean, and M. R. Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *Journal of the American Society of Information Science*, 51(12):1114–1122, 2000.

[8] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Structural properties of the African web. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[9] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. In *Proceedings of the Eighth Australian World Wide Web Conference*, July 2002.

[10] A. Boyko. Test bed taxonomy for crawler. Technical report, International Internet Preservation Consortium, July 2004.

[11] O. Brandman, J. Cho, H. Garcia-Molina, and N. Shivakumar. Crawler-friendly web servers. In *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*, Santa Clara, California, 2000.

[12] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[13] A. Z. Broder, M. Najork, and J. L. Wiener. Efficient url caching for world wide web crawling. In *Proceedings of the twelfth international conference on World Wide Web*, pages 679–689. ACM Press, 2003.

[14] J. Campos. Versus: a web repository. Master thesis, 2003.

[15] C. Castillo. *Effective Web Crawling*. PhD thesis, University of Chile, November 2004.

[16] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.

[17] J. Cho. *Crawling the Web: Discovery and maintenance of large-scale web data*. PhD thesis, Stanford University, November 2001.

[18] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 200–209, September 2000.

[19] J. Cho and H. Garcia-Molina. Parallel crawlers. In *Proceedings of the eleventh international conference on World Wide Web*, pages 124–135. ACM Press, 2002.

[20] J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM Trans. Inter. Tech.*, 3(3):256–290, 2003.

[21] J. Cho, H. Garcia-Molina, T. Haveliwala, W. Lam, A. Paepcke, S. Raghavan, and G. Wesley. Stanford webbase components and applications. Technical report, Stanford Database Group, July 2004.

[22] J. Cho, H. García-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[23] J. Cho and S. Roy. Impact of search engines on page popularity. In *Proceedings of the 13th international conference on World Wide Web*, pages 20–29. ACM Press, 2004.

[24] A. L. da Costa Carvalho, A. J. de Souza Bezerra, E. S. de Moura, A. S. da Silva, and P. S. Peres. Detecção de réplicas utilizando conteúdo e estrutura. In *SBBD*, pages 25–39, 2005.

[25] A. S. da Silva, E. A. Veloso, P. B. Golghe, B. Ribeiro-Neto, A. H. F. Laender, and N. Ziviani. Cobweb a crawler for the brazilian web. In *Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, page 184. IEEE Computer Society, 1999.

[26] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. C. Mogul. Rate of change and other metrics: a live study of the world wide web. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[27] J. Edwards, K. S. McCurley, and J. A. Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *World Wide Web*, pages 106–113, 2001.

[28] N. Eiron, K. S. McCurley, and J. A. Tomlin. Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web*, pages 309–318. ACM Press, 2004.

[29] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 669–678, New York, NY, USA, 2003. ACM Press.

[30] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, June 1999.

[31] N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, November 1996.

[32] D. Gomes, S. Freitas, and M. J. Silva. Design and selection criteria for a national web archive. In *Proc. 10th European Conf. Research and Advanced Technology for Digital Libraries, ECDL*. Springer-Verlag, September 2006.

[33] D. Gomes, A. L. Santos, and M. J. Silva. Managing duplicates in a web archive. In L. M. Liebrock, editor, *Proceedings of the 21th Annual ACM Symposium on Applied Computing (ACM-SAC-06)*, Dijon, France, April 2006.

[34] D. Gomes and M. J. Silva. Characterizing a national community web. *ACM Trans. Inter. Tech.*, 5(3):508–531, 2005.

[35] M. S. I. R. Gordon Mohr, Michele Kimpton. Introduction to heritrix, an archival quality web crawler. In *4th International Web Archiving Workshop (IWAW04)*, Bath, UK, September 2004. Internet Archive, USA.

[36] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. How to build a webfountain: An architecture for very large-scale text analytics. *IBM Syst. J.*, 43(1):64–77, 2004.

[37] M. A. Habib and M. Abrams. Analysis of sources of latency in downloading web pages. In *WebNet*, San Antonio, Texas, USA, November 2000.

[38] Y. Hafri and C. Djeraba. High performance crawling system. In *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 299–306. ACM Press, 2004.

[39] P. M. Hallam-Baker and D. Connolly. Session identification uri. http://www.w3.org/TR/WD-session-id.html, November 2005.

[40] S. Handschuh, S. Staab, and R. Volz. On deep annotation. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 431–438, New York, NY, USA, 2003. ACM Press.

[41] D. Hawking and N. Craswell. Very large scale retrieval and web search. In E. Voorhees and D. Harman, editors, *The TREC Book*. MIT Press, 2004.

[42] M. R. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. *SIGIR Forum*, 36(2):11–22, 2002.

[43] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[44] HypersonicSQL. `http://sourceforge.net/projects/hsqldb/`, 2001.

[45] A. Jaimes, J. R. del Solar, R. Verschae, D. Yaksic, R. Baeza-Yates, E. Davis, and C. Castillo. On the image content of the chilean web. In *LA-WEB '03: Proceedings of the First Conference on Latin American Web Congress*, page 72. IEEE Computer Society, 2003.

[46] B. Kahle. The internet archive. *RLG Diginews*, 6(3), June 2002.

[47] K. Koht-Arsa. High performance cluster-based web spiders. Master's thesis, Graduate School, Kasetsart University, March 2003.

[48] M. Koster. A standard for robot exclusion. http://www.robotstxt.org/wc/norobots.html, June 1994.

[49] B. Liu. Characterizing web response time. Master's thesis, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, April 1998.

[50] N. Ltd. Netcraft: Web server survey archives. http://news.netcraft.com/archives/web_server_survey.html, November 2005.

[51] M. Marshak and H. Levy. Evaluating web user perceived latency using server side measurements. *Computer Communications*, 26(8):872–887, 2003.

[52] B. Martins and M. J. Silva. Language identification in web pages. In L. M. Liebrock, editor, *Proceedings of the 20th Annual ACM Symposium on Applied Computing (ACM-SAC-05)*, Santa Fe, New Mexico, March 2005.

[53] B. Martins and M. J. Silva. The webcat framework : Automatic generation of meta-data for web resources. In *Proceedings of WI-2005, The 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, September 2005.

[54] P. Mockapetris. *Domain names - concepts and facilities*, November 1987.

[55] M. Najork and A. Heydon. On high-performance web crawling. Src research report, Compaq Systems Research Center, 2001.

[56] I. C. F. A. Names and Numbers. Icann — verisign's wildcard service deployment, November 2004.

[57] A. Ntoulas, J. Cho, and C. Olston. What's new on the web?: the evolution of the web from a search engine perspective. In *Proceedings of the 13th international conference on World Wide Web*, pages 1–12. ACM Press, 2004.

[58] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 100–109, New York, NY, USA, 2005. ACM Press.

[59] S. Olsen. Does search engine's power threaten web's independence? *CNET News.com*, October 2002.

[60] Oracle9i. `http://www.oracle.com/ip/index.html?content.html`.

[61] A. Patterson. Why writing your own search engine is hard. *Queue*, 2(2):48–53, 2004.

[62] L. D. Paulson. Building rich web applications with ajax. *Computer*, 38(10):14–17, 2005.

[63] S. S. Punpiti. Measuring and analysis of the Thai world wide web. In *Proceedings of the Asia Pacific Advance Network*, pages 225–230, August 2000.

[64] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 129–138. Morgan Kaufmann Publishers Inc., 2001.

[65] V. Shkapenyuk and T. Suel. Design and implementation of a high-performance distributed web crawler. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, page 357. IEEE Computer Society, 2002.

[66] H. Snyder and H. Rosenbaum. How public is the web?: Robots, access, and scholarly communication. Working paper WP-98-05, Center for Social Informatics, Indiana University, Bloomington, IN USA 47405-1801, January 1998.

[67] The Web Robots Pages. Html author's guide to the robots meta tag. http://www.robotstxt.org/wc/meta-user.html, March 2005.

[68] M. Thelwall. A free database of university web links: Data collection issues. *International Journal of Scientometrics, Informetrics and Bibliometrics*, 6/7(1), March 2002.

[69] Unknown. Spider traps - an upcoming arms race. http://www.jahns-home.de/rentmei/html/sptraps.html, November 2004.

[70] C. E. Wills and M. Mikhailov. Examining the cacheability of user-requested Web resources. In *Proceedings of the 4th International Web Caching Workshop*, 1999.

[71] A. Woodruff, P. M. Aoki, E. Brewer, and P. Gauthier. An investigation of documents from the world wide web. In *Proceedings of the fifth international World Wide Web conference on Computer networks and ISDN systems*, pages 963–980, Amsterdam, The Netherlands, The Netherlands, 1996. Elsevier Science Publishers B. V.

[72] H. Yan, J. Wang, X. Li, and L. Guo. Architectural design and evaluation of an efficient Web-crawling system. *The Journal of Systems and Software*, 60(3):185–193, 2002.