DGWC: Distributed and Generic Web Crawler for Online Information Extraction

Lu Zhang[†], Zhan Bu[†], Zhiang Wu[†] and Jie Cao[*†]
[†]*Jiangsu Provincial Key Lab. of E-Business, Nanjing University of Finance and Economics, China*
[*]*Corresponding author: Jie.Cao@njue.edu.cn*

***Abstract*—Online information has become important data source to analyze the public opinion and behavior, which is significant for social management and business decision. Web crawler systems target at automatically download and parse web pages to extract expected online information. However, as the rapid increasing of web pages and the heterogeneous page structures, the performance and the rules of parsing have become two serious challenges to web crawler systems. In this paper, we propose a distributed and generic web crawler system (DGWC), in which spiders are scheduled to parallel access and parse web pages to improve performance, utilized a shared and memory based database. Furthermore, we package the spider program and the dependencies in a container called Docker to make the system easily horizontal scaling. Last but not the least, a statistics-based approach is proposed to extract the main text using supervised-learning classifier instead of parsing the page structures. Experimental results on real-world data validate the efficiency and effectiveness of DGWC.**

***Keywords*-Public Behavior Analysis; Online Information; Distributed Web Crawler; Main Text Extraction; Supervised-learning classifier**

I. Introduction

Internet currently has become an important information carrier, especially with the rapid growth of Web 2.0 and social media. Online data therefore turns to be one of the most important source to analyze the public opinion and behavior, which is significant for social management and business decision [1]. For example, companies expect to collect and summarize the comments from customers to predict the market prospect, and/or analyze the competitors information to improve the targeted marketing strategies. However, as the enormous amount of web sites, it is daunting or even impossible to gather the online data manually.

Web crawler is generally the most effectively approach to collect online data from web pages automatically. As is well known, a web page is constructed by two types of contents, i.e. meaningful information (maybe texts or multimedia elements) and links to other web pages. The crawler extracts expected data from the meaningful information, and travels to others web pages according to the links to extract more information. It theoretically can cover all web pages in Internet, however, in practice only a limited subset of web sites is enough, and the crawler is usually restricted to access some targeted web sites. Although various web crawlers are proposed for online information extraction [2][3][4], there still exists some problems: i) The number of targeted web pages is usually very large despite it is limited, thus the efficiency of web crawlers is so important that few current open source crawlers can completely meet the requirements; ii) Information extraction needs to parse the structure of web pages, while the web pages are heterogeneous and analyze them individually is time consuming and even impossible.

To address these problems, a **D**istributed and **G**eneric **W**eb **C**lawer (DGWC for short) is proposed in this paper for web information extraction. First, to improve the efficiency of web crawler, we design a distributed crawler system based on `Scrapy`[1], a widely used crawler implemented by Python. Furthermore, we also consider the ability of horizontal scale-up, and employ `Docker` as the containers to encapsulate the crawlers, thus they can be easily developed to new servers with little configuration. For heterogeneous web pages parsing, we note that some meta data, such as title and time, can be extracted by special HTML tags or regular expression. Therefore, the only problem is how to extract the main text in web pages, thus we proposed a statistics-based approach which employs supervised-learning classifier to identify the main text line, and then uses sliding window and score function to extract main texts from web pages.

The rest of this paper is organized as follows: Section II introduces the framework of DGWC. Section III describes the algorithmic details of main texts extraction. The experimental results are presented in Section IV. We also give the related work in Section V and finally conclude this paper in Section VI.

II. System Overview

DGWC is a distributed web crawler based on `Scrapy`, however, many mechanisms are redesigned to meet the requirements of distributed job scheduling, horizontal scale-up and management. In this section, we briefly introduce the framework of DGWC.

A. Job scheduling of DGWC

For distributed systems, job scheduling usually is one the most important issues to determine the success or failure of the system. In DGWC, a job means a URL that the corresponding web page must be downloaded and parsed. In order to parallel run the jobs, the links that parsed out from one web page by a spider should be shared with other spiders (In this paper, the word "spider" means the module that

[1] http://scrapy.org/

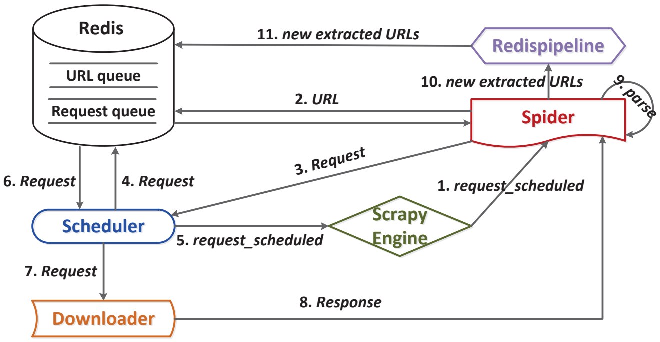

Figure 1: Job scheduling in DGWC

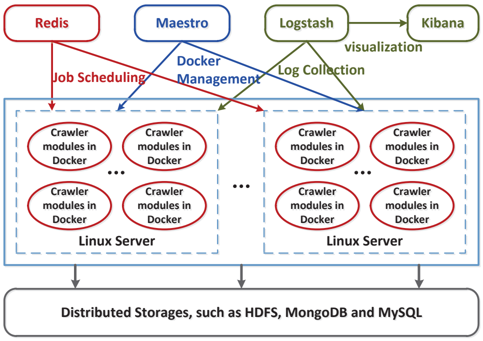

Figure 2: Horizontal scale-up and management for DGWC

parsing web pages in crawler). Scrapy contains a scheduler yet no mechanism is provided for links sharing. In DGWC, we redesign the scheduler and get the links from a shared storage instead of spider.

As shown in Fig. 1, we take Redis, a memory based database, as the shared storage of URLs to obtain higher performance. The job flow of DGWC is as follows: The Scrapy Engine first sends the signal *request_scheduled* to a spider to start the job (Step 1); Then the Spider pops a URL from URL queue in Redis, which will be encapsulated as a *Request* and submitted to the Scheduler (Step 2,3); After the Scheduler receives the *Request*, it pushes this *Request* into the tail of the request queue in Redis, upon sends a signal *request_scheduled* to Scrapy Engine (Step 4,5); To get the content of web pages, the Scheduler pops a *Request* from the head of request queue, and sends it to the Downloader (Step 6,7); The Downloader accesses the web page in terms of the *Request* to get the *Response*, and then submits the *Response* to the corresponding Spider (Step 8); The Spider parse the HTML content in the *Response* to extract expected items, and send the *new extracted URLs* to Redispipeline (Step 9,10); The Redispipeline at last push these *new URLs* into the URL queue in Redis.

Note that in Redis we can use different keys to separate different queues, so that the multiple jobs will not be confused by using unique key among each other.

B. Scalability and management of DGWC

To solve the problem of the large amount of web pages, it is better for the crawler system to parallel access the different web pages. With the shared storage, i.e. Redis, spiders could parallel access it to get jobs. However, if new servers are added to scale up the system, the mass and trivial configurations are time consuming and daunting. In DGWC, we employ Docker, a LXC (Linux Container) based application container engine, to package the crawler program and its dependencies in a virtual container. Therefore the crawler and dependencies can be easily copy to any new Linux servers as a whole, thus get the satisfactory capability of horizontal scale-up.

As shown in Fig. 2, we leave Redis out of container as a global storage to assign jobs, and package the other modules of a crawler into Docker, including Scheduler, Scrapy Engine, Spider, Downloader and Redispipeline. To centralized manage the Dockers in all the servers, we implement Maestro as a controller to add, delete, start and terminate the Dockers. In addition, we use Logstash to gather the running logs and visualized them by Kibana.

The data extracted from web pages by the crawler could be stored in various storages, including HDFS, MongoDB, MySQL and so on. However, the final selection of storage must be carefully considered and make sure it is suitable for the feature of extracted data.

III. Information Extraction

The web pages got by the Downloader module represent as a set of HTML tags, meaningful information and links to other web pages. The spider should parse the web pages to extract expected items, such as title, published time and other information. Furthermore, the URL of links should also be identified to get other web pages. Generally, we construct the web page as a DOM tree and employ DOM based tools, e.g. BeautifulSoup to parse out expected items. However, the structures of web pages are quite different from each other so that it is impossible to parse the web pages using unified program. Fortunately, the items such as title, time and URLs can be identified by special HTML tags (e.g. <title> for title, or <a href> for URL) or regular expressions, therefore the left problem is that how to extract other expected information. Due to the most scenario that usually only the main texts of web pages are needed, we propose an algorithm focus on this simplified problem, which has been implemented in DGWC for main texts extraction.

A. The Overview of the Main Text Extraction Algorithm

To extraction the main texts, we take lines in HTML source as basic units. The line contained in main text is called topical line in this paper, thus the main tasks of the algorithm are 1) to identify potential topical lines in original HTML source; 2) to segment web page into several

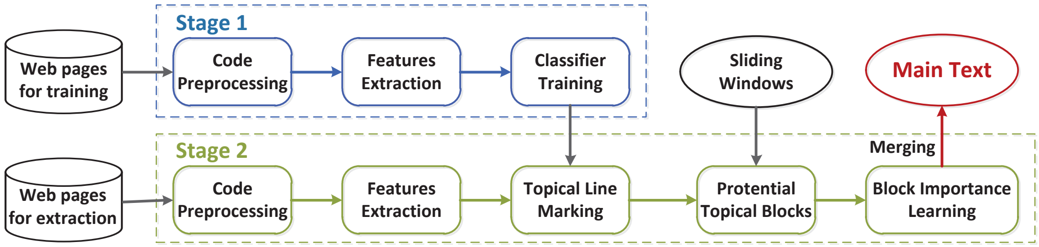

Figure 3: Overview of the main text extraction algorithm

potential topical blocks. Fig. 3 shows an overview of our algorithm, which involves two separate stages: supervised learning classifier construction from training pages and main text extraction by the classifier on target pages. In stage 1, the original HTML source needs necessary preprocessing and the text line features are extracted to represent every text line; then, a supervised learning is practiced to train the classifier using training pages. In stage 2, necessary HTML source preprocessing and text line features extraction are the same as in stage 1, after that, each text line is determined whether it belongs to the main text by the classifier; next, a sliding window is applied to segment web page into several potential topical blocks; finally, a simple election algorithm is utilized to select important blocks which will be united as main text.

B. Data Preprocessing and Features Extraction

In todays web pages, the growth in presentation elements such as tokens, phrases, and named-entities coming from advertising sections and footnotes, are useless yet increases difficulty in main texts extraction.To improve the quality of the extraction results, those presentation elements need filtering according to the following criteria:

1) Get the string between <BODY> tags;
2) Delete all blank lines and redundant white-spaces;
3) Delete HTML tags listed in Tab. I, the contents between which are always noisy information. Note that the tag <a> is improtant for links identifying yet useless for main texts extraction.

After the above three steps, we obtain the web page HTML source with little noisy information.

Typically, the main text usually occupies the center of the web page, and this part of code has intensive text with few links or images. With previous tasks, we split the extracted text into a string sequence of N lines, denoted by an ordered set $L = \{P_1, P_2, \cdots, P_N\}$, where P_i is the ith text line in web page. Therefore, some spatial and content features in a text line are useful to differentiate text line importance. As the spatial features are difficult to capture from the HTML source, in the paper, we only consider the edit distance between the line text and the documents <BODY> tag, namely, the index in set . The content features in a text line could reflect the importance of this text line. In this paper, the selected features are the length of the text line (measured in HTML bytes), the length of the output text line (measured in HTML bytes), the density of it, the number of links and the number of images. We represent the features of a text line as $\{Index, TextLength, OutputTextLength, Density, LinkNum, ImgNum\}$. For example, if we get a text line which leaves 287 lines from the <BODY> tag, and has 198 bytes totally and 170 bytes outputted, and has no link and image, could be represented as $\{287, 198, 170, 0.86, 0, 0\}$, where the density 0.86 is calculated by 170/198, i.e., the ratio of *OutputTextLength* and *TextLength*. Note that if the training data is composed of a number of HTML source files which are from different web sites, the spatial feature such as *Index* needs to be measured using the relative distance (normalized distance), i.e., i/N; where i is the original Index value and N is the total number of text lines in the document. The other features can be normalized in a similar way.

Table I: Some useless tags in HTML source files

Useless HTML tags
<a>, <script>, <noscript>, <style>, <meta>, <!—->, <param>, <button>, <select>, <optgroup>, <option>, <label>, <textarea>, <fieldset>, <legend, <input>, <image>, <map>, <area>, <form>, <iframe>, <embed>, <object>

Although a classifier could be trained by the normalized features above. However, almost all features are numerical, and modeling the continuous feature directly is not suitable for main text extraction. The reason is that we do not have any prior knowledge about the feature distribution between positive and negative instances over the entire data. Therefore we propose an heuristic approach to discretize each continuous feature. First, we sample some text lines from the the center of the web page and mark the sampled text regions as topical region. Then the feature's weighted minimum/mean/maximum values, denote as *F.MIN*, *F.MEAN* and *F.MAX* are calculate according to Eq. (1)(2)(3) over the observed values in the topical region (tr) and non-topical regions (non-tr).

$$F.MIN = \alpha \cdot F.MIN^{tr} + (1-\alpha) \cdot F.MIN^{non-tr} \tag{1}$$

$$F.MEAN = \alpha \cdot F.MEAN^{tr} + (1-\alpha) \cdot F.MEAN^{non-tr} \tag{2}$$

$$F.MAX = \alpha \cdot F.MAX^{tr} + (1-\alpha) \cdot F.MAX^{non-tr} \tag{3}$$

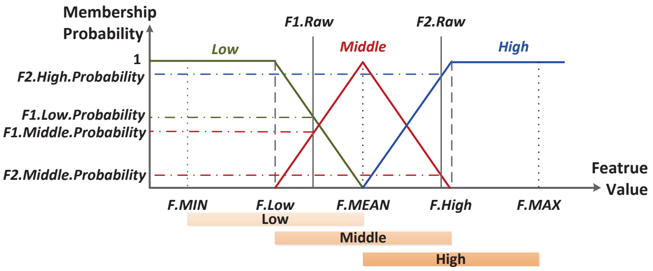

Figure 4: The discretized function of features

where α is a weight value for balancing the importance between the topical region and the non-topical regions. Here, we set the value of α to 0.5. Based on the value of *F.MIN*, *F.MEAN* and *F.MAX*, we then calculate two cut points *F.Low* and *F.High*, where *F.Low* is the average of *F.MIN* and *F.MEAN*, and *F.High* is the average of *F.MEAN* and *F.MAX*. Now, the raw feature values can be represented by their probabilities of falling into three scopes: Low, Middle and High, respectively, as shown in Fig. 4. Note that each feature could only has two non-zero values in Low, Middle and High, and then, we take the larger one as the discretized value of this feature.

C. Topical block segmentation and main text identifying

Once the features are extracted, we can construct a supervised learning classifier with a training set that the positive and negative instants are labeled manually. Various typical machine-learning based classifiers could be employed, including Naive Bayes, C4.5, SVM and so on. The classifier can determine whether a given text line belongs to main text, and the text lines belongs to main text is called topical lines. This initial classification, though rough, labels most of the text lines correctly. However, if there are lengthy copyright notices, comments, and/or descriptions of other stories (not part of the main text), then those will likely to be labeled as topical lines too. Also, if there are descriptions around inline graphics that are part of some advertisement, or lengthy textual advertisements, these may also be labeled as topical lines. False negatives could also be observed when a topical line is not sufficiently long. To address these issues, a sliding window technique is firstly utilized to segment a web page to several potential topical blocks. The process is described below: First, we associate the i^{th} text line with a Boolean variable M_i (TRUE represents that the line is a topical line and FALSE otherwise) according to the classifier in the previous step; and then, scanning the entire HTML source file from top to bottom with a sliding window. The k^{th} potential topical block is represented as $B_k(start_k, end_k)$, where $start_k$ is the start position of the block, and end_k marks where it ends. $B_k(start_k, end_k)$ must satisfy the following conditions:

1) $M_i = FALSE$, if $start_k - \phi \le i \le start_k$ or $start_k - \phi \le i \le start_k$, where ϕ is the length of the sliding window which is empirically set to 5 generally;
2) $M_i = TRUE$, if $i = start_k + 1$ or $i = end_k - 1$;
3) $\underset{start_k<i<j\le end_k}{\text{MAX}} \quad d_{ij} \le \phi$, where M_i and M_j are TRUE.

In other words, in a topical block, no more than $\phi - 1$ continuous non-topical lines can be included. This way, some false negatives from topical line detection can be tolerated.

After extracting potential topical blocks, the proposed algorithm identifies the most informative blocks. An informative block contains meaningful information that would be the target of main text extraction. The other blocks that contain noise information such as advertisements, menus, or copyright statements are considered non-informative blocks. A topical line of a web page usually has a high density of texts with few links or images. Thus we can use a simple Score to measure the informativeness of every HTML line:

$$Score(P_i) = |T_i| + |O_i| + |D_i| - |L_i| - |I_i| \qquad (4)$$

where $|T_i|, |O_i|, |D_i|, |L_i|, |I_i|$ are the value of *TextLength, OutputTextLength, Density, LinkNum, ImgNum*, respectively, and normalized in the range of [0,1] as mentioned in Sec. III-B. if $Score(P_i) > 1.5$, the line will be considered a topical line. To recognize informative blocks, we can use the average length of the plain text lines (T), the average length of the output text line (O), the average ratio of plain text to HTML bytes (D), the average number of links (L), and the average number of images (I). The following heuristic rule is used for evaluating the informativeness of a topical block:

$$Score(B_k) = \frac{\sum\limits_{Start_k \le i \le End_k} Score(P_i)}{End_k - Start_k + 1} \qquad (5)$$

If $Score(B_k) > 1.5$, the block will be considered an informative block, and all the informative blocks will form the main text. It is worth noting that this algorithm can help reduce false positives from topical line detection. A long line of advertising text and some descriptions of the other stories and pictures, which could be marked as topical lines in the previous step, are likely to be more isolated, i.e., more likely to be surrounded by other non-topical text lines, than a typical topical line. Therefore, the Score would be relatively lower for any block that contains that line, when compared with the score of a typical topical block where true topical lines tend to stay closely together.

IV. Experimental Results

In this section, we present experimental results of DGWC. Both the performance and the effectiveness of main text extraction are evaluated. All experiments were performed on a cluster with 8 nodes connected by Gigabit Ethernet. Each node comes with four quad-core E5-2650v2 processors (2.6GHz), 128GB of RAM, 240GB of SSD disk and 600GB of SAS disk.

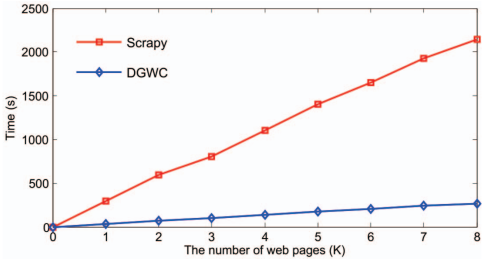

Figure 5: Comparison on spent time between DGWC and Scrapy

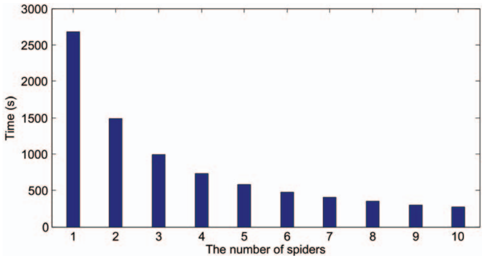

Figure 6: Impact of the spider number on spent time

For performance evaluation, we take the task that to get eight thousand web pages from eight web sites. The sites are listed in Tab. II that half of them are Chinese web sites and the other half are English web sites. In each web site, we get one thousand pages and parse them by `BeautifulSoup` because it is more stable for performance comparison. We first compare the DGWC and standalone version of `Scrapy` by time spent, here the `Scrapy` is run on one node of the cluster and the number of DGWC spiders is 10 (Note that the spider is packaged by `Docker` and one node can run multiple `Docker` packets). To exclude the interference of network delay, we only count the time between response-arrived and parsing-completed. The result is shown in Fig. 5, from which we can see that the DGWC spent obvious less time than `Scrapy` standalone version. Note that the spent time ratio of DGWC and `Scrapy` is a little larger than that of one and spiders number, this is due to the jobs scheduling by shared `Redis` consume some time. However, the results of this experiment are sufficient to validate the efficiency of DGWC. Furthermore, we count the spent time from one spider to ten spiders, as shown in Fig. 6. We can see the speed of time reducing is fall behind the speed of the spider number increasing, which also demonstrate that scheduling jobs between spiders will spent additional time. However, it is worthy that the total time spent in parsing web pages is indeed decreasing.

Table II: Comparison of main text extraction methods

Web Sites	Our Algorithm			WISDOM	VIPS
	P	*R*	*F*	*F*	*F*
www.sohu.com	0.976	0.987	0.981	0.955	0.832
www.163.com	0.973	0.989	0.981	0.926	0.771
www.sina.com.cn	0.943	0.967	0.955	0.912	0.692
www.qq.com	0.931	0.952	0.941	0.916	0.875
edition.cnn.com	0.951	0.979	0.965	0.931	0.856
www.nydailynews.com	0.912	0.943	0.927	0.926	0.801
www.newsday.com	0.961	0.983	0.972	0.942	0.855
www.bbc.com	0.955	0.973	0.964	0.933	0.792

For the effectiveness valuation of main text extraction, we first select 100 web pages from each web site's 1000 web pages that got in the performance experiment. Then, we construct a dataset with total 800 web pages, and label the main text manually. The main text extraction algorithm proposed in Sec. III with SVM as classifier is utilized, and a 10-flod cross validation is applied to evaluate the effectiveness. We adopt the metrics in [5] to assess our results, which include precision, recall and F-measure. A higher value of precision indicates fewer wrong classifications, while a higher value of recall indicates less false negatives. They are calculated as follows:

$$\boldsymbol{Precision} = \frac{|bag(C) \cap bag(MT(WP))|}{|bag(C)|} \tag{6}$$

$$\boldsymbol{Recall} = \frac{|bag(C) \cap bag(MT(WP))|}{|MT(WP)|} \tag{7}$$

where $bag(C)$ denotes the bag of output text/contect associated with a chunk of text C. $|bag(C)|$ is the length of output text/content (measured in HTML bytes) in C. $MT(WP)$ is the main text of a web page *WP*. It is common to use the harmonic mean of both measurements, called F-measure, such as the *F1-measure* defined by Eq. (8) which weighs precision and recall equally important.

$$\boldsymbol{F}1 - measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{8}$$

To validate the effectiveness, we compare our algorithm with two well-known existing DOM based approach WISDOM [6] and Vision-based approach VIPS [7] in accuracy. The results are shown in Tab. II, from which we can see our algorithm achieves high accuracy in main text extraction.

V. Related Work

Various web crawlers such as `Larbin` [2], `Heritrix` [3] and `Scrapy` [4], have been designed for automatically online information extraction. However, few crawler systems are considered carefully for parallel implement and run to meet the requirement of massive data extraction. Besides the performance, another problem bought by the large amount of web sites is pages parsing. In recent years, a large number of researches have addressed to automatically identify information from web sources [8]. Differentiated by their scopes, these works can be categorized into Document Object Model (DOM) based [6][9], vision-based [7][10], and statistics-based [11][12] approaches. Among the three methods, the first two both need to render the page, and exploit specific extraction strategies during the extraction process. They are indeed unsuit for the massive heterogeneous web pages, thus only the statistics-based approaches can be used in large-scale crawler systems. The exist statistics-based approaches usually use density [11] or word count [12] as statistics objects, however, more features in web pages could be used to achieve better accuracy. In this paper, we propose a distributed and generic web crawler system to parallel access and parse web pages. And furthermore, a self-designed main texts extraction algorithm is integrated to parse web pages uniformly.

VI. Conclusion

This paper presents the design of a distributed and generic web crawler system (DGWC) for online information extraction. To improve the performance, we take `Redis` as a shared storage to schedule jobs among multiple `Scrapy` spiders in order to parallel access and parse web pages. Furthermore, the crawler program and dependencies are packaged as a `Docker` container to make the crawler system could be easily horizontal scaling. To extract main text from various heterogeneous web pages, we propose a statistics-based two stages approach. We first extract a set of features and construct a supervised-learning classifier, then identify topical lines using this classifier. To improve the accuracy, a slide window is utilized to recognize potential topical block and extract the final main text based on block score. The experimental results show that DGWC has satisfactory performance to extract meaningful information from massive heterogeneous web pages.

Acknowledgment

This research was partially supported by National Natural Science Foundation of China under Grants 61502222, 71571093 and 71372188, National Key Technologies R&D Program of China under Grants 2014BAH29F01, National Center for International Joint Research on E-Business Information Processing under Grant 2013B01035, Industry Projects in Jiangsu S&T Pillar Program under Grant BE2014141, Natural Science Foundation of Jiangsu Province of China under Grant BK20150988, Surface Projects of Natural Science Research in Jiangsu Provincial Colleges and Universities under Grants 15KJB520012 and 15KJB520011, and Pre-Research Project of Nanjing University of Finance and Economics under Grants YYJ201415.

References

[1] Q. S. Zhang, B. L. Xie, and X. M. Zhang, "Uncertain internet public opinion emergency decision-making method under interval-valued fuzzy environment," in *Applied Mechanics and Materials*, vol. 713. Trans Tech Publ, 2015, pp. 2024–2028.

[2] M. Yan and S. Wang, "System architecture of larbin web crawler," *Computer Study*, vol. 4, p. 045, 2010.

[3] G. Mohr, M. Stack, I. Ranitovic, D. Avery, and M. Kimpton, "An introduction to heritrix an open source archival quality web crawler," in *In IWAW04, 4th International Web Archiving Workshop*. Citeseer, 2004.

[4] J. Wang and Y. Guo, "Scrapy-based crawling and user-behavior characteristics analysis on taobao," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on*. IEEE, 2012, pp. 44–52.

[5] E. S. Laber, C. P. de Souza, I. V. Jabour, E. C. F. de Amorim, E. T. Cardoso, R. P. Rentería, L. C. Tinoco, and C. D. Valentim, "A fast and simple method for extracting relevant content from news webpages," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1685–1688.

[6] H.-Y. Kao, J.-M. Ho, and M.-S. Chen, "Wisdom: Web intrapage informative structure mining based on document object model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 5, pp. 614–627, 2005.

[7] J. Kang, J. Yang, and J. Choi, "Repetition-based web page segmentation by detecting tag patterns for small-screen devices," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 980–986, 2010.

[8] Z. Bu, C. Zhang, Z. Xia, and J. Wang, "An far-sw based approach for webpage information extraction," *Information Systems Frontiers*, vol. 16, no. 5, pp. 771–785, 2014.

[9] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm, "Dom-based content extraction of html documents," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 207–214.

[10] D. Cai, S. Yu, J.-R. Wen, and W.-Y. Ma, "Extracting content structure for web pages based on visual representation," in *Asia-Pacific Web Conference*. Springer, 2003, pp. 406–417.

[11] J. C. Alex, "The easy way to extract useful text from arbitrary html," 2007.

[12] B. Zhou, Y. Xiong, and W. Liu, "Efficient web page main text extraction towards online news analysis," in *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*. IEEE, 2009, pp. 37–41.