Intelligent Crawling of Web Applications for Web Archiving

Muhammad Faheem
supervised by Pierre Senellart

Institut Télécom
Télécom ParisTech; CNRS LTCI
Paris, France

{muhammad.faheem, pierre.senellart}@telecom.paristech.fr

ABSTRACT

The steady growth of the World Wide Web raises challenges regarding the preservation of meaningful Web data. Tools used currently by Web archivists blindly crawl and store Web pages found while crawling, disregarding the kind of Web site currently accessed (which leads to suboptimal crawling strategies) and whatever structured content is contained in Web pages (which results in page-level archives whose content is hard to exploit). We focus in this PhD work on the crawling and archiving of publicly accessible Web applications, especially those of the social Web. A Web application is any application that uses Web standards such as HTML and HTTP to publish information on the Web, accessible by Web browsers. Examples include Web forums, social networks, geolocation services, etc. We claim that the best strategy to crawl these applications is to make the Web crawler aware of the kind of application currently processed, allowing it to refine the list of URLs to process, and to annotate the archive with information about the structure of crawled content. We add adaptive characteristics to an archival Web crawler: being able to identify when a Web page belongs to a given Web application and applying the appropriate crawling and content extraction methodology.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia

General Terms

Design, Languages, Performance

Keywords

Web application, archiving, crawling, extraction, XPath

1. PROBLEM

Since the introduction of Web 2.0, the social Web is becoming an important resource for content extraction. There are millions of users who are using the social Web as a medium for publishing information, discussing political issues, sharing videos, posting comments, managing blogs, and also stating their personal opinion in ongoing discussions. Currently the social Web is not only used by ordinary users but is also getting much attention from political leaders. Nowadays it is getting quite common in the USA and the UK to answer parliamentary questions using Twitter. Recently, on 6 July 2011, President Barack Obama became the first US president to use Twitter as a tool for public communication [13]. Thus the social Web is also becoming a part of political campaigns as well as driving the future political agenda. This strengthens the necessity to preserve social Web data.

But archiving Web data from the social Web in an intelligent manner is still an ongoing challenge. Our aim is to deal with this challenge by introducing a new adaptive approach which relies basically on a knowledge base about the different kind of (social) Web applications. A *Web application* is any HTTP-based application that utilizes the Web and Web browser technologies to publish information. We focus in particular on social aspects of the Web, which are heavily based on user-generated content, social interaction, and networking, as can be found for instance in Web forums, blogs, or on Twitter.

To understand how current archiving tools are not fully up to the task of social Web archiving, consider the simplified architecture of a traditional Web crawler (such as Heritrix [23]) depicted in Figure 1. A Web *crawler* (also known as Web *spider* or *robot*) is a computer program that inspects the Web in a methodical manner and retrieves targeted documents. Traditional Web crawlers crawl the Web in a conceptually very simple way. They start from a *seed* list of the URLs to be stored in a queue (e.g., the starting URL may be the homepage of a Web site). Web pages are then fetched from this queue one after the other and links are extracted from these Web pages. If they are in the scope

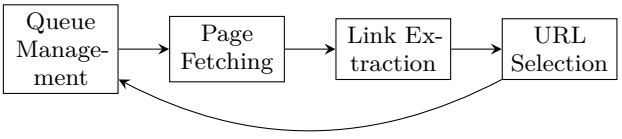

Figure 1: Traditional processing chain of a Web crawler

WWW 2012 Companion, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1230-1/12/04.

of the archiving task, the newly extracted URLs are added to the queue. This process ends after a specified time or when no new interesting URLs can be found.

This approach does not confront the challenges of Web application crawling: the nature of the Web application crawled is not taken into account to decide the crawling strategy or the content to be stored; Web applications with dynamic content (e.g., Web forums, blogs, etc.) may be crawled inefficiently; some content may be missed when it is only accessible through complex crawling actions (AJAX requests, form submissions).

For instance, Web forums hold dynamic characteristics which mean that, for extracting semantic content or for crawling optimization, the nature of these Web forums needs to be understood. Usually the content of a Web forum is stored in a database. When a user makes a request, the response page is automatically generated using a predefined template. When two requests require the same piece of content, the server will return two dynamic pages with same or similar content but with two different URLs. These dynamic pages create redundancy that may be harmful, both because they will require more resources to be crawled, and because the final archive will be of worse quality. Blog systems also contain redundant information: there may be for instance both monthly and yearly archives, that contain duplicate content organized slightly differently. When crawling Web forums and blogs with the traditional crawler approach, we will encounter many of these redundant cases. In extreme cases, the crawler can fall in a *spider trap* because it has infinitely many links to crawl. There are also several noisy links such as to a print-friendly page or advertisement, etc., which would be better to avoid during the constitution of the archive. Traditional approaches are also unable to crawl data from highly scripted Web applications or from the *deep Web* (data accessible behind forms). Finally, classical archiving crawlers do not attempt any form of data extraction whereas archivists and archive users would like to have access to more semantics about the content of the archive, such as timestamps of blog messages, identifiers of contributors, etc., even using complex page navigation for structuring the related information across several pages. For instance, a Web application that organizes its content in a way that one needs to navigate through the calendar to extract the targeted information or, in the case of a Web forum, where one thread can have several posts consisting of several pages and one needs to extract this related information using effective page navigation, the traditional approach will face limitation to do so efficiently, i.e., spending more time in crawling for few interesting pages with no semantics about content.

We briefly discuss next the state of the art. Then we present in Section 3 the proposed approach: to introduce an *application-aware helper* in the crawling process, that assists the crawler throughout the crawling process and will ensure that data is crawled efficiently. In Section 4 we describe our methodology in more detail by describing our Web application detection patterns, a possible structure for a Web application knowledge base, and which kind of crawling actions we want the crawler to perform. We evoke in Section 5 preliminary results and end this paper with a discussion of our future research.

2. STATE OF THE ART

Web crawling is a well-studied problem with still ongoing challenges. Julien Masanès in [18] surveys the field of Web archiving and Web archiving crawling. He discusses in particular crawling the deep Web and stresses the need of archiving the data from both the surface and deep Web for the necessity of Web preservation.

A *focused*, or *goal-directed*, crawler, crawls the Web according to a predefined set of topics [7]. This approach is a different way of influencing the crawler behavior, not based on the structure of Web applications as is our aim, but on the content of Web pages. Our approach does not have the same purpose as focused crawling: it aims instead at a better approach for known Web applications. Both strategies for improving a traditional crawler are thus complementary.

Content in Web applications or *content management systems* are arranged with respect to a template (e.g., template components include left or right side bar of the Web page, page navigation bar, header and footer, main content, etc.) Among the various works on template extraction, Gibson, Punera, and Tomkins [10] have carried out some analysis on the spread of template-based content on the Web. They have found that 40–50% of the Web content (in 2005) is template-based, i.e., part of some Web application. Their findings also suggested that template-based Web pages are growing at the rate of 6–8% per year. This research is a strong hint at the benefit for a crawler to handle in a specific manner Web application crawling.

Though application-aware crawling in general has not yet, to the best of our knowledge, been addressed, there are some efforts on content extraction from Web forums [11, 6]. The first approach, dubbed Board Forum Crawling (BFC) [11], leverages the organized structure of Web forums and simulates user behavior in the extraction process. BFC deals with the problem effectively, but is still confronted with limitations as it is based on a simple rule and can only deal with forums with some specific organized structure. The second approach [6], however, does not depend on Web forum structure. Their iRobot system assists the extraction process by providing the sitemap of the Web application being crawled. The sitemap is constructed by randomly crawling a few pages from the Web application. This process helps in identifying the rich repetitive regions and then further clusters them according to their layouts [25]. After sitemap generation, iRobot obtains the structure of the Web forum in the form of a directed graph consisting of vertices (Web pages) and directed arcs (links between different Web pages). Furthermore a path analysis is performed to provide an optimal traversal path which leads the extraction process in order to avoid duplicate and invalid pages. Our goal would be to develop similar techniques but for arbitrary Web applications, not only Web forums.

As we shall explain, our approach relies on a generic mechanism for detecting the kind of Web application currently crawled. Again there has been some work in the particular cases of blogs or forums. In particular, [14] uses support vector machines (SVM) to detect whether a given page is a blog page. Support vector machines [5, 19] are widely used for text classification problem. In [14], SVMs are trained using various traditional feature vectors formed of the content's bag of words or bag of n-grams, and some new feature for blog detection are introduced such as the bag of linked URLs and the bag of anchors. Relative entropy is used for feature selection. According to the experiments shown in [14], blog

identification was most effective by using features consisting of a combination of URLs, anchors, and meta tags.

Still on the topic of Web application detection, there are some approaches for detecting deep Web sources [3, 4]. The former effort, named *form-focused* crawler [3], was introduced for detecting online databases. This technique uses a focused crawler for a given topic with the assistance of a link classifier which detects searchable forms and prioritizes detected links. This technique, however, has a few limitations: manual tunning, link classifier dependency, and heterogeneous results. The latter approach [4] introduced a more adaptive approach (an *adaptive crawler for hidden-web entities*) that addresses these limitations. It efficiently explores the entry points to the hidden Web with an unknown pattern, then automatically adding this experience to the learning process.

More generally, a few works [1, 16, 15] aim at identifying a general category (e.g., blog, academic, personal, etc.) for a given Web site, using classifiers based on structural features of Web pages that attempt to detect the *functionalities* of these Web sites. This is not directly applicable to our setting, first because the classification is very coarse, and second because these techniques function at the Web site level (e.g., based on the home page) and not at the level of individual Web pages.

Observe that all these application identification techniques heavily rely on trained classifiers. This is a possible direction for our Web application detection mechanism.

3. PROPOSED APPROACH

Our main claim is that different crawling techniques should be applied to different types of Web applications. This means having different crawling strategies for different forms of social Web sites (blogs, wikis, social networks, social bookmarks, microblogs, music networks, Web forums, photo networks, video networks, etc.), for specific content management systems (e.g., WordPress, phpBB), and for specific sites (e.g., Twitter, Facebook). Our proposed approach will detect the type of Web application (general type, content management system, or site) currently processed by the crawler, and the kind of Web pages inside this Web application (e.g., a user profile on a social network) and decide on further crawling actions (following a link, using an API, submitting a form, extracting structured content) accordingly.

To adapt the behavior of traditional crawlers according to our requirements, we have chosen to extend the traditional architecture of a Web crawler in the way depicted in Figure 2. Here the page fetching module (see Figure 1) is replaced by some more elaborate resource fetching component that is able to retrieve resources that are not just accessible by a simple HTTP GET request (but by a succession of such requests, or by a POST request, or by the use of an API), or that are individual Web objects inside a Web page (e.g., a blog post, a comment, a poster's name). An application-aware helper module is then introduced in place of the usual link extraction function, in order to identify the Web application that is currently being crawled, and decide and categorize crawling actions that can be performed on this particular Web application.

These modifications will be implemented in two Web crawlers: the proprietary crawler of the Internet Memory Foundation, with whom we are closely collaborating, and into a customized version of Heritrix [23], developed by the ATHENA research lab in the framework of the ARCOMEM project [2]. The following section will describe the application-aware helper module with more detail.

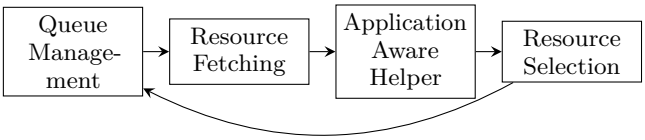

Figure 2: Extended architecture of the Web crawler

4. METHODOLOGY

This section introduces the application-aware helper module. This module assists the archiving crawler for acquiring content from the social Web in an intelligent and adaptive manner. This module enriches the functionalities of the crawler, and makes the crawling process more efficient.

Knowledge base of Web applications. The crawler will be assisted by a *knowledge base* of Web applications that describes how to crawl a Web site in an intelligent manner. This knowledge base will specify how to detect specific Web applications and which crawling actions should be executed. The knowledge base will be arranged in a hierarchical manner, from general categorizations to specific instances (Web sites) of this Web application. For example the social media Web sites can be categorized into blogs, Web forums, microblogs, video networks, etc. Then we can further categorize these specific types of Web applications on the basis of the content management system they are based on. For instance, Wordpress, Movable Type, etc., are examples of blog content management system, whereas phpBB and vBulletin, etc, are examples of Web forum content management systems.

Moreover, a given Web application usually consists of different kinds of Web pages: in a Web forum, there are pages that display lists of forums, pages that display the list of posts under specific forums, and pages that point to individual posts with their comments. Thus, the knowledge base will describe the different kinds of Web pages under a specific Web application and then, based on this, we can define different crawling actions that should be executed against this specific page level.

The knowledge base is to be specified in a declarative language, so as to be easily shared and updated, hopefully maintained by non-programmers, and also possibly automatically learned from examples. The W3C has normalized a Web Application Description Language (WADL) [12] that allows describing resources of HTTP-based application in a machine processable format. WADL is used for describing the set of resources, their relationship with each other, the method that can be applied on each resource, resource representation formats, etc. WADL may be a candidate component and export format, of our knowledge base, but does not satisfy all our needs: description of Web application recognition patterns, and Web application interactions that go beyond simple GET and POST requests. Consequently, our knowledge-based will be described in custom XML format, well-adapted to the tree structure of the hierarchy of Web applications and page levels.

Web application detection module. One main challenge in intelligent crawling and content extraction is to identify the Web application and then perform the best crawling strategy accordingly. There is not much work done on the Web application identification, but there are a few efforts for classifying Web pages under different categorized Web applications [1, 16, 15].

To detect a particular Web application, our knowledge base allows describing several rules, based on URL patterns, HTTP metadata, textual content, XPath patterns, references to a classifier, and, possibly, Web-graph–based features. The identification of the page level inside a Web application can also be done by categorizing the page according to structural properties.

Let us take the example of the vBulletin Web forum content management system, that can for instance be identified by searching for a reference to a `vbulletin_global.js` JavaScript script by using a simple `//script/@src` XPath expression. Pages at the level of "list of forums" are identified[1] when they match the `//a[@class="forum"]/@href` XPath expressions.

Crawling and extraction. After detecting the application to whom the current Web page belongs, the next stage is to determine the corresponding crawling actions. Crawling action scopes go beyond just a list of URLs to add to the queue. It can be any action that involves using APIs to extract relevant data from the social network sites, such as Twitter, or performing complicated interactions with AJAX-based applications, or identifying Web objects in particular Web application. More specifically, crawling actions are of two kinds:

Navigation actions: to navigate to another Web page or Web resources.

Extraction actions: to extract individual semantic objects from Web pages (e.g., timestamp, the blog post, the comments).

We similarly want a declarative language for describing all crawling actions (again, the hope is to have an easily maintainable knowledge base, including machine maintainability). We therefore need a navigation and extraction language able to access data from the deep Web as well as regular URLs. We will use OXPath [9]. OXPath is an extension of XPath, with added facilities for interacting with Web applications and extracting relevant data. It allows the simulation of user actions to interact with scripted multipage interfaces of the Web application (the evaluator relies either on a Mozilla-based or Webkit-based browser). It inherits from XPath as well as allows to use CSS-based selectors. It makes possible to navigate through different pages by using clicks and even allows to extracting information from previous pages. An open-source implementation is available, that will be integrated into our system.

As an example, a simple OXPath action that can be performed on the vBulletin example is

```
//a.forum/@href/{click/}
```

[1]The example is simplified for the sake of presentation; in reality we have to deal with several different layouts that vBulletin can produce.

that "clicks" on every link to an individual Web forum. We refer to [9] for more elaborate examples of OXPath expressions.

We have also studied a few alternatives to OXPath, especially [17, 20, 21, 22, 24]. These approaches, except [24], do not deal with deep Web interaction such as form submissions and page navigation. [22] uses Datalog as a programming language for extracting data from Web pages. This approach introduces the Xlog language as an application of Datalog, which includes predefined extraction predicates. This technique opens the window for the researcher to use Datalog as a base for the extraction process, but still requires some amount of effort to deal with the deep Web. [21] faces the same challenge. This approach also extracts the content from simple pages or from bibliographic pages. It does not simulate any action for form filling or page navigation. They introduced a wrapping language named as Wraplet, that extracts structured data from Web pages (written in HTML). This language is based on wraplet expressions written as a script (a data extraction expression), takes an HTML document as an input, and produces output in the form of XML. But unfortunately this approach can be applied only to singleton pages, which do not hold the dynamic nature of the Web. The technique which is most similar to OXPath and facilitates functionalities such as form submission and navigation is [24]. This paper, written in a very simple tone, provides several examples to clarify the concepts. The authors have introduced a system named browser-oriented data extraction system (BODE) that extracts the informations from Web pages and uses URL links to navigate to the next page for information extraction. BODE harvests Web applications even when they which are using scripting functions such as JavaScript or AJAX for accessing the next page. It also simulates user actions to deal with the deep Web. In despite of all these advantages, BODE does not consider memory management constraints, which makes it unsuitable for our system where we want to continuously crawl and archive large volumes of data: browser instances are replicated for multiple page navigation, which raises the issue of performance. In our system, we will deal with-large scale object extraction, and for that we need a system which takes care of performance and memory management. OXPath deals with this properly and scales well in time and memory.

5. RESULTS

Since this PhD work has only been started a few months ago, we have focused on developing an architecture and the methodology described in the previous section. A basic prototype of the application-aware helper has been implemented, with a proof of concept that it is indeed feasible, with recognition for a couple of Web applications. For instance, the prototype has been evaluated against the vBulletin application, on a variety of Web sites using this content management system. For now, the system is able to detect the type of the Web application, the level in the Web application, and executes the corresponding crawling actions. Recently, we have integrated the YFilter system [8] (a NFA based filtering system) for efficient indexing of detection patterns, in order to quickly find the relevant Web applications. Now the system is able to extract more interesting pages with a minimum effort as compared to traditional crawling approach. We still need to execute crawling actions by using

an OXPath evaluator (or directly convert them into URLs or XPath expressions when possible) and interface our helper with the crawler, but the initial results are satisfactory and promising for our future research.

6. FUTURE WORK

A number of interesting challenges warrant further investigation and will be our agenda for the rest of this PhD:

(1) Using XPath 1.0 expressions for detection patterns faces some expressiveness limitations: in some cases, for instance, regular expressions may be required to identify a Web application. We have the option of switching to XPath 2.0 expressions or to add extension functions for this purpose, but we should strive also at keeping a language that is as declarative as possible for optimization purposes.

(2) One significant challenge is to investigate the possible automatic, unsupervised, learning of new Web applications (by the inference of common patterns), and the adaptation to slight changes in the templates that render the wrappers unusable.

(3) We also must ensure throughout our work the possible fine integration with the crawler(s) by developing mechanism for interacting with the other components. Among the challenges here is the fact that the crawler should still be responsible for all Web interactions, in order to maintain politeness constraints, whereas, for instance, some crawling actions may require going through an external program (an API crawler, or an OXPath evaluatior).

Obviously, an additional challenge throughout this work is to come up with metrics that allow formal evaluation of the performance of our system (both in terms of effectiveness and efficiency) with respect to classical Web crawling approaches.

7. ACKNOWLEDGMENTS

The described work was funded by the European Union Seventh Framework Programme (FP7/2007–2013) under grant agreement 270239 (ARCOMEM). We are also grateful to Julien Masanès (Internet Memory) for discussions on the topic of this PhD.

8. REFERENCES

[1] E. Amitay, D. Carmel, A. Darlow, R. Lempel, and A. Soffer. The connectivity sonar: Detecting site functionality by structural patterns. In *HT*, 2003.

[2] ARCOMEM Project. `http://www.arcomem.eu/`, 2011–2014.

[3] L. Barbosa and J. Freire. Searching for hidden-Web databases. In *WebDB*, 2005.

[4] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *WWW*, 2007.

[5] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, 1992.

[6] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang. iRobot: An intelligent crawler for Web forums. In *WWW*, 2008.

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

[8] Y. Diao, M. ALTINEL, M. J. Franklin, H. Zhang, and P. Fischer. Path sharing and predicate evaluation for high-performance XML filtering. *ACM TODS*, 2003.

[9] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, and A. J. Sellers. OXPath: A language for scalable, memory-efficient data extraction from web applications. *PVLDB*, 4(11), 2011.

[10] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *WWW*, 2005.

[11] Y. Guo, K. Li, kai Zhang, and G. Zhang. Board forum crawling: A Web crawling method for Web forums. In *WIC*, 2006.

[12] M. Hadley. Web application description language. `http://www.w3.org/Submission/wadl/`.

[13] International Business Times. `http://www.ibtimes.com/articles/175488/20110706/obama-twitter-townhall.htm`, 2011.

[14] P. Kolari, T. Finin, and A. Joshi. Svms for the Blogosphere: Blog Identification and Splog Detection. In *AAAI*, 2006.

[15] C. Lindemann and L. Littig. Coarse-grained classification of Web sites by their structural properties. In *CIKM*, 2006.

[16] C. Lindemann and L. Littig. Classifying Web sites. In *WWW*, 2007.

[17] M. Liu and T. W. Ling. A rule-based query language for HTML. In *DASFAA*, 2001.

[18] J. Masanès. *Web archiving*. Springer, 2006.

[19] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Workshop on Neural Networks for Signal Processing*, 1997.

[20] A. Sahuguet and F. Azavant. Building light-weight wrappers for legacy Web data-sources using W4F. In *VLDB*, 1999.

[21] N. Sawa, A. Morishima, S. Sugimoto, and H. Kitagawa. Wraplet: Wrapping your Web contents with a lightweight language. In *SITIS*, 2007.

[22] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using Datalog with embedded extraction predicates. In *VLDB*, 2007.

[23] K. Sigurðsson. Incremental crawling with Heritrix. In *IWAW*, 2005.

[24] J.-Y. Su, D.-J. Sun, I.-C. Wu, and L.-P. Chen. On design of browser-oriented data extraction system and plug-ins. In *JMST*, 2010.

[25] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *SIGKDD*, 2007.