Computer Science and Information Systems 12(1):91-114 DOI: 10.2298/CSIS131120081P

Distributed and Collaborative Web Change Detection System

Víctor M. Prieto, Manuel Álvarez, Víctor Carneiro, and Fidel Cacheda

Communication and Information Technologies Department
University of A Coruna, Campus de Elviña s/n - 15071 (A Coruna)
{victor.prieto, manuel.alvarez, victor.carneiro, fidel.cacheda}@udc.es

Abstract. Search engines use crawlers to traverse the Web in order to download web pages and build their indexes. Maintaining these indexes up-to-date is an essential task to ensure the quality of search results. However, changes in web pages are unpredictable. Identifying the moment when a web page changes as soon as possible and with minimal computational cost is a major challenge. In this article we present the Web Change Detection system that, in a best case scenario, is capable to detect, almost in real time, when a web page changes. In a worst case scenario, it will require, on average, 12 minutes to detect a change on a low PageRank web site and about one minute on a web site with high PageRank. Meanwhile, current search engines require more than a day, on average, to detect a modification in a web page (in both cases).

Keywords: Content refresh, Incremental crawling, Crawling systems and Search engines.

1. Introduction

Currently, the WWW is the biggest information repository ever built. According to the study presented by Gulli and Signorini [17], the Web consists of thousands of millions of pages. Furthermore, it is continuously growing. Due to its large size, it is indispensable to use search engines to access the information which is relevant to the user. Search engines are complex systems that allow several operations to be performed with the information, such as collecting, storing, managing, locating and accessing it. Systems that perform the task of gathering data are the crawlers, programs that can traverse and analyze the Web in a certain order, following links between different pages.

A crawling system faces many challenges because of the quantity, the variability and the quality of the information that it has to gather. Among others, we can highlight these features: technologies that we have to consider to access the documents, on both the server side [28] and the client side [8], problems related to the Web content and the Web Spam [19] [15], repeated contents [23], etc. Moreover, because the contents of the Web are continually changing, once a crawling system has finished traversing the Web, the crawler must start again to maintain the content indexed by the search engines up to date.

At this point another important challenge for a crawling system appears: not all pages change at the same frequency. Some web pages are highly variable, so the time spent on a complete traversal of the Web could be too great to ensure the quality of the user searches. For this reason, search engines consider other elements to improve the quality of their

content with the least impact on performance. The most common is to use recrawling policies, based on the changes history of each resource and the relevance thereof. Based on them, the crawler will perform a recrawling process for updating the copy of each document in its repositories. After that, but not necessarily immediately, the indexes will be regenerated, updating the content for the users' queries. The recrawling policies also consider information provided by web sites through different mechanisms:

- The HTTP protocol provides the Last-Modified header to indicate the last modified date of a resource. In some cases, the web servers have not correctly implemented this mechanism or in other cases the servers use content managers to dynamically generate the web pages. In any case, the crawler requires another access to the resource in order to detect any changes.
- Some web servers provide RSS[1] feeds to provide information about the changes in the web pages. However, the RSS systems are not used by all websites and their use requires a great deal of work by the creators.
- Other servers can provide information about change frequency in their resources through the robot exclusion protocol (file "robots.txt" and "sitemap.xml"). The disadvantage of this protocol is that it needs to have access to the root directory, and in a lot of cases the web page creator does not have this type of access. Moreover, it is complex and expensive to maintain, because it is necessary to know exactly the change frequency of each web resource [30].
- Finally, in other cases, a crawler may have agreements with some web content providers to receive notifications of their changes. These agreements are only carried out with great web sites where both the search engines and the company are interested in keeping their content updated, therefore for the major part of the web sites these agreements are impossible.

The underlying problem in the recrawling process is to know when the content has changed. Ideally, a crawler would have to recrawl a page just after it has been modified. If it is accessed before it has changed, the visit is useless. On the other hand, the longer it takes the crawler to revisit the updated page, the information becomes more and more stale (and probably less relevant). Furthermore, another important aspect of the recrawling process is how much the content has changed. It is useless to process a page and observe that the changes found are minimal or irrelevant to the indexing process.

We could divide the recrawling process in two different parts:

- To detect when a web page has been modified.
- To recrawl that web page.

This work is focused on the first part. We present the Web Change Detection (WCD) system, which is capable of detecting when a web page changes faster than current state of the art techniques. At present, crawling systems, in general, try to guess when a web page has changed and must face all the problems discussed above.

The WCD system can notify the crawler when a web page has changed and therefore improve the recrawling policy of the search engine. Thus, the content repository generated by the crawler will be more up-to-date, thereby improving the quality of the results presented to the web users. The WCD system is based on a distributed and collaborative

[1] http://www.rssboard.org/rss-specification

architecture, similar to the one used by "Google Analytics" [2] to obtain information about page usage (source IP, time on page, links used, etc.). In our case we use the passage of a user through a web resource as an agent for web change notification. For this, the web creator simply embeds a reference to the WCD agent on the monitored pages, which will notify any change to the WCD server.

This system presents the following advantages over the technologies currently used by crawling systems in order to detect the best time to recrawl a web page:

- It does not depend on the implementation of the Last-Modified in the web server, nor of the maintenance of files with information on change estimated frequency, nor agreements with the owners of the web sites.
- Part of the processing to decide whether the web page has been modified is performed on the client side, saving resources in the crawling system.
- It allows to detect the web modifications faster than the existing techniques, and at the same time, optimises the usage of the resources. This is due to the fact that no unnecessary visits to the web pages are required.

In short, the mechanisms used by search engines are based on statistical information, guessing when a web page has changed. On the other side, the WCD system is notified when a web page has been modified. The main benefit is that the freshness of the web pages indexed is improved and, consequently, the quality of the search results is improved.

The structure of this paper is as follows. Section 2 discusses the related work and Section 3 presents the motivation behind this article. Section 4 thoroughly explains the architecture of the system, its main features and its advantages with regard to other models. Section 5 describes the performed experiments and discusses the obtained results. We present results about the performance of the system and its efficiency in detecting the web changes faster than the crawlers of the current search engines. Finally, sections 6 and 7 detail the conclusions we have obtained and possible future work, respectively.

2. Related Work

First, we present a series of studies aimed at characterizing the Web. The knowledge of their characteristics and structure is essential to select the most appropriate policies to ensure the repositories are as up-to-date as possible. Among the studies examining Web features is that performed in 2007 by Baeza-Yates *et al.* [6], where they discuss various features of the Web. The same authors presented in 2004 and 2005 two articles, [29] and [5], respectively, focusing specifically on the characteristics of the Spanish and Chilean Web. Another relevant study is [7], where Baeza-Yates *et al.* analyze the structure of the Web and its dynamism.

There are also a number of studies focused on characterizing Web dynamism, as that presented by Brewington and Cybenko [9], where the authors analyze the dynamism of web content, and propose improving the updating of indexes by using empirical models and a new metric that measures the frequency of changes. Similarly [10], Cho *et al.* discuss the dynamism of the Web and its implications for incremental crawling. Finally, we want to highlight the article [24], where Lewandowski performed a study over 3 years

[2] http://www.google.com/analytics/

analyzing the update strategies of different search engines on a set of web pages. He concluded that none of the search engines provided up-to-date copies, even for the daily updated pages.

On the other hand, there are numerous works that present methods of keeping the repositories of search engines up-to-date. Among them is a study performed by Cho and Garcia-Molina [12], where the authors not only discuss the dynamics of web content, but also propose estimators for the frequency of changes for improving the refreshment of the contents of a search engine. The same authors in the article [11], propose a new policy for content refresh and compare it with the other policies: static, semi-random and completely random. Similar to the above article is a 2003 paper by Fetterly *et al.* [16], where the authors studied a set of pages for several weeks looking for models to help to define refreshing policies. There are other approaches like Kharazmi *et al.* [21], that proposes using data mining techniques to extract information that helps to improve the refresh policy of a search engine. In 2009, Pandey and Olston presented a study [27], where they try to characterize the longevity of web pages, based on the divergence of the fragments that make up a page in the different visits made by the crawler. Based on this longevity metric they developed various refresh policies.

Also there are multiple applications or services for detecting changes on a few web pages, such as TrackEngine[3] or ChangeDetection.com[4]. These tools access to the web pages at configured time intervals and then compare their content to detect changes. These applications could be used on a limited scope (e.g. for technology watch tasks) where the number of web pages to monitor is reduced but do not operate on a Web scale.

The WCD system presented in this article differs from these previous works in how the problem is faced. While these works try to keep the repositories updated by estimating when the web resource may have changed, our system is able to detect in a few minutes when a web resource has changed. To the best of our knowledge, this is the first work presented in this direction to solve the problem of the updating of search engine repositories.

3. Motivation

The first step in this work is to study the behavior of the main search engines (Google, Yahoo! and Bing) when recrawling a web page. In this sense, we have run an experiment to measure when a web page changes and when it is detected by these search engines.

Specifically, we have randomly selected a set of 150 web domains from the web site SeeTheStats[5]. This web site provides the URL for web domains that are willing to share some information about user accesses from Google Analytics. For each web domain, only the home page was considered in our experiment. In order not to bias the experiment, two different types of web sites were considered: web pages with low PageRank (between 0 and 2), and web pages with high PageRank (between 3 and 5). Twenty-five sites for each PageRank were randomly selected. Web sites with higher PageRank were not considered because web sites with PageRank higher than 5 are not available on the SeeTheStats web site. This distinction was made because we considered that the relevance of a web page

[3] `http://www.trackengine.com`
[4] `http://www.changedetection.com`
[5] `http://www.seethestats.com`

(i.e. its PageRank) could be one of the factors used in the recrawling policy by search engines (i.e. more effort will be made to keep more important pages up-to-date).

To determine when a search engine detects a change, we have obtained the cached version of each web page from the three search engines. More specifically, the cache date provided us with the information of the last access to the web page. Each web page was monitored during 30 days (from April 30th to May 30th, 2012). Due to search engine limitations, each web page could be only monitored every 12 hours.

At the same time, to detect when a web page changed, each page was being monitored every 12 hours using our own crawling system, for the same period of time. The main difficulty of this experiment was to decide when a web page has been modified. For this, we have extracted and compared the useful content for each version of the web page. To extract the useful content, we have used the approach proposed by Donghua *et al.* in [14]. This approach is based on the location of the main content being very centralized and having a good hierarchical structure. The authors have proposed an algorithm that judges the content by several parameters in the nodes (Link Text Density, Link Amount, Link Amount Density and Node Text Length). They found that the values of these parameters for the DOM nodes [1] with useful content are significantly different from that of other DOM nodes in the same level.

In order to compare the useful content of each version of the web page, and to decide whether the web page has been modified, we have followed the approach proposed by Manku *et al.* [25]. For this, we have used the $shash$ tool to detect near duplicates [6]. This tool divides each document in n tokens, each one of them with a weight and calculates a hash of each token. Finally, with the weight and the hash of each token, it creates a fingerprint for the useful content of the web page. In their previous work, Manku *et al.* considered that a pair of documents are near duplicate if and only if their fingerprints are at most 3-bits apart. In our work, we have considered that two web page versions are near duplicates if their fingerprints differ at least 6 bits. In this way, small or minimum changes in a web page, which do not mean real modifications in the content, are discarded and only significant changes are considered. In this way, we only require to the search engines that detect significant changes in the web pages.

Finally, with this data, we can measure:

- How much time search engines have their data stale for.
- The number of changes not detected by the search engines. This occurs when a web page is modified twice or more between two crawler visits.
- The number of times crawlers return to a web page unnecessarily.

First, we explore in Table 1 a general view of the results obtained. In this table, and in the following, each column represents the obtained result for a set of web sites with a particular PageRank. The columns $LowPR$ and $HighPR$, represent the mean for the web sites with PageRank between 0 and 2, and 3 and 5, respectively.

On one side, we present the average time spent between two consecutive changes. As expected, as the PageRank increases so does the frequency of the changes: a page with PageRank 5 is changed, on average, twice as much as a page with PageRank 0. That is, the more important a web site is, the more dynamic will be, changing its content more

[6] `http://d3s.mff.cuni.cz/~holub/sw/shash/`

frequently. On average, a site with low PageRank will change almost 11 times in a month, while a site with high PageRank will change nearly 17 times.

On the other side, the average re-visiting time of the main search engines follows the same pattern: pages with higher PageRank are visited more frequently. The three search engines will re-visit the pages with higher PageRank, on average, every two and a half days. In the pages with lower PageRank, Bing, Google and Yahoo!, will re-visit every four days, on average.

In general terms, search engines follow a conservative approach: they will re-visit a web page when it is highly probable that a change has occurred. This causes their index to be outdated and some modifications to be missing for some time.

Table 1. Frequency of web page changes and search engines accesses. Time units are hours

	Low PageRank				High PageRank			
	PR 0	PR 1	PR 2	**Low PR**	PR 3	PR 4	PR 5	**High PR**
Change	76.02h	66.02h	60.01h	**67.35h**	51.83h	41.33h	36.83h	**43.33h**
Google	114.17h	111.71h	98.74h	**108.10h**	78.47h	57.07h	65.82h	**67.12h**
Yahoo!	121.24h	96.80h	86.21h	**101.42h**	84.42h	66.23h	59.64h	**70.10h**
Bing	108.34h	96.79h	92.33h	**99.16h**	84.42h	63.97h	60.24h	**69.55h**

Table 2 provides more detailed results obtained for the domains analyzed, grouped by PageRank. All results are average values for the 150 domains studied. The row *Outdated Time* refers to the percentage of time search engines have their index stale, that is, a web page has changed, but the search engine has not yet realized. *Unnecessary Refresh* counts the number of visits performed by the crawlers when the web page has not changed and *Undetected Changes* counts the number of times a web page has changed between two consecutive search engine accesses.

In the case of Google, we observe that for sites with low PageRank, its cache and indexes are outdated 61.31% of the time. For web sites with high PageRank, that percentage decreased to 53.00%, about 8 points less. Focusing on the average number of unnecessary accesses made by Google's crawler, they were less than one. This implies that during the whole month studied, the crawler only once made an unnecessary visit. However, it skipped an important amount of modifications: 6.91 for low and 7.5 for high PageRank sites. Also note that, although the number of missing changes is higher for the more important sites, as they change more frequently, the percentage of undetected changes is lower for the high PageRank sites (45% of the changes are unnoticed) than for the low PageRank sites, where nearly 65% of the changes are missed.

In summary, Google uses a conservative approach, revisiting a web page only when it is quite certain that the resource may have changed. Google focuses on the pages with higher PageRank, revisiting them more frequently and, therefore, detecting more changes. However, its indexes are outdated more than half of the time in both types of sites (low and high PageRank) and many modifications are not detected (being worst for sites with low PageRank).

Table 2. Detailed results for Google, Yahoo! and Bing

	Google							
	PR 0	PR 1	PR 2	**Low PR**	PR 3	PR 4	PR 5	**High PR**
Outdated Time	50.19%	69.20%	64.54%	**61.31%**	51.39%	38.08%	69.51%	**53.00%**
Unnecessary Refresh	1	0.50	0.63	**0.71**	0.18	0.61	1.12	**0.64**
Undetected Changes	7.69	7.42	5.63	**6.91**	9.45	6.30	6.75	**7.50**

	Yahoo!							
	PR 0	PR 1	PR 2	**Low PR**	PR 3	PR 4	PR 5	**High PR**
Outdated Time	59.48%	46.62%	43.67%	**49.92%**	62.89%	60.24%	53.61%	**58.92%**
Unnecessary Refresh	1	0.1	1.27	**0.79**	1.54	1.23	2.5	**1.75**
Undetected Changes	6.92	5.25	5.27	**5.81**	8.18	5.69	6.62	**6.83**

	Bing							
	PR 0	PR 1	PR 2	**Low PR**	PR 3	PR 4	PR 5	**High PR**
Outdated Time	53.87%	46.62%	42.52%	**47.67%**	62.89%	54.80%	55.15%	**57.61%**
Unnecessary Refresh	1.00	0.3	1.09	**0.79**	1.27	1.15	2.37	**1.60**
Undetected Changes	6.92	4.84	3.45	**5.07**	9.36	5.23	4.87	**6.49**

The results obtained by Yahoo! indicate that their repositories are outdated 49.92% of the time for websites with low PageRank and 58.92% for sites with high PageRank. Regarding the unnecessary refresh, the results are slightly worse than Google's, and for the undetected changes, reciprocally, the results are slightly better.

Yahoo!'s repositories are outdated, on average, half of the time, but it is able to obtain slightly better global results than Google (54.42% vs. 57.16% outdated time). However, Yahoo! is less balanced, with better results on the sites with lower PageRank. The same conservative approach is observed in the Yahoo! results.

Bing and Yahoo! results are quite similar, as expected from [2]. Bing obtains the lowest average outdated time (52.64%), with better results for low PageRank sites (similarly to Yahoo!). The unnecessary refresh and the undetected changes are similar to the values obtained by Yahoo! but slightly better.

In summary, we have observed, as expected, that the main search engines visit more frequently web sites with higher PageRank (missing less changes). Also, a conservative approach is observed in the three search engines studied, preferring to keep their repositories outdated than consuming resources to re-visit an unmodified page.

Finally, the main difference between the three search engines analyzed is that Yahoo! and Bing keep their repositories more updated for web sites with low PageRank, while Google has a similar behavior for both types. This may seem awkward, but it is related to the work by Arasu *et al.* [4]. In this work, the authors proposed that, when a certain page changes too often and if we cannot maintain it up-to-date under our resource constraints, it is better to focus our resources on the pages that we can keep track of.

In this section we have analyzed the outdated time of the repositories of the main search engines (Google, Yahoo! and Bing). In general terms, the three search engines have a similar behavior, with some minor differences. However, it is clear that this model

produces quite important gaps in the data indexed by the search engines that may have an impact in the search user experience. In the following section, we present the WCD system that tries to solve this problem.

4. System Architecture

The objective of the WCD system is to detect any change on a web page. When a web page is inserted in the WCD system, it is said that it is being monitored. From that moment on, the WCD system will detect the changes produced on the page and it will notify them to the search engine. It is then the task of the search engine to decide when to crawl that page again, based on its revisiting policy and its resources available.

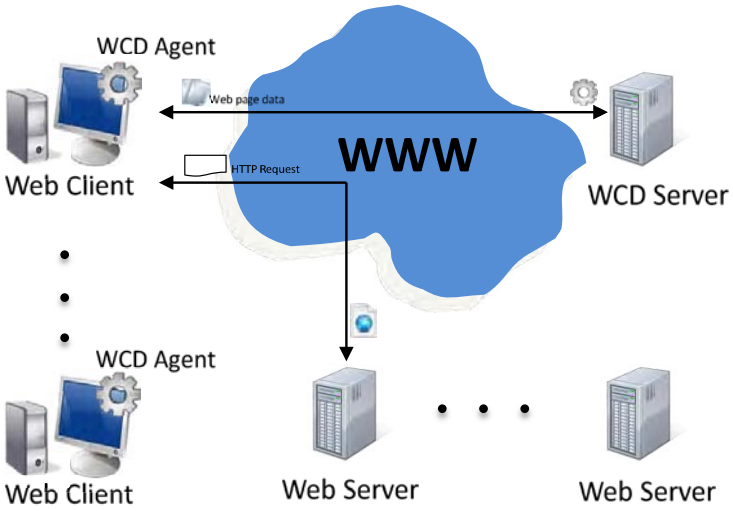


Fig. 1. Collaborative architecture of the WCD system

In this section, we will explain and discuss the architecture of the WCD system. First, we will describe roughly the architecture of the system. Then, we will explain in detail the components of the architecture and the interaction between them. After that, we will comment on the integration of the system in a search engine. Finally, we will describe the advantages and disadvantages of the proposed system.

4.1. WCD System Architecture

The WCD system is based on the distributed architecture of the World Wide Web and incorporates a WCD server and a WCD agent to create a collaborative architecture in order to detect web page changes. These are the main components of the WCD system architecture:

- Web client.

- Web server.
- WCD agent.
- WCD server.

The web client is a common web browser and it is responsible for loading the web page, requesting to the WCD server the WCD agent and executing it. The web server is a common web server that stores one or more monitored web pages. Each monitored web page will include a reference to the WCD agent. As usual, when a web client requests a web page, the server will send its content to the browser.

The WCD agent is a browser-based application that will send information about the web page modifications to the WCD server. The WCD server has two main tasks. On one hand, it will store and send the WCD agents to the web clients. On the other hand, it will store and update the data about all the monitored web pages. In Figure 1 we show the high level architecture of the system.

In short, the WCD system uses a push model, that is, the WCD Agent notifies changes in web pages to the WCD server just when the web pages are visited. In the following subsections we discuss in detail the WCD server and Agent.

WCD server The WCD server is composed of two subsystems:

- The Agent Manager: it will handle the WCD agents (using the Agent Storage), it will receive the agent requests sent by the web browsers and decides (based on different parameters) which WCD agent to send (see next section).
- The WCD Change Detector: it is responsible for receiving the web page information sent by the WCD agents and detecting the modifications, updating the web page information (e.g. the last modification date) on the WCD Repository.

The WCD Repository stores all the information about the monitored web pages. For each page, the following data is stored:

- Summary: it is computed by the WCD agent.
- Last modification date.
- Last access date.
- Change rate: defined as the number of detected changes per time unit (e.g. hours).
- Importance of the page: for example, using PageRank.

Figure 2 shows the elements of the WCD server. The server provides two interfaces: the Agent Management Interface, that will receive the WCD agent requests sent by the web browsers, and the WCD Data Interface, that will receive the web page information submitted by the WCD agents.

The requests received by the Agent Management Interface are processed by the Agent Manager. The Agent Manager could use the information stored in the WCD Repository (e.g. the last access date) and some information about the client (e.g. browser type and version) to decide which agent should be sent in each case.

The information received through the WCD Data Interface is processed by the WCD Change Detector. This module will compare the web page summary submitted by the agent with the one stored in the repository in order to detect if any change has been produced. The information in the repository will be updated: the last access date and the remaining fields if any change has occurred.

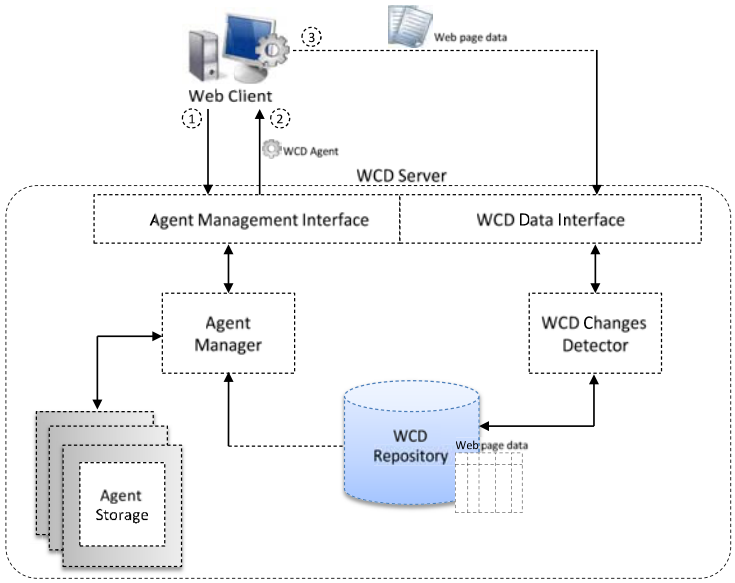


Fig. 2. Components of the WCD server

WCD agent The WCD agent is a program that is downloaded from the WCD server to be executed on the web browser client. It will create a summary of the web page and it will send the web page information back to the WCD server for monitoring purposes.

Initially, we consider only two different types of WCD agents, the Web Digester Agent and the Void Agent, although the architecture is designed to operate with any number of WCD agents.

The Web Digester Agent will compute the summary of the page by calculating the MD5 hash on its useful content. The useful content refers to its main content, where the information is really placed, without HTML tags, links, images or other similar formatting information. We follow the approach developed by Donghua Pan [14], to extract the useful content of the following parts of the web page: title, header, body and footer. Then, the agent will calculate the MD5 hash for each part (this constitutes the web page summary) and send the following information to the WCD server: the summary, the URL and the current date.

The WCD agent sends the web page data to the WCD server through an asynchronous request [20]. In this way the web users do not observe delays and their experience in the corresponding web page is not affected.

The Void Agent will not compute any summary on the client browser and, therefore, will not send any notification to the WCD server. This agent is useful when the WCD server considers that the web page information is updated, for example, if the last access has been just a few seconds ago.

The WCD server can send the corresponding WCD agent depending on a number of factors: local (as time spent from the last update) or remote (as the version of the web browser). For example, we can consider the time spent since the last data update.

The next section explains in more detail the operation between the web server, the web browser, the WCD server and the WCD agent.

4.2. WCD Operation

In this section we will describe the operation of the WCD system. First of all, it is important to highlight that the WCD system does not require any changes, add-ons or extensions to the web server nor the web client. Its operation is based on the WCD agent and Server.

From the webmaster point of view, the operation with the WCD system is quite straightforward. Once a webmaster decides to include a web page in the WCD system, he must insert a reference to the WCD agent in the web page.

Once this is done, the common scenario of the WCD system is as follows (the steps can be seen in Figure 2):

1. First, the user makes a request from the web browser to the corresponding web server. The web server responds by sending the content of the web page, which includes the reference to the WCD agent.
2. Then the web browser will request the WCD agent to the WCD server, to complete the web page content. The Agent Manager will query the repository for the last update time for the web page. Depending on the time spent since the last update, the Agent Manager will send a Web digester Agent or a Void Agent.
3. Finally, the web browser will execute the WCD agent. If the Web Digester Agent was sent to the client, it will generate the summary of the web page and send it back to the WCD server. Then, the WCD Change Detector will compare the page data with the data stored in the WCD repository, and will update it if necessary.

4.3. WCD Integration

Current search engines use a crawler to continuously traverse the Web looking for new web pages and changes on the web pages already indexed. The crawler uses multiple crawling processes to improve performance, that could perform different types of traversals of the Web. Some of them will visit the Web periodically, i.e., re-starting the traversal when it is already completed. Another group will conduct a crawling of certain pages or web sites, based on recrawling policies. Among these policies we could remark the following: a) the page relevance, b) the frequency of updates of each page and web site, and c) the type of website (newspapers, business pages, blogs, opinion pages, etc.).

In order to detect web page changes, current search engines follow a pull method. Based on several statistical data (i.e. relevance or changes rating in the past), the search engine guesses that the web page has probably changed and therefore a recrawling is worthwhile.

A search engine with its WCD system integrated will be able to know that a web page has changed and then decide if it is worth recrawling.

The integration of the WCD system in a search engine architecture is done through the WCD Access Interface (see Figure 3). This interface provides management and access operations to the information contained in the WCD Repository to the search engine and viceversa, the WCD system can notify the search engine about web page changes.

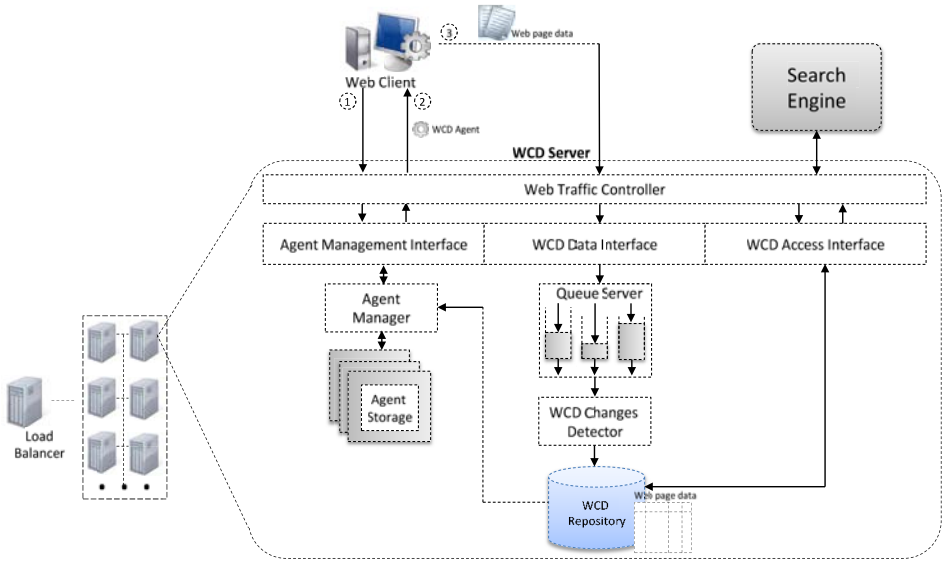


Fig. 3. Complete architecture of the WCD system

The communication between the search engine and the WCD system could work in three different modes: pull, push or a combination of both, according to the needs of the corresponding search engine. In the pull mode, the search engine queries to the WCD system for the web pages modified. In the push mode, the WCD system will notify the search engine about which web pages have been modified. As both modes are complementary, they can also be combined.

In order to use the WCD system in a real search engine, certain considerations must be taken into account regarding security and scalability.

In some situations, a user could modify the data sent to the system by the web browser agent to notify that the page has changed when in fact it has not. Actually, this is not an important issue, because this only implies that the web page will be revisited unnecessarily at some point, but no invalid content is included in the search engine. Nevertheless, for managing these situations, the WCD system uses a module called Web Traffic Controller. This module is responsible for allowing or disallowing the access to the WCD system. In order to do this, the Web Traffic Controller maintains an index with the IPs and URLs that have to be filtered due to security issues. This data is stored in the WCD Repository. When the crawler reprocesses the page, it will check if the page has really changed. If the modification has not occurred, the crawler will notify the WCD system through the WCD Management Interface, so the WCD Traffic Controller can store the IP from which the manipulation has been performed, to filter the next notifications sent from this IP. The Web Traffic Controller also maintains a list of recent accesses for detecting DDoS attacks (Distributed Denial of Service) and filtering IPs from the generated blacklist.

Regarding scalability issues, the system must be resistant to sporadic load peaks caused by heavy use of the system, or by DDoS attacks. To deal with these situations,

when the system detects an excessive rise in the load, the system follows the protocol described below (we differentiate the WCD Management interface and the WCD Data Interface):

- WCD Management Interface: in this scenario, the system will always return the Digester Agent avoiding querying the WCD repository. This is to avoid losing any notification of changes in pages and to avoid accessing the database to check when the web page was last modified.
  In the event that the system continues to increase its load, it will send the Void Agent. Thus, although it cannot detect a modification in a web page, we ensure that the web browser will not send a data request to the WCD system.
- WCD Data Interface: this interface receives information of the modifications performed in a web page and sends this to the WCD Change Detector by means of the Queue Server. The Queue Server is responsible for storing this information in the corresponding queue for further processing. So, in load peaks, the system can return a success code to the client almost instantaneously and perform the processing of the data when the system load decreases. Then, the system cleans the remaining contents of queues, only analysing the most recent data for each page. After this, the system begins to process the data of the queues and updates the information stored in the WCD Repository.

Apart from the elements discussed above, and due to the high number of requests that the WCD system receives, we have considered the possibility of distributing the system. For this, we have created a new element named Load Balancer, which is responsible for routing each new request to the process with the least load. In this way, the system is balanced and all the processes of the system are working, avoiding wasting resources.

Figure 3 shows the full architecture of the WCD system. The Load Balancer, the Queue Server and the Web Traffic Controller reduce the system load, managing all requests received and helping to prevent DDoS attacks and remote data modification.

4.4. WCD Advantages and Disadvantages

In this section we discuss the set of advantages and disadvantages of using the WCD system in a search engine. Among the achieved improvements we want to highlight the followings:

- Fast change detection: the WCD system will be able to know in a few minutes when a web page has changed versus current search engines that guess when a web page may have changed based on statistical information (and therefore, may fail).
- Distributed processing: part of the processing to decide whether the web page has been modified, that is the “summary” of the web page, is done on the client side, which frees up the search engine, saving valuable resources.
- Low maintenance: from the point of view of the content creator, a reference to the WCD agent must be inserted only once. It does not require access to the root directory nor prior knowledge of the frequency of changes to web pages such as those that must be specified in “robots.txt” and “sitemap.xml” files [30], and more importantly, the user does not depend on the implementation of the Last-Modified header in the web server.

All the advantages discussed above allow, on one hand, to improve the quality of content stored in the repository, since it will be more up-to-date, and on the other hand, greatly reduce resource usage in the crawling process.

However, we are aware that the system has disadvantages, among which are:

- A minimum webmaster cooperation is required to include a reference to the WCD agent.
- Undue notification of web page modifications, that is, a client alerts about a change in a web page which has not happened.
- Scenarios with a high number of requests in a short period of time meaning that all the requests cannot be processed in real time. These situations can be due to heavy use of the system, or by DDoS attack.

Regarding the first one, some other web applications (e.g. Google Analytics) provide some benefits to the webmasters to promote their cooperation. We follow the same policy, as webmasters benefit from the WCD system by easily keeping the search engines updated with their last web page changes with no effort. However, some other solutions could also be used such as including the WCD agent as an extension or plug-in for the main web browsers or by developing a toolbar.

Regarding the last two, we have proposed a series of elements and protocols for action to mitigate the risk of them happening and if they were to happen that their consequences be minor (see section 4.3).

5. Experimental Results

This section describes and discusses the experiments performed to verify the effectiveness and efficiency of the WCD system. In order to perform the experiments, we have implemented a prototype of the proposed system.

The WCD agents and their references on the web pages were implemented using JavaScript. JavaScript is a widely used technology in web pages and it provides all the functionalities required by a WCD agent, as defined in section 4.1. It is also important to remark that the technology used for implementing the WCD system is not a key factor for the experimental results.

We have performed two types of experiments:

- Change detection experiments: we have analyzed the ability of the WCD system to detect web pages changes and compared the results obtained with the main search engines. Two different options are considered: a best case and a worst case scenario.
- Performance experiments: we have analyzed the performance of the WCD system in order to test whether the system can be used in real environments with large volumes of data and user accesses. For this, we have carried out load tests over the system and simulation experiments.

5.1. Change Detection Experiments

Best Case Scenario One of the assumptions behind the WCD system operation is that, after a webmaster modifies a web page, he will immediately load the web page to check that it is correctly displayed. This constitutes our best case scenario.

In order to test this best case scenario, we have created our own web site (i.e. a blog). For 30 days (from May 11th to June 11th, 2013) we added a post every 12 hours (i.e. every 12 hours the content of the site was changed). The objective of this experiment was to compare the results obtained by the crawlers of the main search engines (Google, Yahoo! and Bing) with the results obtained by the WCD system.

The blog was created at Blogspot `http://webchangedetector.blogspot.com.es/`, for several reasons:

- We can customize the home page and include the reference to the WCD agent.
- We have fully reliable information about real changes in the web site.
- We have the site access logs and, therefore, we can check when the crawlers accessed the web site. This is crucial to determine the time that the data for search engines becomes outdated.

In order to accelerate the process of visiting and indexing the web site by the search engines, we have registered the site in the three search engines studied. However, the results for Bing and Yahoo! are not shown because during the month that the experiment was performed these search engines had not indexed the page (in fact, at the moment of writing this article, the site has still not been indexed). Table 3 shows the results obtained by Google and the WCD system.

Table 3. Best case scenario results

	Google	WCD
Outdated Time	61.13%	0.00%
Unnecessary Refresh	3	0.00
Undetected Changes	5	0.00

First of all, the results obtained by Google are coherent with those obtained in the global experiment (see Section 3), although slightly worse in the outdated time and unnecessary refresh. More specifically, the results show that on Google our blog was outdated 61.13% of the time. The number of unnecessary accesses is slightly higher than the global results, probably because the crawler is still learning the change frequency of the web site. On the other hand, Google has missed 5 changes, which is, in fact, a much better result than the result obtained in the previous experiment. This is probably due to the regular changes produced in this experiment (i.e. every 12 hours) versus the random changes that occur in the real web sites analyzed in our first experiment.

Regarding the WCD system, all the changes were detected immediately. After each new post, the blog administrator loaded the web page (with the WCD agent) and the WCD agent reported the modification to the system. A search engine using the WCD system could achieve a perfect score, with the index always up-to-date, without missing any changes (0 undetected changes) and without unnecessary visits to the web page (0 unnecessary refresh).

Table 4. User accesses per day for the monitored web sites

	PR 0	PR 1	PR 2	**Low PR**	PR 3	PR 4	PR 5	**High PR**
User accesses	51.10	60.85	89.39	**67.11**	571.43	1332.84	733.85	**879.37**

Worst Case Scenario In this section we explore the behavior of the WCD system in a worst case scenario. In this case, we assume that changes will occur, but the webmaster will not load the page to check the changes. Therefore, we depend only on the web users that will visit the page. The objective is to measure when the changes are detected and compare the results obtained with the main search engines.

Ideally, to test this scenario, we would need a set of web pages (i.e. one hundred) with different PageRanks, that were monitored by the WCD system and indexed by the main search engines. At the same time, we would need to log every user access and every change in each web page. Unfortunately, this is beyond our means and, therefore, we decided to use simulation to measure the WCD system results.

In Section 3 we presented the results for change detection of Google, Yahoo! and Bing on 150 web sites. This will constitute our baseline.

To simulate the behavior of the WCD system we need two main variables:

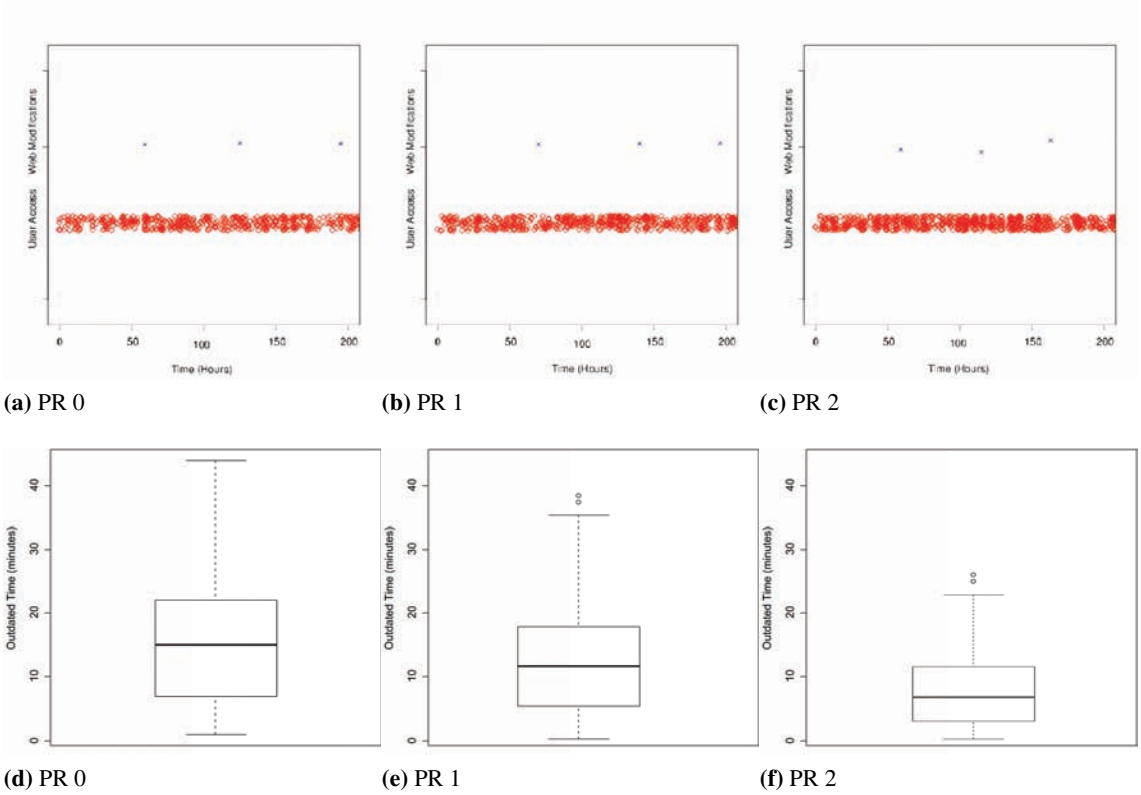
- Web page changes.
- Web page user accesses.

Regarding the former, we have studied the time elapsed between two consecutive changes on each of the 150 monitored web pages. The hypothesis was that changes should follow a Poisson distribution, as described in [9] and [10]. A Kolmogorov-Smirnov one sample test was done for each page and the results conclude that for every one of the 150 tests, it would fit a Poisson distribution with a p-value below 0.005. The mean for the Poisson distribution is estimated from the average change rate obtained for each PageRank (see Table 1).

The latter is also simulated using a Poisson distribution, following the works by Mikael Andersson *et al.* [3] and by Gündüz and Özsu [18], that show that user accesses follow a Poisson distribution, under different circumstances.

To calculate the average user accesses per day, we extracted the statistics from the web site SeeTheStats for each monitored web site (in fact, this was the main reason to randomly select the web sites from SeeTheStats). Table 4 shows the results. These values will be used as an estimation of the mean for each Poisson distribution.

Using both stochastic variables (user accesses and page modifications) we have simulated the behavior of the WCD system for different web pages with PageRank ranging from 0 to 5, during 200 hours (approximately one week). In each case, five different simulations were performed and the results showed correspond to the average values. The Figure 4 includes results of domains with PageRank between 0 and 2, and Figure 5 the results obtained in domains with PageRank between 3 and 5. Each of the figures contains, on one hand, the user accesses according to a particular PageRank (red squares) and the web modifications (blue crosses), and on the other hand, the time that our system would contain outdated data in accordance with the simulation performed.



(a) PR 0 **(b)** PR 1 **(c)** PR 2

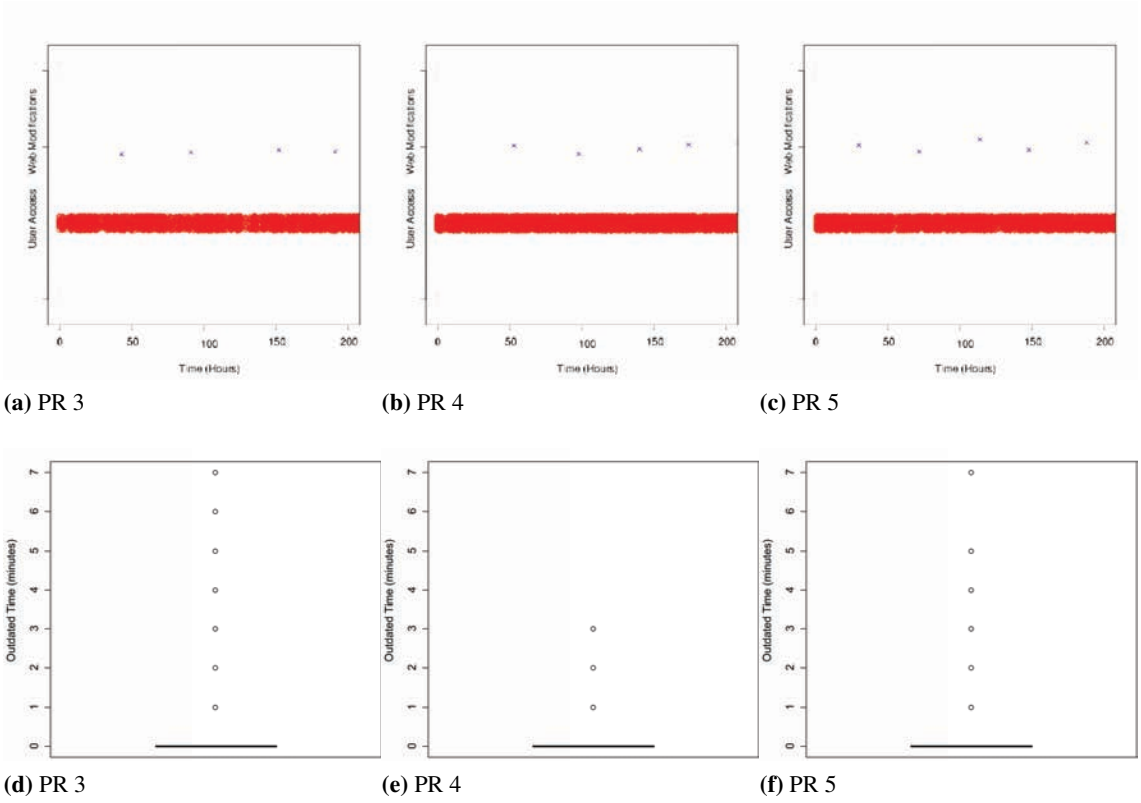
(d) PR 0 **(e)** PR 1 **(f)** PR 2

Fig. 4. WCD results for web sites with PageRank between 0 and 2

Figure 4 and Figure 5 show the box and whisker plot for the time required by the WCD system to detect a web page modification, according to the different PageRank studied. As expected, in the pages with lower PageRank the WCD system requires more time to detect a change. For example, on PageRank 0 the system needs 15.08 minutes on average to detect a change, meanwhile on PageRank 5 this time is below one minute. The maximum time to detect a change in web pages with PageRank 0 is 45 minutes, while on PageRank 5 it is only 7 minutes. In fact, the slowest time to detect a change was produced on a page with PageRank 0.

In order to compare the results obtained by the WCD system with the search engines studied, we have calculated the time that each crawler needs to detect a web modification. These results are shown in Table 5 and were extracted from the data obtained in Section 3.

Analyzing the results on web pages with low PageRank, the search engines take one day and a quarter (Google more than one day and a half), to detect a modification. However, the WCD system will need, on average, only 12 minutes. This is two orders of magnitude smaller. The results for the web pages with higher PageRank show that Google, Yahoo! and Bing will take roughly one day to detect a change. In this case, Google is a



(a) PR 3 **(b)** PR 4 **(c)** PR 5

(d) PR 3 **(e)** PR 4 **(f)** PR 5

Fig. 5. WCD results for web sites with PageRank between 3 and 5

little faster than the other two, as expected from the results obtained from Section 3. Again, the WCD system is able to largely improve these results. In this case, the average time to detect a change by the WCD system is roughly one minute, which is three orders of magnitude smaller.

These results show that the pull model used by current search engines performs poorly when detecting web page changes, because the crawler must guess when the modification will occur. On the contrary, the WCD system is based on a push model, relying on the web users' accesses to collaboratively detect and notify the changes to the search engine. The results obtained by the WCD system, even in the worst case scenario, show that web page changes can be detected, on average, in just a few minutes.

5.2. Performance and Scalability Experiments

In this section we analyse the system from the point of view of its performance and scalability.

The first step is to estimate the size of the indexable Web and the percentage of the Web for each PageRank. For that we have used the study presented by Gulli and Signorini

Table 5. Average change detection time for the search engines and the WCD. The results have been calculated from the data of Table 1 by substracting the frecuency of web page changes from the frecuency of access for each search engine. Time units are hours

	PR 0	PR 1	PR 2	**Low PR**	PR 3	PR 4	PR 5	**High PR**
Google	38.15h	45.69h	38.73h	**40.75h**	26.64h	15.75h	28.99h	**23.79h**
Yahoo!	45.22h	30.75h	26.2h	**34.07h**	32.59h	24.90h	22.81h	**26.77h**
Bing	32.32h	30.77h	32.82h	**31.81h**	32.59h	22.64h	23.41h	**26.22h**
WCD	0.22h	0.15h	0.25h	**0.21h**	0.04h	0.02h	0.04h	**0.03h**

[17], where the authors estimate that the indexable Web contains aproximately 11.5 billion web pages. The percentages of each PageRank on the Web have been obtained analysing the PageRank of 1,000 randomly selected web pages of SeeTheStats[7], the same web site used to obtain the number of user access per web page. Finally, based on the number of user accesses showed on Table 4, we can estimate the total requests received by the WCD system. In Table 6 we show the number of web pages for each PageRank, and the total web pages accesses on a daily basis.

It is interesting to note that as the PageRank increases the number of user visits also increases, although the number of web pages is reduced. In total, we estimate that the WCD system will have to manage arround 2.12 trillion accesses per day.

We have considered two WCD agents in the system operation: the Digester Agent and the Void Agent. We define t (digester threshold) as the minimum time between two Digester Agents are submitted to a web page p. If the time spent since the last time the Digester Agent was sent is greater or equal to t, the system will send the Digester Agent. Otherwise it will send the Void Agent. So, the lower t the more updated will be the WCD Repository and, the more resources will be required, and viceversa for a higher threshold. We have followed this approach in order to analyse the WCD performance improvement by reducing the accesses and insertions on the WCD Repository.

Table 6. Number of web pages and user accesses according to the PageRank

	PR 0	PR 1	PR 2	**Low PR**	PR 3	PR 4	PR 5	**High PR**
% of Web	45.40%	21.60%	16.60%	83.6%	9.00%	4.20%	3.20%	16.4%
Web pages	5E+09	2.5E+09	1.9E+09	9.4E+09	1E+09	4.8E+08	3.7E+08	18.5E+08
User Acceses	3E+11	1.5E+11	1.7E+11	6.2E+11	5.9E+11	6.4E+11	2.7E+11	15E+11

In order to estimate how many requests can be processed by one WCD server, we have deployed the WCD server on a PC with Intel Core i5 6300 at 2.80GHz, 8GB RAM and Ubuntu 12.04 LTS. We have used the following thresholds, in hours, for $t = 24, 20, 15, 10, 5, 1\ and\ 0$. The case of $t = 0$ represents when the WCD server always

[7] `http://www.seethestats.com/`

sends the Digester Agent and therefore the WCD Repository is as updated as possible. In this case, the WCD server can process 65,160 request per minute. However, if the system takes 20 or 24 hours to send a Digester Agent, the throughput raises to 69,000 and 72,000, respectively.

Finally, we can measure how many WCD servers we need to detect any web page modification on the whole Web. Moreover, as it may be expected that the webmaster would only be interested in keeping updated the home page or the most relevant pages, we have analysed the performance for an usage of 25%, 50% and 100% of the Web. The number of WCD servers used on each case is shown in Figure 6.

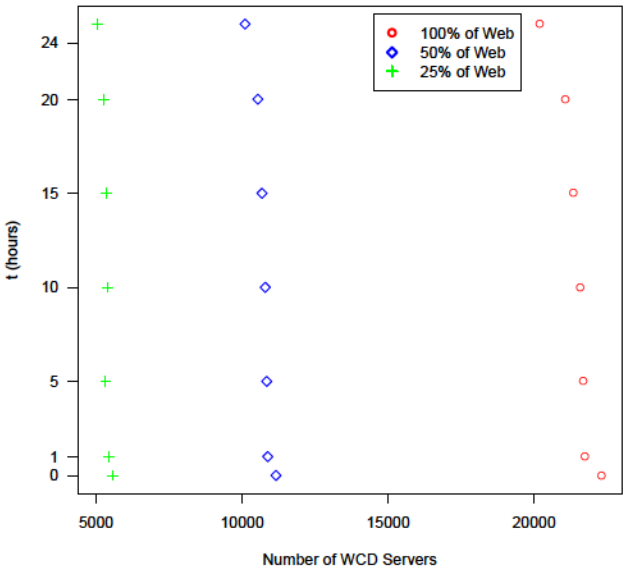


Fig. 6. Relation between number of WCD servers and time t required to detect web page changes

The results show that, to have the repositories as updated as possible, that is $t = 0$ (the WCD servers always send the Digester Agent), the system needs 22,315 WCD servers for a 100% of the Web, or just 5,578 whether the system is used by the 25% of the Web. As t is increased, the number of WCD servers is slightly decreased. For example, for the whole Web, with $t = 10$, 21,576 WCD servers are required, and 20,195 WCD servers with $t = 24$. Analysing the points of the figure, we can see that the main gap appears between $t = 0$ and $t = 1$. So, a good compromise between the detection time and the number of resources would be to set the digester threshold t to 1 hour.

Finally, we want to highlight that currently Google is using around 2 millions of servers [8]. The WCD system would require only 1.11% of these servers and will detect any change in any web page of the Web in just a few minutes (on average).

An interesting future work would be to use a different value of t according to the relevance of a web page, that is according with its PageRank. In this way, the WCD system would have a small t for web pages with high PageRank, and therefore, these pages will be more updated in the WCD Repository. However, this approach may have similar disadvantages as current crawling systems, and so, a deeper analysis is required.

6. Conclusions

This paper tackles the problem of deciding the right moment to recrawl a web page. Current search engines are based on a pull model, which provides poor performance in the detection of web page changes. In our experiments, we have observed that the main search engines' (Google, Yahoo! and Bing) indexes are outdated more than half of the time.

On the other hand, we present the WCD system that is based on a push model and is capable of detecting when a web page changes much faster than the main search engines. The main idea behind the WCD system is that, after any webmaster modifies a web page, he will immediately load the page to check that everything is correctly displayed. This constitutes our best case scenario and our experiments show that, in this case, the WCD system will immediately detect any change on a web page, while current search engines (i.e. Google) will present outdated content to the user 61% of the time. Moreover, our system does not miss any modification on the web page nor pays any unnecessary visits to the site.

In the worst case scenario we assume that a web page may be changed without reloading the page to verify the changes. Therefore, we depend only on the users' accesses. Even in this case, the WCD clearly outperforms the traditional search engines. On pages with low PageRank, the search engines studied needed, on average, one day and a quarter to detect a change, while our system needed only 12 minutes. Even in the slowest case of our experiments, the WCD system required only 45 minutes to detect a change. On pages with higher PageRank, the performance of the WCD system is even better. Current search engines needed approximately one day to detect a change, meanwhile our system was able to detect a change in about one minute (three orders of magnitude smaller).

In another set of experiments we have also tested the performance and scalability of the WCD system. We have considerated different scenarios depending on the time that the WCD system sends the Void Agent or the Digester Agent. The results show that, in order to detect a change on any web page of the whole indexable Web, the WCD system would require 22,315 servers (just 1.11% of the servers used by one of the main search engines).

In summary, we have presented the WCD system, which based on a distributed and collaborative architecture, is able to detect web page changes in just a few minutes.

[8] https://plus.google.com/114250946512808775436/posts/VaQu9sNxJuY

7. Future Work

Future work will focus on the integration of the WCD system in a crawler. We will consider the possibility of using an existing crawler (Nutch [22], Heritrix [26], etc.), or the development of a new crawler based on the WCD system architecture. Also, the design of the WCD system following the Map-Reduce distributed architecture [13] seems interesting and promising.

Another future work regarding the WCD agent will be to improve the summary of the website content. We plan to use a hash that summarizes the contents of the web page [25], instead of creating a MD5 of the different parts of the page.

Finally, we will analyze several improvements to the WCD system, such as improved security from attacks (massive updates of contents) or false content updates.

Acknowledgments. This research was supported by Xunta de Galicia CN2012/211, the Ministry of Education and Science of Spain and FEDER funds of the European Union (Project TIN2009-14203).

References

1. The w3 consortium the document object model. http://www.w3.org/DOM/ (2011), [Online; accessed 18-February-2011]
2. The search engine land. http://searchengineland.com/yahoos-transition-to-bing-organic-results-complete-49228 (2012), [Online; accessed 20-October-2012]
3. Andersson, M., Bengtsson, A., Höst, M., Nyberg, C.: Web server traffic in crisis conditions. In: In In Proceedings of the Swedish National Computer Networking Workshop, SNCNW (2005)
4. Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., Raghavan, S.: Searching the web. ACM Trans. Inter. Tech. 1(1), 2–43 (2001)
5. Baeza-Yates, R., Castillo, C.: La web chilena (2004)
6. Baeza-Yates, R., Castillo, C., Efthimiadis, E.N.: Characterization of national web domains. ACM Trans. Internet Technol. 7 (May 2007)
7. Baeza-Yates, R., Saint-Jean, F., Castillo, C.: Web structure, dynamics and page quality. In: Laender, A., Oliveira, A. (eds.) String Processing and Information Retrieval, Lecture Notes in Computer Science, vol. 2476, pp. 453–461. Springer Berlin / Heidelberg (2002)
8. Bergman, M.K.: The deep web: Surfacing hidden value (2000)
9. Brewington, B., Cybenko, G.: How dynamic is the web? pp. 257–276 (2000)
10. Cho, J., Garcia-Molina, H.: The evolution of the web and implications for an incremental crawler. In: Proceedings of the 26th International Conference on Very Large Data Bases. pp. 200–209. VLDB '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
11. Cho, J., Garcia-Molina, H.: Synchronizing a database to improve freshness. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. pp. 117–128. SIGMOD '00, ACM, New York, NY, USA (2000)
12. Cho, J., Garcia-Molina, H.: Estimating frequency of change. ACM Trans. Internet Technol. 3, 256–290 (August 2003)
13. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51, 107–113 (January 2008)
14. Donghua Pan, Shaogang Qiu, D.Y.: Web page content extraction method based on link density and statistic. In: Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference. pp. 1–4 (2008)

15. Fetterly, D., Manasse, M., Najork, M.: Spam, damn spam, and statistics: using statistical analysis to locate spam web pages. In: Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004. pp. 1–6. WebDB '04, ACM, New York, NY, USA (2004)
16. Fetterly, D., Manasse, M., Najork, M., Wiener, J.: A large-scale study of the evolution of web pages. In: Proceedings of the 12th international conference on World Wide Web. pp. 669–678. WWW '03, ACM, New York, NY, USA (2003)
17. Gulli, A., Signorini, A.: The indexable web is more than 11.5 billion pages. In: Special interest tracks and posters of the 14th international conference on World Wide Web. pp. 902–903. WWW '05, ACM, New York, NY, USA (2005)
18. Gündüz, S., Özsu, M.T.: A poisson model for user accesses to web pages. In: ISCIS. pp. 332–339 (2003)
19. Gyongyi, Z., Garcia-Molina, H.: Web spam taxonomy. Technical Report 2004-25, Stanford InfoLab (March 2004)
20. Holdener, A.T.: Ajax: The Definitive Guide. O'Reilly Media (2008)
21. Kharazmi, S., Nejad, A.F., Abolhassani, H.: Freshness of Web search engines: Improving performance of Web search engines using data mining techniques. In: Internet Technology and Secured Transactions, 2009. ICITST 2009. International Conference for. pp. 1–7 (Nov 2009)
22. Khare, R., Cutting, D.: Nutch: A exible and scalable open-source web search engine. Tech. rep. (2004)
23. Kumar, J.P., Govindarajulu, P.: Duplicate and near duplicate documents detection: A review. European Journal of Scientific Research 32, 514–527 (2009)
24. Lewandowski, D.: A three-year study on the freshness of web search engine databases. J. Inf. Sci. 34, 817–831 (December 2008)
25. Manku, G.S., Jain, A., Das Sarma, A.: Detecting near-duplicates for web crawling. In: Proceedings of the 16th international conference on World Wide Web. pp. 141–150. WWW '07, ACM, New York, NY, USA (2007)
26. Mohr, G., Kimpton, M., Stack, M., Ranitovic, I.: Introduction to heritrix, an archival quality web crawler. In: 4th International Web Archiving Workshop (IWAW04) (2004)
27. Olston, C., Pandey, S.: Recrawl scheduling based on information longevity. In: Proceedings of the 17th international conference on World Wide Web. pp. 437–446. WWW '08, ACM, New York, NY, USA (2008)
28. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: Proceedings of the 27th International Conference on Very Large Data Bases. pp. 129–138. VLDB '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
29. Ricardo Baeza-Yates, C.C., Lopez, V.: Characteristics of the web of spain (2005)
30. Sun, Y., Zhuang, Z., Giles, C.L.: A large-scale study of robots.txt. In: In WWW '07: Proceedings of the 16th international conference on World Wide Web. pp. 1123–1124. ACM Press (2007)

V´ctor M. Prieto is currently the CEO of his own business in web information services. He received his Bachelor's Degree in Computer Engineering from the Universidade da Coruña in 2007 and a Ph.D. Degree in Computer Science in Department of Information and Comunications Technologies at the same university in 2013. His main research fields are web crawling, Hidden Web and Web Spam. He has also published several papers in international journals and has participated in multiple international conferences.

Manuel Álvarez is an Associate Professor in the Department of Information and Communication Technologies, at the Universidade da Coruña (Spain). He received his

Bachelor's Degree in Computer Engineering from the Universidade da Coruña in 1999 and a Ph.D. Degree in Computer Science from the same University in 2007. His research interests are related to information retrieval and data integration. Manuel has managed several projects at national and regional level in the field of data integration and Hidden Web accessing. He has also published several papers in international journals and has participated in multiple international conferences.

Fidel Cacheda is an Associate Professor in the Department of Information and Communications Technologies at the Universidade da Coruña (Spain). He received his Ph.D. and B.S. degrees in Computer Science from the University of A Coruña, in 2002 and 1996, respectively. He has been involved in several research projects related to Web information retrieval and multimedia real time systems. His research interests include Web information retrieval and distributed architectures in information retrieval. He has published several papers in international journals and has participated in multiple international conferences.

Victor Carneiro received his Ph.D. and B.S. degree in Computer Science from University of A Coruña, A Coruña, Spain, in 1998 and 1993, respectively. He has been an associate professor of the Department of Information and Communication Technologies, University of A Coruña, Spain, since 1995. He has participated in a lot of research projects and professional experiences related with network management, distributed systems and information retrieval over Internet. Nowadays he is working in technologies oriented to recommender systems based on collaborative filtering techniques.

Received: November 20, 2013; Accepted: September 20, 2014.