A User-Centric Recursive Rule-Based Web Crawler

Xiaoying Chang
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
xiaoyingchang@cmail.carleton.ca

Changcheng Huang
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
huang@sce.carleton.ca

Andrew Droll
Gnowit Inc.
Ottawa, Canada
andrew@gnowit.com

***Abstract*—The rapid growth of the World Wide Web poses tremendous scaling challenges to general-purpose crawlers in terms of the need to harvest massive amounts of big data. The key solution is to selectively crawl web pages that meet certain requirements, so as to decrease the total number of webpages to be visited and increase resource efficiency. In this paper, we propose a user-centric recursive rule-based crawler that will crawl the web pages in one domain according to user-specified rules. The process is recursive and can reach any depth. The structure of user-specified rules is introduced, together with our PhantomJS/Java solution for interpreting these rules to perform crawling. We test the crawler on several existing websites, and the results show that highly accurate contents from deep inside domains can be fetched using a small number of user-specified rules and at a low time cost.**

Keywords—user-specified rule; crawling instruction; recursive web mining; big data retrieval; big data acquisition; AI

I. INTRODUCTION

Web crawling is the process of locating, fetching, and storing data about available web pages, as well as refreshing stored data about the contents of previously downloaded web pages [1]. A web crawler is a program developed to perform such tasks. The crawling process typically proceeds as follows: Seed Uniform Resource Locators (URLs) are stored in a list before crawling. The crawler starts visiting each of the seed URLs and downloading the pages from the Web. It then detects possible links from the downloaded pages and adds them to its list of future pages to be crawled. The crawler will repeat this process until all the links in the list have been visited or desired number of pages are downloaded [2].

The rapid growth of the World Wide Web results in an increasing amount of information available to crawlers that rely on this type of recursion. This makes the crawler's ability to harvest massive amounts of data (big data) an important consideration. To tackle this scalability problem, several approaches have been proposed, including parallel web crawling, geographically distributed web crawling, and focused web crawling. In parallel web crawling, multiple crawling processes are run in parallel to retrieve information from the web [3]. Each crawling process starts with its own set of seed URLs and follows links without interfering with other crawling processes. In this way, the download rate can be optimized [4]. Another effective way to achieve scalability is geographically distributed web crawling, in which several geographically distributed client machines operate independent crawlers targeting domains hosted in relatively close proximity [5]. This approach can decrease crawling latency and reduce network overhead because the crawler is much closer to the web pages it needs to crawl. the two alternate models mentioned above still require crawlers to fetch all available web pages, which will often waste significant time and storage resources if some available pages are irrelevant to the crawler operator's objective. This represents an efficiency problem, which often compounds the performance challenges crawlers face when dealing with increasing volumes of web data.

To make the crawling process more efficient, focused web crawlers have been proposed to selectively locate and download web pages that best match a pre-defined set of topics [6]. In focused web crawling, a crawler specializing in one or more topics only seeks web content relevant to those topics. In this model, the focused crawler will be much faster to detect webpage changes within its focus than a crawler that crawls the entire Web. Such crawlers can also reduce hardware cost and increase network resource efficiency [7]. However, classifying topics on the Web is challenging, in particular when faced with today's constant expansion of new categories of information. In general, focused web crawlers cannot guarantee that all pages relevant to its defined topics are actually crawled. Indeed, focused crawlers must rely on either metadata or statistically-based heuristics, on example-based machine learning, or on pre-computed indices of content (e.g. search engines) in order to function efficiently. To address focused crawling needs in a more flexible way, user-centric web crawling has been proposed. User-centric crawling has emerged to solve some of these typical problems associated with focused crawling [8]. This crawling paradigm aims to schedule web pages for selective downloading with the purpose of maximizing the quality of the user experience for those who will eventually query the data retrieved by the crawler [8]. Therefore, the effectiveness of crawling is measured by estimates of users' satisfaction.

Different from existing user-centric solutions which are based on general users' interests for keyword-based crawling, our work in this paper is inspired by the frame-systems in artificial intelligence (AI), in which frame's terminals are filled with values based on how human mind works. The main idea of our paper is to allow users (usually system administrators who collect users' requirements) to specify a small number of rules describing crawling tasks for a given domain according to

their needs, and then to have the crawler to perform a rule-based crawl following crawling instructions built from these rules. Moreover, the depth search theory is adopted to achieve recursive crawling.

The contributions of this paper can be summarized as follows: Firstly, a simple rule engine is designed for defining user-specified rules. Secondly, a hybrid PhantomJS/Java solution, which interprets crawling instructions from the user-specified rules is developed. Based on the interpreted rules, this solution is then capable of performing recursive, tailored crawls of web domains. Thirdly, experimentation is conducted to test crawl paths of various depths using the described solution. The test results show that highly accurate contents from deep inside domains are fetched by the smart crawler using a small number of user-specified rules in a short period of time.

The rest of this paper is organized as follows: Section II specifies details of our user-centric recursive rule-based web crawler, section III provides experimentation and test results for the proposed crawler, and section IV describes conclusions drawn from this research.

II. User-Specified Rule and Recursive Crawling Instruction

In this section, we describe the user-centric recursive rule-based crawler we have developed for this research. The goal is to allow users of a search engine to specify a small number of rules which are transformed to powerful smart crawling instructions for a given domain, tailored to recursively retrieve particular data items of interest quickly and with little user effort. The overall process works as follows: At first, some user-specified crawling rules that can be used to construct a recursive structure are specified for a given domain, where each rule includes several specific, pre-defined attributes. Each attribute has a user-specified value which is used to specify how data should be extracted from certain web pages in the domain. These rules use a structured format, which allows them to specify the construction of internal crawling instructions. Instead of adding every link to the list of future pages to be crawled as traditional crawlers do, our smart crawler only follows links that satisfy the attributes in these constructed instructions. The crawler functions in a recursive way based on this process, where, for each subsequent link to be visited, an internal crawling instruction is built based on the relevant pre-set user-specified rules. This recursion terminates once the list of subsequent links to be crawled is empty.

Next, we use the Government of Canada Consultations website demonstrated in Fig. 1 as an example to make clear how the proposed crawler works. The URL of this website is: http://www1.canada.ca/consultingcanadians/page/search (henceforth shortened to "GCC"). Suppose a user or an administrator in charge of a group of users wishes to get all PDF file links within Consultation pages related to "international" from different depths of this web domain. To accomplish this goal, we must define a set of user-specified rules with the format shown in Table I. From the table we can see that each row represents one rule consisting of several attributes. For simplicity, we name them Rule 1, Rule 2, … etc. respectively from the second row, third row, until the last row in the table. Below, we describe each table attribute in more detail, as well as how these user-specified rules are used to build crawling instructions in a recursive manner.

A. User-Specified Rule Attributes

1) baseURL: This attribute is the first column in Table I, where the value of *baseURL* is "GCC". It is the web domain to which the rule applies. This also represents the seed URL for crawling of the domain – the crawler starts here and proceeds recursively by following the user-specified domain rules.

2) ruleType: This attribute indicates the action specified by the rule (the second column in Table I). It takes one of two values: "keep" or "follow". If the *ruleType* is "keep" (Rule 2 and Rule 4), all identified links will be saved to a file, but not scheduled for future crawling (**"**keep" rules are often used as crawl termination conditions). Alternatively, if the *ruleType* is "follow" (Rule 1 and Rule 3), then all identified links will be saved to another file, which stores links that are scheduled for future recursive crawling. This attribute can control the number of links to be visited in the subsequent crawling level. If certain links only need to be retained for indexing or final scraping of data, but do not need to be crawled for additional deeper links, then using a keep instruction can result in the desired behavior while restricting unnecessary downloads.

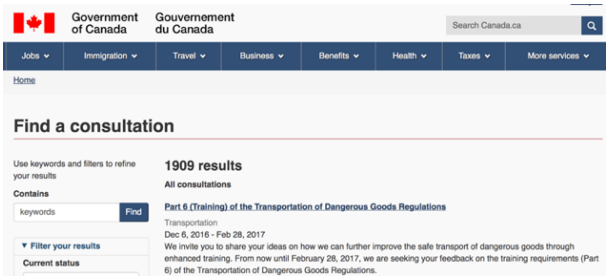

Fig. 1. Government of Canada Consultations webpage.

TABLE I. Consultation Crawling Rules

baseURL	ruleType	fromType	toType	pattern	id	class	tag
GCC	follow	base	topLevel	international	all	col-xs-12 col-sm-7 col-md-8 col-lg-9	all
GCC	keep	topLevel		.pdf	all	all	all
GCC	follow	topLevel	secondLevel	international	all	all	main
GCC	keep	secondLevel		.pdf	all	all	all

3) fromType and toType: These two attributes (column 3 and column 4 in Table I) provide the main mechanism for controlling which rules apply to which webpages in any given web domain, and also imply each rule's corresponding crawling level. Our recursive rule-based crawling process is achieved by correct selection of the values of these two attributes as well as the *ruleType* attribute.

To be specific, in the smart crawler system, each link scheduled for crawling is assigned a page type which can appear in either *fromType* or *toType* attribute. The *baseURL* of the domain is given the type "base". If a page to be visited has page type "base", then all rules with *fromType* "base" (Rule 1) are executed on that page – in other words, the *fromType* of a user-specified rule indicates the page type of pages to which the rule should be applied. Working in tandem, the *toType* rule attribute represents the page type to be assigned to all links identified for next crawling level, in which the rules (Rule 2 and Rule 3) whose *fromType* ("topLevel") equal this *toType* will be executed. This process will be executed recursively to any depth until a termination condition is met (after execution of Rule 4) when no rules with the *ruleType* equaling to "follow" exist. Rules with *ruleType* equaling to "keep" need only specify *fromType*, since they do not result in any further links to be scheduled for crawling. Conversely, rules with *ruleType* equaling to "follow" must specify both *fromType* and *toType*, since they are expected to result in the identification of additional links to follow which must each be assigned a page type. Besides, it is necessary to have at least one rule with *ruleType* equaling to "follow" and *fromType* equaling to "base" in order for a domain crawl to progress past the initial *baseURL*.

4) pattern: The value of this attribute (column 5 in Table I) is a regular expression defining a filter on links to be kept or followed (depending on the *ruleType*). The default value of the *pattern* attribute is "all", which indicates that no filter should be applied to the identified links – in this case, all links matching the other rule attributes will be retained to be either kept or followed. If the value of *pattern* is not "all" (Rules 1 - 4), the crawler will filter the set of identified links to only those which contain a match to the pattern regular expression. Since users want to get all the PDF file links within "international"-related Consultation pages, Rule 1 and Rule 3 will ask the crawler to retrieve links containing "international" keywords (e.g. http://www.dfo-mpo.gc.ca/international/psma-cfpr/index-eng.htm). After visiting each of these links, PDF file links (e.g. http://www.oecd.org/dac/effectiveness/34.pdf) will be returned as instructed by Rule 2 and Rule 4. This regular expression-based URL filtering mechanism, on a per-rule basis, provides users of the recursive crawler with fine-tuned control over which links should be retained at every stage. Appropriate specification of these parameters can result in a very large reduction in the time requirement to access desired data within the web domain.

5) id, class and tag: The values of these attributes (last three columns in Table I) give the id, class name, and tag name of Hyper Text Markup Language (HTML) elements within the visited webpage, respectively. The default value of each of these attributes is "all". If all of these attributes are simultaneously set to the default value of "all" (Rule 2 and Rule 4), then identified links on the page belonging to HTML element(s) with any *id*, *class* name or *tag* name will be kept or followed (as determined by the *ruleType*). Additionally, it is required that at most one of *id*, *class*, or *tag* may be given a non-default value for any particular rule. This is due to the fact that only one of these attributes is required to specify certain element(s) in the Document Object Model (DOM). If one attribute's value is not "all" (Rule 1 and Rule 3), then only links that are contained within DOM elements matching the respective HTML attribute are given consideration (with respect to being kept or followed) by the crawler. In the case of Rule 1, since in the main webpage, all Consultation page links only reside in the HTML elements whose class name is "col-xs-12 col-sm-7 col-md-8 col-lg-9" as specified from the Consultation webpage which is shown in Fig. 2, so we set this class name to this Rule's *class* attribute value.

This *id*, *class*, or *tag* specification allows the user to indicate that only links appearing on certain parts of particular page types should be considered for retention or crawling – in other words, it allows further filtration of pages to be visited by isolating elements from the page's design. Together with the pattern attribute, the *id*, *class*, or *tag* attributes facilitate powerful rule-based filtration of the crawler's trajectory through any given domain, with the capability to filter based on both the semantic form of URLs (via pattern) and on the position of links on each page (via *id*, *tag*, and *class*).

All attributes in each rule decide the crawling process of a crawler together to achieve our goal. Following Rule 1 in Table I, our crawler will label all the Consultations links (filtered by *class* attribute) containing "international" (filtered by *pattern* attribute) with the type "topLevel" (*toType* attribute), and save them to the file containing links to be visited in the next stage. Rule 2 will let the crawler save all the PDF file links (filtered by *pattern* attribute) within all the "topLevel" (*fromType* attribute) links to the result file. At the same time, as instructed by Rule 3, all "international"-related links (filtered by *pattern* attribute) within main content (filtered by *tag* attribute) will be tagged with the type "secondLevel" (*toType* attribute) and be crawled in the next level. Finally, according to Rule 4, all PDF file links (filtered by *pattern* attribute) in the 3rd depth of the web domain will be retained to the result file. Generally, rules with same *fromType* will be combined together to construct crawling instructions for the crawlers to follow, we will introduce this in the next part with more details.

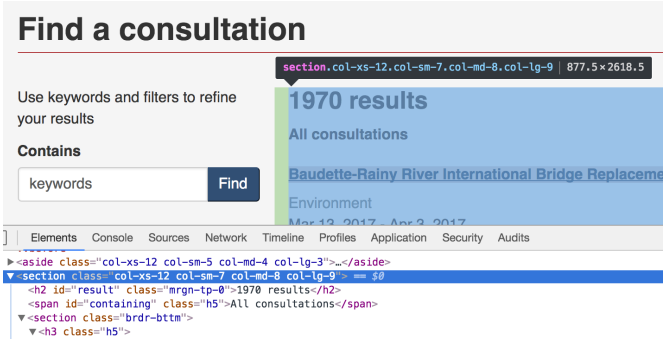

Fig. 2. Consultation webpage with specified class element(s).

B. Recursive Crawling Instruction

Once all necessary user-specified rules have been constructed and properly stored, the crawler application can be started from the rules with *fromType* equal to "base" on that domain. The system will automatically construct comprehensive crawling instructions for each page to be crawled (starting with the *baseURL* for that domain). The structure of a crawling instruction in the JavaScript Object Notation (JSON) format is shown below:

{"baseURL": "", "urlToCrawl": "", "keepRules": [{"pattern": "", "id": "", "class": "", "tag": "", "filterField": "", "filterValue": ""}], "followRules": [{"toType": "", "pattern": "", "id": "", "class": "", "tag": ""}]}.

Here, the "baseURL" JSON parameter is identical to the *baseURL* rule attribute described above. The "urlToCrawl" JSON parameter is the URL of the link specified for crawling at present – the target link for this instruction. In particular, when crawling the first page on any given domain, the "urlToCrawl" parameter is equal to the "baseURL" parameter. The "keepRules" parameter is itself a JSON-type array which includes all user-specified rules for this page whose *ruleTypes* are "keep", while "followRules" is also a JSON-type array composed of all user-specified "follow" type rules. We notice that each element in "followRules" has the "toType" parameter, and that these are lacking in the "keepRules" array. This is to be expected – as discussed in earlier, *toType* only applies to rules with *ruleType* equal to "follow". We remark on the fundamental quality of this JSON format: All rules for the link presently to be crawled are encapsulated together in the JSON instruction. If there are multiple rules for the current page, all of them are specified at once – thus, the page need only be crawled once, and all relevant rules are applied to its DOM after that single crawl.

In terms of the system's technological design, user-specified rules are stored in a Structured Query Language (SQL) database table. A Java application is in charge of reading rules from the table and constructing JSON instructions, as well as scheduling and triggering actual crawls, which are executed for individual pages by a PhantomJS script based on a JSON instruction passed as an argument. For each page to be crawled, a full JSON instruction object, as specified in the previous paragraph, is constructed. Thus, the number of instructions constructed is equal to the number of pages crawled. Each page's JSON instruction is based entirely on the user-specified rules whose *fromType* is equal to the page's page type. This design gives the crawler extreme flexibility in accessing specific desired data, while retaining the ability to crawl to any depth within a given domain. These advantages allow our recursive rule-based crawler to outperform other crawlers at tasks focused on retrieving specific, customized data from deep within domains. Moreover, by scheduling rule-based crawls frequently, it is possible to use the smart crawler to find newly posted content (or updated content) quickly, while minimizing network and resource overhead, since very few pages need to be downloaded.

As for our example, the first crawling instruction is built from Rule 1 whose type is "base". The goal is to extract all links from the main content element related to "international" of the search page and label each of them with the type "topLevel". The crawling instruction built using this rows is:

{"baseURL": "GCC", "urlToCrawl": "GCC", "keepRules": [], "followRules": [{"toType": "topLevel", "pattern": "international", "id": "all", "class": "col-xs-12 col-sm-7 col-md-8 col-lg-9", "tag": "all"}]}.

The crawling process is as follows: initially, the crawler extracts links from the Government of Canada Consultations website according to the "followRules" in the above instruction. The links identified by the single rule with *ruleType* equal to "follow" are each labeled with page type "topLevel". After *baseURL* crawling is completed, sequential crawling of the identified links to follow commences. For each of the links (type is "topLevel") to follow, a crawling instruction is built from Rule 2 and Rule 3 whose *fromType* equal to "topLevel". Assume that the URL of one such link is "GCC1". Then the crawling instruction for this link is:

{"baseURL": "GCC", "urlToCrawl": "GCC1", "keepRules": [{"pattern": ".pdf", "id": "all", "class": "all", "tag": "all"}], "followRules": [{"toType": "secondLevel", "pattern": "international", "id": "all", "class": "all", "tag": "main"}]}.

At this time, the crawler will follow all the links with the type "topLevel", and analyze their DOMs. Documents with PDF format will be kept (Rule 2). Those links from the webpage elements whose tags are "main" and also contain "international" are indexed as subsequent links to follow, with the page type "secondLevel" (Rule 3). Suppose that the URL of one such link is "GCC2". Then the crawling instruction for this link is:

{"baseURL": "GCC", "urlToCrawl": "GCC2", "keepRules": [{"pattern": ".pdf", "id": "all", "class": "all", "tag": "all"}], "followRules": []}.

According to this instruction, all the PDF file links within these "secondLevel" type links will be kept but not followed (Rule 4), and the crawling process will terminate afterwards. In the end, all PDF file links containing "international" keyword and also within elements whose tags are "main" are extracted from each "topLevel" link. Moreover, all PDF file links for each "secondLevel" link are extracted in the last crawling level.

By recursive application of a small set of rules of the type defined above, the smart crawler can perform extremely customized crawls of any domain, returning structured data on identified content in a bandwidth- and compute-efficient manner (requiring only minimal initial configuration by a user or system administrator) while still locating any recently-added content or modified URLs.

Our discussion so far should illustrate the power of our recursive rule-based system to allow aggressive guidance of the crawler's trajectory through domains, and to permit fine-tuned filtration of which links are followed or kept at each stage. As far as we know, this is the first research that defines user-specified rules tailored to specific recursive big data acquisition preferences. Previous crawlers visit too many webpages because they either crawl every link, or broad swaths of links that are very generally related to specific topics. Even user-centric crawlers tend to crawl every accessible page, despite

employing sophisticated scheduling to limit unnecessary refreshes of stale content that is unlikely to have been updated.

Our crawler design is significantly more favorable for cases in which the search engine provider serves many clients who each desire access to very customized data sources – including data that are frequently updated, added or changed, but organized in a complicated way deep inside domain structures. Furthermore, because web domains usually use fixed HTML templates, the contents that clients are interested in will always within the same HTML id, class and tag, which makes the rule-based crawler more robust without the necessity to change rules' attribute values. Training system administrators and advanced users in rule construction is reasonably straightforward, and requires only some surface analysis of the structure of links within the target domain. By pre-determining the crawling trajectory, downloads of irrelevant pages will be avoided because the crawler only retrieves pages necessary to meet the user's needs, resulting in greatly increased speed, regardless of how deeply the pages are embedded in the source domain.

III. Test Results

We use a hybrid approach of Java and PhantomJS to implement the smart crawler, where Java is used to build crawling instructions and to analyze crawling results, and PhantomJS acts in the role of the actual crawler, visiting webpages according to crawling instructions and storing results in files to be subsequently read by Java. In our implementation, we use the open source database PostgreSQL as storage for sets of rules. It is installed on a MacBook Pro laptop with a 2.7 GHz Intel Core i5 processor and 8 GB 1867 MHz DDR3 memory. We use pgAdmin3 as the GUI for the database to create rules. The timeout interval for downloading a webpage in PhantomJS is set to 1 second.

We performed several test cases in terms of total crawling time, percentage of the number of useful links to the total number of saved links (the number of useful links, i.e., user required links, is the same for different kinds of crawlers, so we use this metric to identify the crawling efficiency of a crawler), and total number of crawling instructions created by Java with different desired crawl depths and rule complexity. The results are compared with the results of existing classification-based (focused and user-centric) crawlers and traditional web crawlers (crawl every link in one web domain). In all of our test cases, the traditional crawler generated much larger values in three metrics, i.e. total crawling time, total number of crawling instructions, and total number of saved links than the other two approaches. If the test results of the three crawlers are shown in the same diagrams, the results of the other two approaches will become almost invisible. Therefore, we use 10% of the results of traditional crawlers for those first two metrics as our bases for normalizing the total crawling time and number of crawling instructions resulting from the other two approaches. We also choose 10% of the third metric when calculating percentage of useful links of the traditional web crawler.

We will illustrate the test results with two examples: Government Consultation domain crawling and YouTube video domain crawling. Their test results are shown in Fig. 3 and Fig. 4. The graphs illustrate that our proposed crawler can return highly accurate content (higher percentage of user required links and fewer useless links) from deep inside domains using very few rules and crawling instructions in less total crawling time. Next, we introduce these two test cases in more detail.

A. Government "Consultation" Domain Crawling

Test 1 implements the example given in Section II. It is a 3-depth crawling process with the crawling rules showing in Table I. As comparisons, the classification-based web crawler and traditional crawler are tested with the same crawling targets as our crawler. The results are shown in Fig. 3, which clearly demonstrates that our proposed crawler performs the best in terms of all test metrics. As explained in Section II, our recursive crawler pre-defines user-specified rules tailored to specific recursive big data acquisition preferences by taking advantage of highly static HTML templates of websites. Thus, the downloading of irrelevant pages is avoided because the crawler only retrieves necessary pages at any depth, recursively, to meet the user's requirements with significantly increased speed. In contrast, classification-based crawlers and traditional crawlers visit too many webpages by visiting links that are very generally related to specific topics, including stale contents that rarely change, or downloading and updating every page, which greatly decreases the efficiency of the crawler from users' and administrators' perspectives.

We can also see from the graph that the more links to crawl, the greater the number of crawling instructions that will be built, which leads to longer crawling time and lower percentage of useful links.

B. YouTube "Video" Domain Crawling

In test 2, we select a very popular video watching web domain with more features – the YouTube website, (https://www.youtube.com) to exemplify how our crawler works for different kinds of crawling tasks.

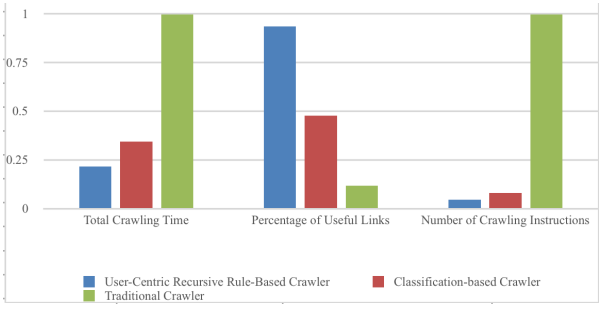

Fig. 3. Government "Consultation" domain crawling results.

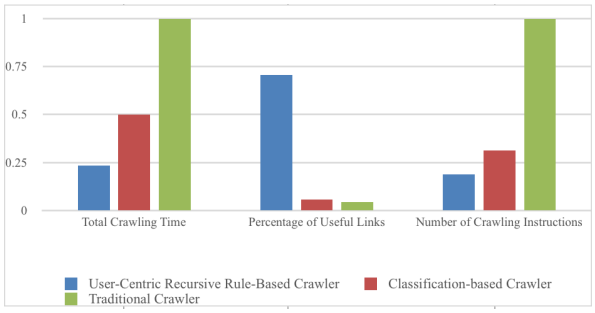

Fig. 4. YouTube "video" domain crawling results.

TABLE II. YOUTUBE CRAWLING RULES

baseURL	ruleType	fromType	toType	pattern	id	class	tag
YT	follow	base	topLevel	/feed/	all	feed-item-dismissable	all
YT	keep	topLevel		youtube	browse-items-primary	all	all
YT	follow	topLevel	secondLevel	/channel/	browse-items-primary	all	all
YT	follow	secondLevel	thirdLevel	watch	browse-items-primary	all	all
YT	keep	thirdLevel		youtube	watch-header	all	all

Suppose that users are interested in locating all the video links in the YouTube "recommended" and "popular" pages, and saving a copy of all metadata about those video links. With YouTube, those links can be identified by a number of keywords and metadata. The crawling rules created for the crawler to retrieve the above contents are shown in Table II**,** where we use "YT" to represent the base domain URL for YouTube. This is a 4-depth recursive crawling process in which the first step is to parse the HTML content at the base URL to get links including recommended and popular videos and tag these links with type "topLevel" (Rule 1). This is achieved by setting the *pattern* to "/feed/" because all links to popular and recommended videos are within webpages whose URLs contain this keyword. The class name is set to "feed-item-dismissable" to narrow down the search scope to page elements where these URLs reside.

Following Rule 2, when visiting each "topLevel" link, the crawler will save all the video links containing "youtube" (to ensure the videos come from the YouTube web domain) in the main section (within the HTML element whose *id* is "browse-items-primary") to the result file. At the same time, according to Rule 3, those video links containing "/channel/" keyword will be tagged with the type "secondLevel" to be visited in the next level. Note that these "secondLevel" links are the channel webpage links the above recommended or popular videos belong to. Then comes to the third crawling level, in which each of the "secondLevel" links will be visited as designated by Rule 4. As a result, video links containing "watch" keyword in the main section of each channel webpage will be tagged with the type "thirdLevel". Finally, all links containing metadata information about each "thirdLevel" link will be extracted. The crawler will terminate afterwards since there are no "follow" type rules in this next crawling level.

Test results are shown in Fig. 4, in which classification-based crawler and traditional crawler are tested to achieve the same goal as our crawler. This graph illustrates the superiority of our user-centric recursive rule-based crawler in the video domain with deeper crawling depth (more recursive processes) and more user-specified rules.

By analyzing all the test cases, we can conclude that the number and type of rules use by our approach are so flexible that all kinds of users' crawling requirements in any domain can be easily met. More importantly, we can use the crawler to perform crawling tasks in different domains by only modifying the rules, without changing the code of the crawler in order to execute different crawling tasks. This significantly reduces the workload to program different kinds of highly customized crawlers.

IV. CONCLUSION

We proposed a user-centric recursive rule-based web crawler that allows a user (usually system administrator who collects user requirements) to set up highly customized recursive domain crawling tasks using small sets of rules, returning only desired content, in a structured way following crawling instructions generated from these user-specified rules. We used Java/PhantomJS to implement this crawler, and tested the approach for several user requirements across different web domains. The results show that the crawler is able to fetch user-desired links efficiently by only visiting and downloading certain links, at any depth. Storage and network utilization is greatly reduced by eliminating the retrieval of unnecessary content. Furthermore, our crawler reduces the workload to program different kinds of highly customized crawlers by adopting the same code for any crawling domain and goal. Future work will extend the crawler to handle interactions with Rich Internet Application (RIA) elements on pages. Moreover, a web browser plugin will be designed to make it easy to generate custom crawler rules using a GUI-based interface.

REFERENCES

[1] C. Olston and M. Najork, "Web crawling," Foundations and Trends in Information Retrieva. USA, vol. 4, pp. 175-246, Mar. 2010.

[2] M. H. Alam, J. Ha, and S. Lee, "Novel approaches to crawling important pages early," Knowl. Inf. Syst. London, vol. 33, pp. 707-734, Dec. 2012.

[3] J. Cho and H. Garcia-Molina, "Parallel crawlers," In Proceedings of the 11th international conference on World Wide Web," Hawaii, USA, May 07 – 11, 2001, pp. 124-135.

[4] M. A. Kausar, V. S. Dhaka, and S. K. Singh, "An Effective Parallel Web Crawler based on Mobile Agent and Incremental Crawling," Journal of Industrial and Intelligent Information, vol. 1, pp. 86-90, June 2013.

[5] C. Fetzer, P. Felber, É. Rivière, V. Schiavoni, and P. Sutra, "Unicrawl: A practical geographically distributed web crawler," in Proceedings of the 8th International Conference on Cloud Computing, New York City, USA, June 27-July 2, 2015, pp. 389-396.

[6] A.Gupta and P. Anand, "Focused web crawlers and its approaches," in Proceedings of the Futuristic Trends on Computational Analysis and Knowledge Management, Noida, February 25-27, 2015, pp. 619-622.

[7] S. Chakrabarti, M. Van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific Web resource discovery," Comput. Netw. Netherlands, vol. 31, pp. 1623-1640, May. 1999.

[8] S. Pandey and C. Olston, "User-centric web crawling," in Proceedings of the 14th international conference on World Wide Web, Chiba, Japan, May 10 - 14, 2005, pp. 401-411.