ISSN 2278-3091
Volume 3, No.5, September - October 2014
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://warse.org/pdfs/2014/ijatcse03352014.pdf

Implementation of a Lucene based Blog Search Engine*

Zhanying Jin[1], Sujoung Oh[2], Minsoo Lee[3]
[1]Dept. Computer Science and Engineering, Ewha Womans University, Korea, jxy5130@gmail.com
[2]Dept. Computer Science and Engineering, Ewha Womans University, Korea, sujoung0719@naver.com
[3]Dept. Computer Science and Engineering, Ewha Womans University, Korea, mlee@ewha.ac.kr

Abstract : We combined the frameworks Lucene and Heritrix, to design a restaurant blog search system. We crawl the Citysearch web page using Heritrix and try to extend the FrontierScheduler class which is part of Heritrix. In order to extract the full text without html tags, we parse the crawled html pages with Jsoup. We then pre-process the crawled pages to eliminate unnecessary contents via Web page purification. We build the indexes for these content using Lucene in Eclipse, and extend the ranking as well as implement the search UI page with JSP.

Key words : blog, crawling, Lucene, ranking, search

1. INTRODUCTION

The Restaurant search system can be divided into four main parts, namely the data collection module, Web pre-processing module, indexing module, and search module. Among them, the page pre-processing module includes another sub-module which is the module to perform the purification of Web pages. The main contributions of this paper can be outlined as follows:

First, we introduce how the Web page crawler Heritrix [1] works, and the details of how it can be used to crawl Web pages. We added a URL matching function to make it crawl the much more adequate pages which are extracted.

Second, we introduced the Web purification technique using HtmlParser, and gave a complete algorithm of how to use the HtmlParser to parse a page in order to achieve the purpose of purification. Web page elimination has always been an indispensable part in search engine systems. We described the purification method which was used as well.

2. RELATED RESEARCH

2.1 Web Crawler

Web crawlers are programs that exploit the graph structure of the Web to move from page to page. The web crawler's responsibility is to travel from the seed domains to their linked pages and then go further to these linked pages ' linked pages. In this way, the crawler can visit almost the whole Internet or visit almost all the assigned Web domains [2][3]. Heritrix is a web crawler designed for Web archiving. It's written in Java and provides a free software license. The main interface is accessible using a Web browser, and there is a command-line tool that can optionally be used to initiate crawls.

2.2 Web page Purification

Web pages are written in HTML consisting of plain texts, tags and links to image, audio and video files, and other pages. The html pages consist of tags. Most HTML tags work in pairs. Each pair consists of an open tag and a close tag indicated by < > and </> respectively. Jsoup [4] is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data [5]. We use Jsoup to eliminate the html tags.

2.3 Web-based search UI

Once a search index is created, usually a simple search user interface is provided to use the index. A Web-based user interface can be built using JSP. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

3. SYSTEM DEVELOPMENT

3.1 Data Extraction

We collect html files from Citysearch [6] using Heritrix. Citysearch is an online city guide that provides information about businesses in the categories of dining, entertainment, retail, travel, and professional services in cities throughout the United States. Visitors to each of the local city guides in Citysearch will find contact information, maps, driving directions, editorial, and user reviews for the businesses listed. The Main page of searching for a restaurant is shown in Figure 1.

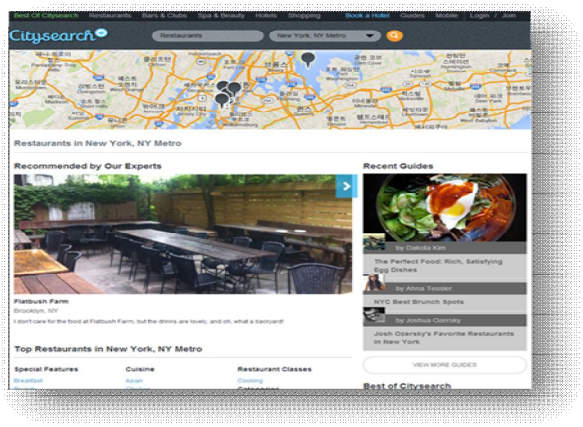

Figure 1: Restaurant search page of CitySearch[3]

* This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(No. 2012R1A1A2006850).

This work was also supported by the National Research Foundation of Korea grant funded by the government (Ministry of Education, Science and Technology) (NRF-2012M3A9D1054744).

After downloading Heritrix from the URL http://crawler.archive.org/downloads.html, we imported it into Eclipse and run the project. At this point, the Heritrix server is running in the background and is listening to port 8080 and can be accessed through the URL http://localhost:8080 through the browser. We could then enter the user name and password which was set in the Heritrix.properties configuration file and create a crawler job and run the job as shown in Figure 2.

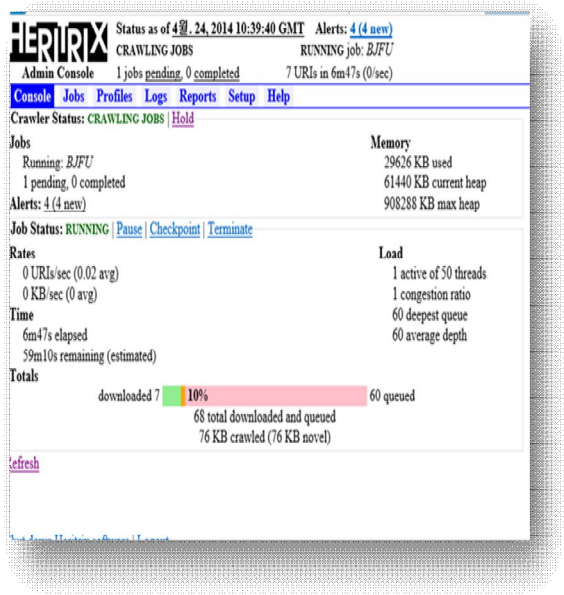

Figure 2: Creating a crawler job with Heritrix [1]

We improved the main classes of the FrontierScheduler as shown in Figure 3. The scheduler method was extended to include the URL matching constraints, namely only when the conditions are satisfied the URL will be transferred to Frontier in a scheduled manner. Finally, as shown in Figure 4, the extension of the module prohibits certain results from joining the results after the collecting and gathering is done.

```
public void shedule(CandidateURI caUri){
    String url = caUri.toString();
    try{
        if(url.indexOf("restaurant")!=-1||
                url.endsWith(".jpg")||url.indexOf("robots.txt")!=-1||url.indexOf("dns:")!=-1){
            getController().getFrontier().schedule(caUri);
        }else{
            return;
        }
    }catch(Exception e){
        e.printStackTrace();
    }finally{

    }
}
```

Figure 3: Customizing the crawling task

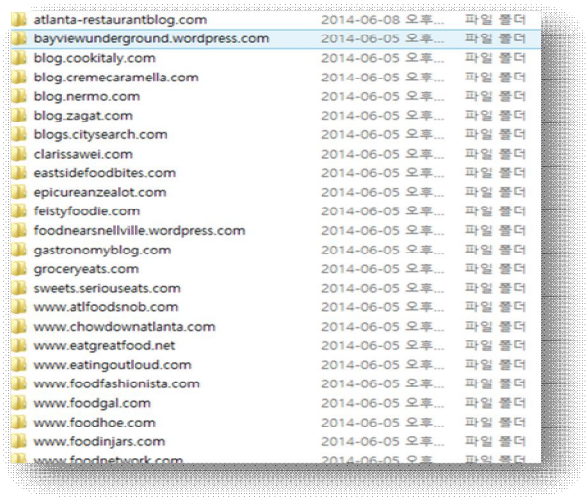

Figure 4: Resulting file directory after crawling

3.2 Web page purification

After Web page crawling is completed, the next step is to pre-process the page. Web crawlers take the original page which contain a lot of useless information, such as advertising, and there exist a large number of Websites with duplicate pages. While you browse the Web, you will find pages not only giving you the information you want, but also with a lot of useless information such as advertising links, navigation bar, etc. Using Jsoup can solve this problem to extract undesired content from html pages.

```
public static String extractText(Reader reader) throws IOException {
  StringBuilder sb = new StringBuilder();
  BufferedReader br = new BufferedReader(reader);
  String line;
  while ( (line=br.readLine()) != null) {
    sb.append(line);
  }
  String textOnly = Jsoup.parse(sb.toString()).text();
  return textOnly;
}
```

Figure 5: Code for Web page purification with Jsoup[4]

3.3 Index Creation and UI Implementation

When pre-processing is completed, the next step is to build indexes with Lucene [7][8][9]. The code for building indexes is relatively simple. Once the building indexes is completed, we implement the simple search page with JSP. Building of the indexes and the simple Web-based UI for the search pages are shown in Figure 6 and Figure 7, respectively.

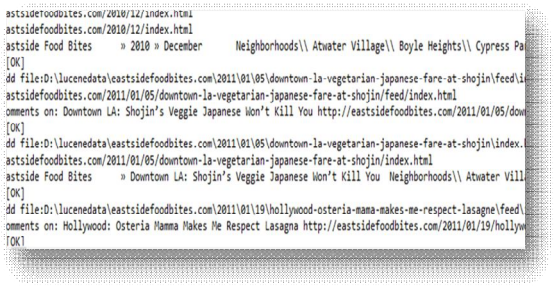

Figure 6: Building the search index

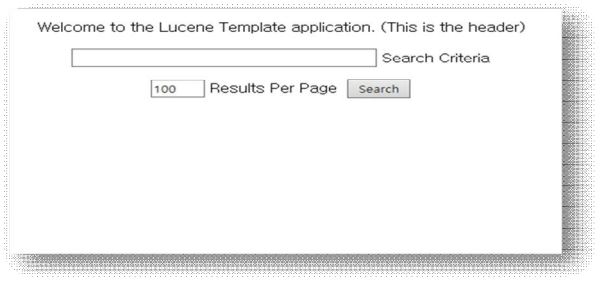

Figure 7: Main search page

3.4 Enhancing the Ranking

The following is the scoring formula which is used to sort the results in Lucene [10] :

$$score(q,d) = coord(q,d) * queryNorm(q) * \sum_{t \text{ in } d} (tf(t \text{ in } d) * idf(t)^2 * t.getBoost() * norm(t,d))$$

Boost is the indexing factor for each excitation field which is set to the default value 1.0. The higher boost value represents the more importance of term t. There are two ways to set the value for the boost. One is set in the index, and the other is in the query item. When setting the boost value while building the index, Field.setBoost (boost) indicates that the field is more important than the other fields; Document.setBoost (boost) indicates that the document is more important than the other documents

When building indexes the count for like in Facebook can be viewed as somewhat related to the boost value. The Boost value of a document is set to the default value (1.0) and we can add the value of the like count in Facebook. After this modification is done, a document with a high like count by others will also have a higher boost value. And the higher the score of the document, the more likely it will be in the top ranked results.

Figure 8: Result page of search

4. CONCLUSION AND FUTURE WORK

We have proposed a search engine for blogs by combining the Lucene and Heritrix frameworks. We extend the FrontierScheduler class which is part of Heritrix, and crawl the Web pages using this extended module of Heritrix. We then pre-process the crawled pages to eliminate unnecessary contents via Web page purification using Jsoup. We build the indexes for these content using Lucene in Eclipse, and extend the ranking as well as implement the search UI page with JSP.

For future research directions, when parsing the crawled html pages, we can further extract the text and check the similarity with the title and use it to calculate the similarity. We could further fine-tune and enhance the ranking idea of the Facebook like count. Other approaches to extend the scoring formula in Lucene can also be explored by investigating each component in the scoring formula.

REFERENCES

1. Heritrix, [Online] Available: http://crawler.archive.org/index.html
2. Web Crawler, Wikipedia, [Online] Available: http://en.wikipedia.org/wiki/Web_crawler
3. V. Shkapenyuk and T. Suel. **Design and implementation of a high performance distributed web crawler**, In Proc. 18th Int'l Conf. Data Engineering, San Jose, California, 2002, pp. 357-368.
4. Jsoup, [Online] Available: http://jsoup.org
5. N. Derouiche. **Automatic Extraction of Structured Web Data with Domain Knowledge**, in Proc. 28th Int'l Conf. Data Engineering, Washington, DC, 2012, pp. 726-737.
6. Citysearch, [Online] Available: http://www.citysearch.com
7. Lucene, [Online] Available: http://lucene.apache.org
8. R. Gao, D. Li, W. Li, and Y. Dong. **Application of Full Text Search Engine based on Lucene**, Advances in Internet of Things, Vol. 2 No. 4, pp. 106-109, 2012. doi: 10.4236/ait.2012.24013.
9. Y. C. Li and H. F. Ding. **Research and Application of Full-Text Search Engine Based on Lucene**, Computer Technology and Development, Vol. 20, No. 2, pp. 4-56, 2010.
10. X. Zhang and Y. Zhou. **Improvement of an Algorithm for Ranking Pages Based on Lucene**, Computer System Application, Vol. 18, No. 2, pp. 155-158, 2009.