How Crawlers Aid Regression Testing in Web Applications: The State of the Art

Shikha Raina
Computer Science and Engineering
Amity University
Noida, India – 201301

Arun Prakash Agarwal
Computer Science and Engineering
Amity University
Noida, India - 201301

ABSTRACT
In today's world web applications have increasingly become very popular, and with the same speed they are been upgraded frequently. This poses a big challenge for web application testing. To ensure web application quality, we must perform adequate testing for the new features as well as regression testing the web application in each of iteration. This requires us to systematically identify/locate changes introduced in the new iteration. This paper surveys various tools which identify/locate new changes, which greatly facilitates web application testing in short release cycles. Also we will describe what are web crawler's and show how crawlers can aid in regression testing of web applications.

Keywords
Regression Testing, Web application testing, Web Crawlers.

1. INTRODUCTION
Dynamic Web Applications have become a popular business application delivery model [1], shifting more and more software applications to the web. However, its rapid release cycles, typically 2-3 weeks, create a big challenge for web application quality assurance. Under high release pressure, developers can easily make mistakes during implementation [1]. For example, a PHP web page developer adds a new field into a search form, which may need to insert extra form validation both at Client Side(i.e. through JavaScript) and at Server Side also. In this case, if the front-end developer forgets to add any form validation and also update the database schema at the back-end, a system failure, typically in a form of MySQL exception, would happen. With short release cycles, good documentation is luxury. Consequently, it is difficult for quality assurance engineers to systematically identify/locate changes based on existing documents. To efficiently test web applications, automated regression testing [2] has been introduced to re-test existing features. This helps to greatly reduce regression testing costs. Automated regression testing tools have been widely used in software industry, e.g., HtmlUnit [3] and Selenium [4]. However, automated regression testing tools were not developed to systematically identify and test changes introduced in a new iteration, where defects can be easily introduced. Therefore, adequate testing of newly introduced changes in each of iteration can be a critical step to ensure web application quality. The problem of adequately testing new changes, i.e., new features and their impacts, can be divided into two sub-problems: (1) how to quickly and accurately identify/locate new changes; and (2) how to effectively generate and run tests for the new changes. The second sub-problem has been well studied [2, 5]. However, little work has been reported on the first sub-problem [1], this paper presents two tools, named Zoomer and ReWeb/TestWeb that have been implemented to automatically identify newly introduced changes in a new iteration.

We will explain the usefulness of the tools Zoomer and ReWeb/TestWeb, built by researchers to aid the regression testing of web applications. The paper is organized as follows: Section 2 presents the background of web applications and its architecture. In Section 3, we will discuss what a web crawler is and give a brief intro of various crawling strategies, their use and limitations. The tools developed to support regression testing in web applications are described in Section 4. Finally, Section 5 concludes this paper and discusses future work.

2. BACKGROUND
The aim of this section is to give a brief insight into the web application background. The chapter is organized as follows: Section 2.1 tries to answer the question what Web applications are and gives an overview of the overall Web application architecture.

2.1 What is a Web application?
A Web application is a special case of application, designed to be executed in a Web-based environment. More precisely a (dynamic) Web application is a mix of programs that dynamically generate hyper-documents (dynamic Web pages) in response to some input from the user, and static hyper-documents [6]. A (static) Web site is simply a collection of static hyper-documents (static Web pages). While a Web application is basically a program running on a Web server and a set of fixed Web pages, there is much more to be considered in the activities of regression testing [6]. The behavior and the quality of a Web application depend on all its components. Web applications contains many components that are linked together to deliver the desired usability of the application [6].

The fundamental elements of Web applications are:

Client/Server model: a browser (client) sends a request asking for a Web page over a network (Internet, via the protocol HTTP) to a Web server (server), which returns the requested page as response of the request [6]. These web pages that are being requested for can be either static pages or dynamic pages. The dynamic pages are computed on run time by web application depending upon the user input at that time. And the content of a static page is fixed and mainly stored in a cache or some directory on the server. The programs that generate dynamic pages at run-time (called server programs), as for example CGI (Common Gateway Interface) scripts and servlets, run on the application server and can use information stored in databases and other resources [6].

HTML language: Web pages are written principally in HTML language (HTML stands for the HyperText Markup Language). A Web page consists of text, which is an

unstructured sequence of characters, and HTML elements. Elements are enclosed within a start-tag and an end-tag notation. This is an HTML element: < p> my paragraph starts here. < /p> Where the HTML element starts with a start tag ( ), the content of the HTML element is "My paragraph starts here" and the HTML element ends with an end tag ( ). The power of HTML is in its links to other HTML resources. In specific tags of an HTML document (for example the anchor <a> tag), the URL (Uniform Resource Locator) of another HTML document can be accessed. If the user clicks the mouse button on the text of an anchor element, the browser automatically retrieves a new Web page and then displays it. The language HTML is defined precisely in [6].

Client/server interactions: The HTML code can activate the execution of a server program by means of a SUBMIT input within an HTML element of type FORM or anchor. Data flows from a server program to the HTML code are achieved by embedding values of variables inside the HTML code, as the values of the attributes of some HTML elements. In the opposite direction, the basic mechanism for data value propagation is by means of form parameters. Hidden parameters are constant values that are just transmitted to the server (possibly recording the values of some previous computation), while non hidden input parameters are gathered from the user. Parameter passing is strictly by value and the invocation of the server program is a control transfer without return [6]. Server programs can exploit persistent storage devices (such as databases) to record values and to retrieve data necessary for the construction of the HTML page. Moreover, session specific data (session variables) can be stored at the server side and maintained across successive executions. Cookies act similarly to session variables, with the only difference of being stored at the client side instead of the server. Such data are used to identify the ongoing interaction and to record data that need to survive past the end of execution of server programs [6].

Finally, there are several server side languages that are available for the construction of Web applications (e.g., PHP, Java, Perl, VBscript, etc.). The same is true for the client side code (e.g., Java, Javascript, jQuery etc.).

3. OVERVIEW OF WEB CRAWLER

WebCrawler is a Web service that assists users in their navigation by automating the task of hyperlink traversal, creating a repository of the searched web pages, and fulfilling user's request from the repository. Conceptually, WebCrawler is a node in the Web graph that contains links to many sites on the net, shortening the path between users and their destinations. Such a simplification of the Web experience is important for several reasons:

First, WebCrawler saves users time when they search instead of trying to guess at a path of links from page to page. Often, a user will see no obvious connection between the page he is viewing and the page he seeks. For example, he may be viewing a page on one topic and desire a page on a completely different topic, one that is not linked from his current location. In such cases, by jumping to WebCrawler — either using its address or a button on the browser — the searcher can easily locate his destination page. Such time savings is especially important given the increase in the size and scope of the Web: between 1994 and 2000, the Web grew in size by four orders of magnitude [7].

Second, WebCrawler's simplification of the Web experience makes the Web a more friendly and useful tool. Navigating the Web by using keyword searches is often more intuitive than trying to use a Uniform Resource Locator (URL) to identify a Web page directly. If users have a good experience, they are more likely to continue to use the Web, and such repeat usage will continue to fuel the growth of the medium. Arguably, search engines like WebCrawler have contributed to the continued simplicity and growth of the Web.

Finally, WebCrawler is useful because it can provide some context for a searcher's particular query: by issuing a well-formed query, a searcher can find the breadth of information about that particular topic and can use that information to further refine his goal. Searchers frequently issue a broad query which they refine as they learn more about their intended subject.

3.1 Various Crawling Strategies

3.1.1 Centralized Crawling

In [8, 9] researchers have tried to solve the problem of web crawling by focusing on AJAX based applications. In [10, 11] they have focused on web crawling for the purpose of search and indexing. In [12], the aim is to make RIAs accessible to search engines that are not AJAX-friendly. In [13] the focus is on regression testing of AJAX applications, whereas [14] is concerned with security testing of web widget interactions, [15] focuses on invariant-based testing. However, except for the work done in [8, 9] most of the research is concerned with their ability to crawl RIAs and not the actual efficiency of crawling. Crawling RIAs in its naive form seems to favor the standard Breadth-First and Depth-First strategies, which have been used in most of the published research with some modifications. One of the earliest attempts for an AJAX crawling algorithm and optimization is presented in [11]. The authors proposed an AJAX crawler that crawls the application based on user events and builds a model of the application. The application is modeled using transition graphs which contain all the application entities (states, events and transitions). The crawler uses the Breadth-First search strategy to trigger all the events present in the page. If the DOM of the page changes then a new state and corresponding transition is added to the transition graph. After a new state is reached, the crawler uses a reset to go back to the initial state and invoke the next event in the initial state. Once all the events in the initial state have been explored, the crawler explores in a similar fashion the discovered states in the order they are discovered. In addition to the crawling strategy, the authors also proposed few optimizations to improve the efficiency of the crawling process. They suggested caching of JavaScript function execution results to save expensive server calls. If the same JavaScript function is invoked again along with the same parameters, then the cached results are used instead of executing the function again. In [10], the authors introduced an AJAX aware search engine for indexing the contents of RIAs. Similar to traditional search engines, it contains a crawler, indexer and query processor, but the components are adapted to handle RIAs. The AJAX crawler has the role of identifying events in the application states. The crawler starts with identifying and executing events on the first page. The crawler uses a standard Breadth-First search. The crawler identifies a new state if an event execution generated a new DOM tree and the content of the DOM is different from already discovered states. The result of the crawling process is maintained in a special application model which is annotated with new information as the crawling proceeds. The authors recommended the exploration of a limited number of different events and different states or having a maximum limit on the depth of the crawl. The other components of the

search engine, such as the indexer, read the information from the application model discovered by the crawler.

3.1.2 Distributed Crawling

Due to the large size of the web, it is often the case that crawlers use several nodes (i.e. computers) to crawl the web simultaneously. Distributed crawling of the web has been extensively described in the literature [22]. [16] classifies distributed crawlers based on their work assignment method into three classes: Independent Assignment: Different crawlers start from different URLs and crawl the web independently. This approach may lead to overlap and duplication of work. Dynamic Assignment: This approach is based on one or more units that keep track of discovered and executed tasks. Upon discovering a task, the node will inform these units and if it is a new task, the unit will add it to the task queue. Nodes then ask the unit for workload, and the unit assigns tasks to the probing nodes [17]. The first prototype of Google roughly followed this architecture: A centralized unit, called URLserver, stores the list of URLs and orders a set of slave nodes to download each URL. All downloaded pages are stored in a unit, called Storeserver. The retrieved pages are then indexed distributive-ly. Both the downloading and indexing tasks requires centralized units of coordination [18]. Static Assignment: In this approach a set of homogeneous workers are allocated unique IDs. The mapping function maps each task to one of the assigned IDs. Upon encountering a task the crawler examines the task and decides whether the task falls under its jurisdiction or belongs to another node. In the first case, the node takes care of the task autonomously. Otherwise, the node will inform the node responsible for the task [16]. Different proposals suggest different matrices and algorithms to derive the mapping function. In [19] the distribution of the task of crawling of the different URLs is performed by hashing the URL (either only the host-name part, or the entire URL) and distributing the resulting hash values to the different crawlers, for instance, using the distributed hash table (DHT) of a peer-to-peer system. [20] also includes the geographic information about the crawlers and the searched servers into the task distribution algorithm in order to allocate a crawler that is geographically close to the server to be crawled. Ubi-Crawler [21] uses the so-called consistent hashing approach to allocate the tasks to the different crawlers in such a way that there are only minimal changes when crawlers are reconstructed after some time period. This approach among the web crawlers is used to obtain better error tolerance.

4. TOOLS USING WEB CRAWLER

4.1 Zoomer Overview

Zoomer was implemented using Java. It analyzed the web pages in a web application to identify/locate new changes. To retrieve web pages from a web application, it integrated Tansuo [1], a tool to explore web pages in web applications (Web Crawler). It also integrated HtmlUnit [1], a tool for constructing the HTML DOM tree representation for a web page. Firstly, it retrieved web pages from a new iteration. Secondly, for each web page, Zoomer constructed its HTML DOM tree representation with XPaths [1]. Thirdly, Zoomer compared HTML DOM tree representations in the new iteration with saved HTML DOM tree representations in the previous iteration. This comparison worked will capture changes introduced in the new iteration, e.g., element changes and element properties changes. In the following, we will introduce its architecture, workflow and function features.

4.2 Zoomer Architecture

As shown in Fig 1, Zoomer consists of six components: Driver, Tansuo, Parser, Repository, Comparator, and Presenter [1].

4.2.1 Driver

Being the main component in this tool, it was responsible for coordinating other components to identify/locate changes introduced in a new iteration. Driver decides which component should be used at each step [1].

4.2.2 Tansuo

This component was responsible for exploring the web pages in a web application. It worked in a recursive process. For example, it retrieves the home page, by following the URL provided by a user. If there were links and forms in a web page, Tansuo will handle them, e.g., by clicking links or submitting forms, to reach more web pages. This process was repeated until the whole web application has been explored [1].

4.2.3 Parser

This component was responsible for parsing web pages. Parser obtains the HTML DOM tree of a web page through the HtmlUnit library [4]. Then, it traversed the generated HTML DOM tree in a depth-first-search manner. When a new element was encountered, it constructed an XPath [1] for the new element and saveed this new XPath for future comparison [1].

4.2.4 Repository

This component was responsible for saving the elements and their properties. An element was identified by its web page URL and its XPath in the web page. Meanwhile, it also saved comparison results. All the information was saved in a set of files that was accessed by Driver in the future [1].

4.2.5 Comparator

This component was responsible for comparing web pages and their elements in two iterations. Comparator received elements and their property information from Driver and compared them to identify/locate the changes. In other words, for each element in the new iteration, Comparator checked whether it existed in the previous iteration. For each element in the previous iteration, Comparator also checked whether it still existed in the new iteration [1].

4.2.6 Presenter

This component was responsible for presenting comparison results to users. It used JFreeChart [1] to generate pie charts that show change percentage of web pages, elements and element properties. Presenter also inserted links of changed web pages, elements, and properties into comparison result web pages, so that users could easily navigate to the changed web pages and elements that they were interested in [1].

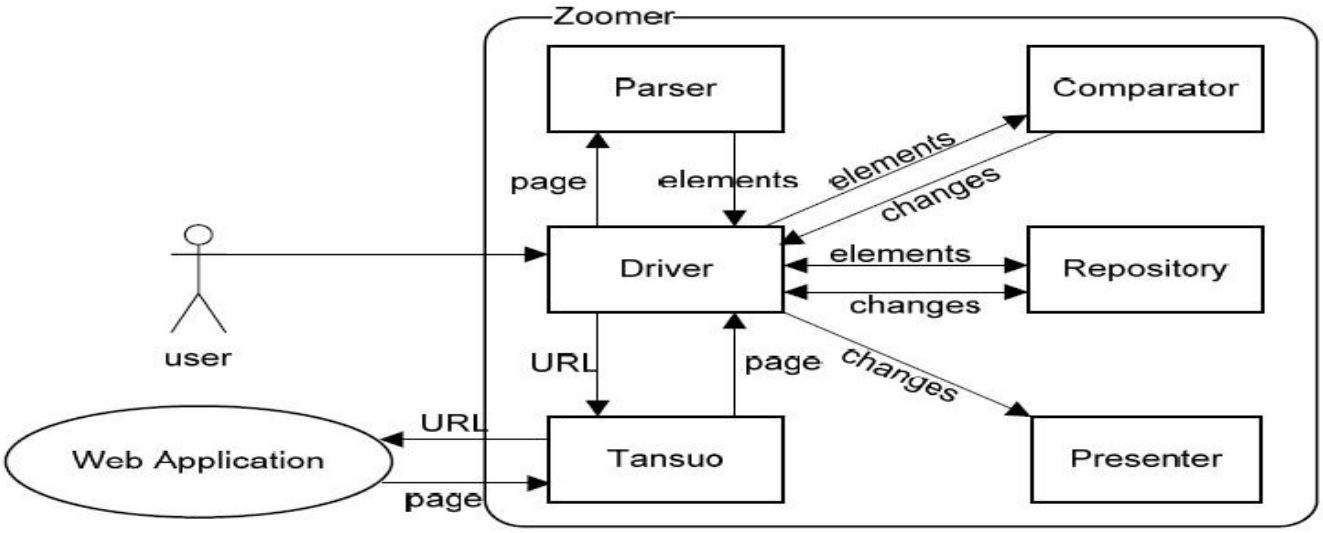

Fig 1: Zoomer's Architecture

4.3 ReWeb and TestWeb Overview

The two tools ReWeb and TestWeb that were developed to support analysis and testing of Web applications. ReWeb was used to download and analyzes the pages of a Web application with the purpose of building a UML model of it, in accordance with the Meta Model. TestWeb generated and executed a set of test cases for a Web application whose model was computed by ReWeb [23]. The whole process was semiautomatic, and the interventions of the user are indicated within diamonds in Fig 2.

4.4 ReWeb Architecture

The ReWeb tool consisted of three modules: a Spider, an Analyzer and a Viewer [23].

The Spider downloads all pages of a target web site starting from a given URL. Each page found within the site host was downloaded and marked with the date of downloading. The HTML documents outside the web site host were not considered. The pages of a site were obtained by sending the associated requests to the Web server. The result of such requests was always an HTML page, so it was not possible to discriminate between dynamic and static pages [23].

The Analyzer used the UML model of the web site to perform several analyses [23], some of which were exploited during static verification.

The Viewer provided a Graphical User Interface (GUI) to display the Web application model as well as the output of the static analyses. The graphical interface supported a rich set of navigation and query facilities including zoom, search, focus and HTML code display. Among the available views, the history view showed the structure of the site over time, the system view represented the organization of pages into directories. The data flow view was used to display the read/write accesses of pages to variables. They used incoming/outgoing edges linking pages to variables respectively [23].

4.5 TestWeb Architecture

TestWeb contained a test case generation engine (Test generator); able to determine the path expression from the model of a Web application, and to generate test cases from it, provided that a test criterion is specified. Generated test cases were sequences of URLs which, once executed, granted the coverage of the selected criterion. Input values in each URL sequence were left empty by the Test generator, and the user had to fill in them, possibly exploiting the techniques traditionally used in black box testing (boundary values, etc.) [23].

TestWeb's Test executor was used to provide the URL request sequence of each test case to the Web server, attaching proper inputs to each form. The output pages produced by the server, marked in the UML model with a non empty use attribute, were stored for further examination. After execution, the test engineer intervened to assess the pass/fail result of each test case [23]. For such evaluation, user opened the output pages on a browser and checked whether the output was correct for each given input. During regression check such user intervention was no longer required, since the oracle (expected output values) were the one produced (and manually checked) in a previous testing iteration [23].

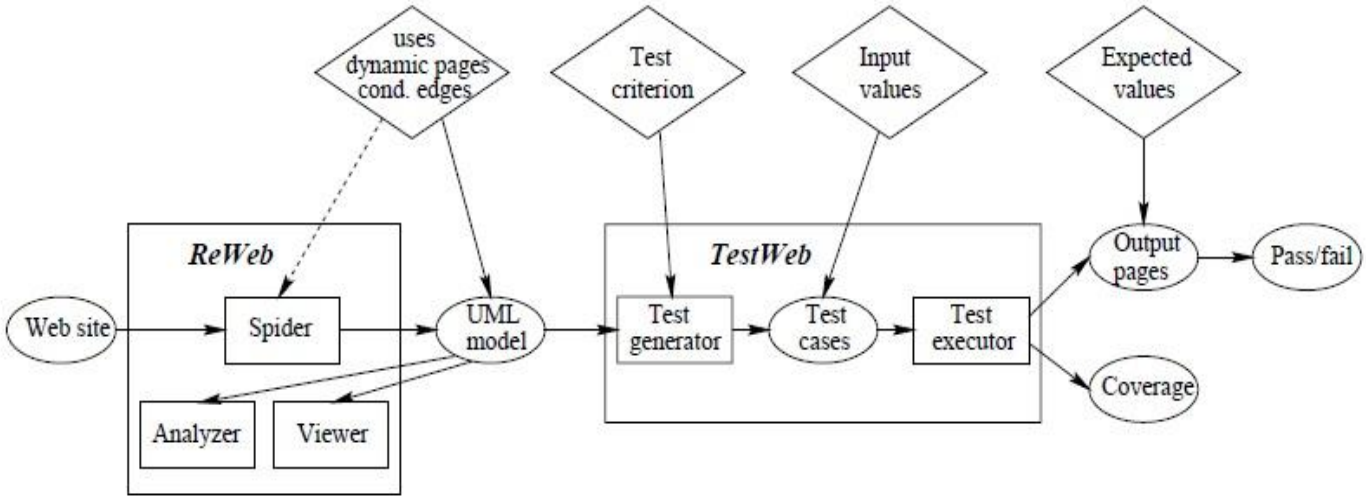

Fig 2: ReWeb and TestWeb Tool Architecture.

5. CONCLUSIONS AND FUTURE WORK

The tools and techniques explained in this paper were successfully applied to several real world Web applications, among which are Wordnet, Amazon and Bookstore web application [1, 23].

We discussed Zoomer's architecture. The previous results have shown that Zoomer can effectively identify/locate changes introduced in a new iteration [1]. ReWeb's views were useful to understand the site organization, both in terms of navigation paths (history view) and of variable usage (data flow view) [23]. TestWeb's generator and executor of test cases were exploited to exercise the two sites up to a satisfactory level of coverage. An anomalous behavior of Amazon was revealed during the testing activity. It was highlighted by the sequence of operations to be performed during the execution of one of the automatically generated test cases [23].

Our future work will be devoted to the building our own tool written in PHP Language, mainly for PHP based web applications, we will use crawler as an input to tool and provide comparison results of two versions of Web application as an output, which will provide an aid in regression testing of those web applications.

6. REFERENCES

[1] Wenhua Wang and Yu Lei, Zoomer: An Automated Web Application Change Localization Tool, in: Journal of Communication and Computer 9 (2012) 913-919.

[2] V. Kettunen, J. Kasurinen, O. Taipale, K. Smolander, A study on agility and testing processes in software organizations, in: Proceedings of the 2010 International Symposium on Software Testing and Analysis, 2010, pp. 231-240

[3] HtmlUnit, DOI: http://htmlunit.sourceforge.net/.

[4] Selenium, DOI: http://seleniumhq.org.

[5] R. Binder, Testing Object-Oriented Systems, Addison Wesley, 2000.

[6] HTML Working Group. Transitional Document Type Definition, HTML 4.01, W3C Recommendation 24 December 1999.

[7] Matrix Information and Directory Services, Inc. Matrix.Net Home http://www.mids.org/.

[8] K. Benjamin, G. v. Bochmann, M. E. Dincturk, G.-V. Jourdan, and I. V. Onut, "A strategy for efficient crawling of rich internet applications," in Proceedings of the 11th international conference on Web engineering, ICWE'11, 2011.

[9] M. E. Dincturk, S. Choudhary, G. v. Bochmann, , G.V. Jourdan, and I. V. Onut, "A statistical approach for efficient crawling of rich internet applications," in Proceedings of the 12th international conference on Web engineering, ICWE'12, 2012.

[10] C. Duda, G. Frey, D. Kossmann, and C. Zhou, "Ajaxsearch: crawling, indexing and searching web 2.0 applications," Proc. VLDB Endow., vol. 1, pp. 1440–1443, Aug. 2008.

[11] C. Duda, G. Frey, D. Kossmann, R. Matter, and C. Zhou, "Ajax crawl: Making ajax applications searchable," in Proceedings of the 2009 IEEE International Conference on Data Engineering, ICDE '09, pp. 78–89, IEEE Computer Society, 2009.

[12] A. Mesbah, E. Bozdag, and A. v. Deursen, "Crawling ajax by inferring user interface state changes," in Proceedings of the 2008 Eighth International Conference on Web Engineering, ICWE '08, pp. 122–134, IEEE Computer Society, 2008.

[13] D. Roest, A. Mesbah, and A. van Deursen, "Regression testing ajax applications: Coping with dynamism." in ICST, pp. 127–136, IEEE Computer Society, 2010.

[14] C.-P. Bezemer, A. Mesbah, and A. van Deursen, "Automated security testing of web widget interactions," in Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ESEC/FSE '09, 2009.

[15] A. Mesbah and A. van Deursen, "Invariant-based automatic testing of ajax user interfaces," in Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on, pp. 210 –220, may 2009.

[16] J. Cho and H. Garcia-Molina, "Parallel crawlers," in Proceedings of the 11th international conference on World Wide Web, WWW '02, 2002.

[17] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos, "Parallel crawling for online social networks," in Proceedings of the 16th international conference on World Wide Web, WWW '07, 2007.

[18] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in Proceedings of the seventh international conference on World Wide Web 7, WWW7, 1998.

[19] B. T. Loo, L. Owen, and C. S. Krishna murthy, "Distributed web crawling over dhts," 2004.

[20] J. Exposto, J. Macedo, A. Pina, A. Alves, and J. Rufino, "Information networking. towards ubiquitous networking and services," ch. Efficient Partitioning Strategies for Distributed Web Crawling, pp. 544–553, Springer-Verlag, 2008.

[21] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: a scalable fully distributed web crawler," Softw. Pract. Exper., vol. 34, pp. 711–726, July 2004.

[22] C. Olston and M. Najork, "Web crawling," Found. Trends Inf. Retr., vol. 4, pp. 175–246, Mar. 2010.

[23] Filippo Ricca and Paolo Tonella, Analysis and Testing of Web Applications, in: 2001 IEEE.