1C1-R-5-1

Community Driven Search Engine Based on Community's Proxy Server Log

Guntur Dharma Putra[*1] Ferry Astika Saputra[*2] Kenzi Watanabe[*3]

[*1]Graduate School of Science and Engineering, Saga University

[*2]Department of Information Technology, Electronic Engineering Polytechnic Institute of Surabaya

[*3]Graduate School of Science and Engineering, Saga University

In this paper, we are introducing a method to improve search engine capabilities by using user preference achieved with the help of community's proxy logs. The goal is focused to build a custom search engine that providing community-specific results. To achieve such search engine, we use proxy server logs from Network Operation Center of EEPIS-ITS and fetch the unique URL and user field as raw data. Getting the needed data, then we crawl the title and meta information from all of the unique URLs. Then, document vector is created in order to make those textual data turn into a machine-friendly numerical data. To find the topic, based on those URLs and its meta information, we cluster it into 10 or more preferable clusters using *k*-means algorithm. Those results, finally, would be our base to create the search engine and we use vector space model to provide a search result from user's query.

1. Introduction

Search engine is a key tool for finding information in World Wide Web (WWW). According to the performed research studies, number of web sites on the Internet is above 156 million [Yalçin 10]. Thus, the exponential growth of the number of WWW users and its application as well as the rapid growth of WWW objects makes though challenges to all search engines.

Typical search engines provide the same results for a given query independent of the user or the situation in which the query is being issued [Almeida 04, Taghavi 11]. As a result, some of users will be unsatisfied because the relevance of each information retrieved is highly independent in the context in which query is entered. For example, results for the query "Squid" will depend whether the user is seeking information about a marine creature or interested on the technology of web-caching server. This might be happen because of the original design of the search engines are built to provide answers to all query requests regardless who the users are. But in fact, when users post their query, they send their personal information about their interest [Taghavi 11].

Web-caching servers are widely used in some developing countries in order to seek a way out of limited internet bandwidth problem by reducing access time and bandwidth requirements [Taghavi 11]. Those servers which are usually used in every community are unexpectedly keeping user's information within its server log. Having that proxy logs, actually we could extract user preferences by processing those logs and extracting any useful information.

In this work, we use that opportunity to improve search engine capabilities by user preference achieved with the help of community's proxy logs. Nevertheless, our work is limited to building the search engine only and there was no further testing or performance comparison yet.

The rest of this paper is organized as follows: Section 2 reviews related works. The data and methodology are comprehensively explained in section 3, while section 4 describes the system design overview. Finally, concluding remarks and future work are drawn in section 5.

Contact: Kenzi Watanabe, Graduate School of Science and Engineering, Saga University, +81-952-28-8828, watanabe@is.saga-u.ac.jp

2. Related Works

Within the context of community related search engine topic, for example, a community-aware search engine research [Almeida 04] was carried out in order to make answers to a query become dependent to specific user information need. A novel ranking technique that combines content-based and community-based evidences are used along with the theory of Bayesian belief networks for approaching their goals.

Whilst Almeida's aim to conduct a community-aware search engine is comparably identical with our intention to improve search engine capabilities so it would be making the satisfaction of the user goes up, we are adopting different approach to achieve that aim. Proxy server log processing was extensively used to extract user preference on a community.

Moreover, proxy log processing and analysis were also being researched with query distribution approach for search engine [Taghavi 11]. That research results could be used to develop long to short term business plans for search engine service providers.

3. The Data and Methodology

3.1 The Data

This work was carried out with logs of web-caching server, Squid, collected from Network Operation Center of EEPIS-ITS (Electronics Engineering Polytechnic Institute of Surabaya – Institut Teknologi Sepuluh November), Indonesia. Log files from 3 servers, which are serving as web proxy HTTP server, are collected during a period spanning from August 2011 up to November 2011. This data simply represents the community of EEPIS-ITS campus, Indonesia.

The log files used for this work were derived from Squid's *access.log* file. Those logs contain 10 fields whilst only 2 of them were used in order to extract user preferences:

- URL: The URL (Uniform Resource Locator) is a global address for specifying the location of a resource or a transaction. In this work we only parsed the URL up to the TLD (Top-level domain).
- RFC931: The field contains username who send the requests to the proxy server.

3.2 The methodology

(1) Processing the proxy logs and crawling meta information

This work requires a comprehensive processing of proxy logs in order to extract user's information from raw data. As described above, the needed information is just the URL and RFC931 field. Then we would crawl the meta information of those URLs.

http://0.7.channel.facebook.com/
http://www.facebook.com/
http://0.124.channel.facebook.com/

www.facebook.com

Fig. 1. List of URLs that are pointing to www.facebook.com.

In the real parsing process, there would be lots of URLs that are just pointing to a same URL. That is just simply because lots of components contained within a web page are coming from multiple sources [Yalçin 10].

If only we crawl the meta information from those one pointing URLs, in fact, we got null result and only one of them has the meta information. Thus, those URLs mean nothing and we could wipe all of those unimportant URLs out and just important URLs still remain. In order to achieve it, we implemented white and black-list filtering mechanism:

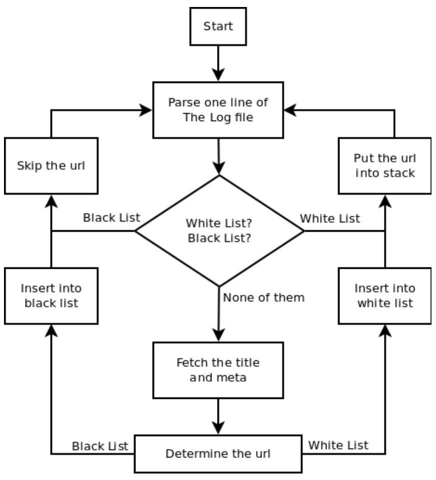

Fig. 2. White and black-list filtering flowchart.

The prior process (filtering) would result a list of important URLs and we could continue to further process of crawling the meta information.

(2) Document Vector and TF-IDF weighting scheme

The TF-IDF (Term Frequency-Inverse Document Frequency) weighting scheme is a numerical statistic which reflects how important a word is to a document in a collection or corpus. While in this work, we have collections of URLs that are simply representing collections of documents. Each of those documents has its own meta information or collection of words.

Before applying TF-IDF weighting scheme, we have to give those data some treatments:

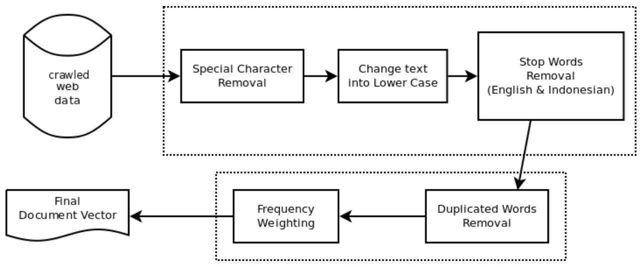
crawled web data
Special Character Removal
Change text into Lower Case
Stop Words Removal (English & Indonesian)
Duplicated Words Removal
Frequency Weighting
Final Document Vector

Fig. 3. Document vector processing flowchart.

Then the TF-IDF is calculated as:

$$weight(i,j) = \begin{cases} (1+\log(tf_{i,j}))\log\dfrac{N}{uf_{i,j}}, & if\ tf_{i,j} > 0 \\ 0, & if\ tf_{i,j} = 0 \end{cases} \qquad (1)$$

$tf_{i,j}$ is the number of occurrences of word w_i in URL u_j, while $uf_{i,j}$ is the number of URLs in the collection in which w_i occurs in.

Having all URLs calculated using the TF-IDF formula, then those numerical values are converted into a matrix $M_{(word,\ url)}$ that later would be used for clustering using *k*-means algorithm and for the vector space model.

(3) *k*-means and Vector Space Model

The Vector Space Model is commonly used structured form for text data in which individual text documents are represented as a set of vectors [Jing 06]. Later the matrix M would be converted into single vector $V_{(word)}$ so that those collection of words could be clustered using *k*-means algorithm. Converting process involves algebraic model of Vector Space Model. Then the vector is calculated as such:

$$V_i = \sqrt{\sum_{j=1}^{n} M_{i,j}} \qquad (2)$$

K-means clustering process will use that vector's values. Then the result of clustering would be our base for grouping the URLs which could be categorized in the same cluster. The *k*-means algorithm flowchart is as follows:

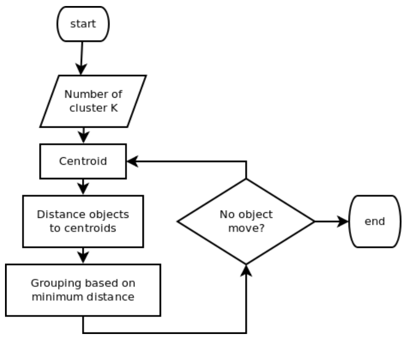
start
Number of cluster K
Centroid
Distance objects to centroids
Grouping based on minimum distance
No object move?
end

Fig. 4. K-means algorithm flowchart.

(4) Vector similarity

To do retrieval in the vector space model, documents are ranked according to similarity with the query as measured by the cosine measure or normalized correlation coefficient. Thus, every document in this collection of documents would be measured the similarity with the search query requested by users.

Firstly, user's query would be treated as a single document and weighted using the same TF-IDF formula (1) as the URLs were computed. Getting the result, those 2 vectors (query *Q* and each URLs *D*) measured by using vector similarity:

$$sim(Q,D) = \frac{\sum_{i=1}^{n} W_{qi} * W_{di}}{\sqrt{\sum_{i=1}^{n} (W_{qi})^2 * \sum_{i=1}^{n} (W_{di})^2}} \qquad (3)$$

W_{qi} is the weight of word W_i in the query *Q*, while W_{di} is the weight of word W_i in the URL *D* (meta information).

The most similar vector, in this case is URL, would have the highest value of formula 3. Finally, the search engine result would be derived from formula 3 written above.

4. System Design Overview

The overall system design is described in the following scheme:

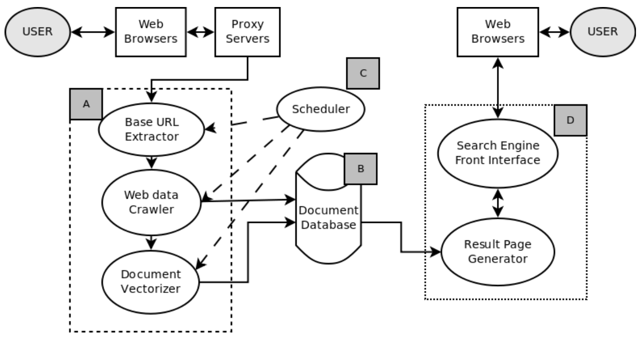

Fig. 5. The overview of our system design.

4.1 The processing Unit

The processing unit, described in the picture as A, plays a role in preparing data needed for the local search engine (D). The processing unit would extract URL and RFC931 field from the Squid *access.log* files, then crawl the meta information of each extracted URL, and finally create a document vector of those crawled data in order to transform those textual data into numerical data. Later on, those prepared data would be tidily stored in the document database (B) for further process.

4.2 Updating Scheme

The document database would be updated in a 24-hour period corresponds to the Squid *access.log* files that is also updated every day, and that is scheduler (C) work for. In the updating scheme, in order not to duplicate the URLs that have been listed on the database, we did some tricky steps:

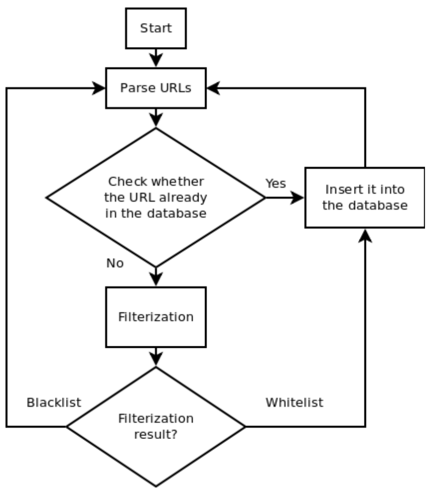
Start
Parse URLs
Check whether the URL already in the database
Yes
Insert it into the database
No
Filterization
Blacklist
Filterization result?
Whitelist

Fig. 6. URL updating scheme flowchart.

Later the new URL list will be re-crawled in order to get new meta information and create a new document vector.

4.3 Local search engine

Firstly, local search engine (D) would extract saved data in document database (B) and measure the similarity between user's query and every document as mentioned in chapter 4.4. To rank the URL in the search result page shown as the final result to the user, local search engine uses calculated vector similarity and URL's TF-IDF value as described in the following graph:

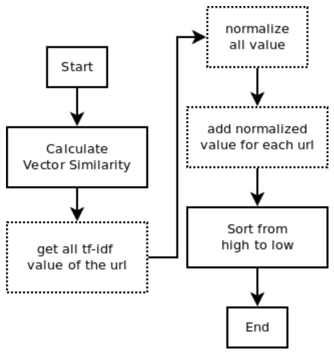
Start
Calculate Vector Similarity
get all tf-idf value of the url
normalize all value
add normalized value for each url
Sort from high to low
End

Fig. 7. Ranking algorithm flowchart.

5. Concluding Remarks and Future Work

In this work, we presented a community driven search engine which its goal is to provide more specific results than the ordinary search engines do. Although we did not compare our community customized search engine with the search engine in the real daily live, we showed that it is possible to create a custom community driven search engine from community's proxy server log.

In order to improve this community driven search engine's capabilities, there are some works that could be carried out particularly:

- Considering a deeper user's query treatment, including a query recommendation, spelling and grammar check.
- Benchmarking this community driven search engine with the search engine on the market would be

particularly important to check this search engine performance.

- To check whether this community search engine runs smoothly and provides exactly what those communities want could be benchmarked using different proxy server logs provided from different communities.
- Using other approaches for modeling the user preference from the communities, such as the theory of Bayesian belief networks, could be carried out in order to perform a further research.

Acknowledgments

The authors would like to thank to Mr. Idris Winarno as the head of Network operation Center of EEPIS for giving access to the real proxy data.

References

[Taghavi 11] M. Taghavi, et al., An analysis of web proxy logs with query distribution pattern approach for search engines, Comput. Stand. Interfaces (2011), doi:10.1016/j.csi.2011.07.001

[Yalçin 10] N. Yalçin, U. Köse, What is search engine optimization: SEO, Proceeding of the 2010 World Conference on Learning, Teaching, and Educational Leadership, Elsevier, 2010.

[Lambert 09] F. Lambert, Online community information seeking: The queries of three communities in Southwestern Ontario, Journal of Information Processing and Management, Elsevier, 2009.

[Jing 06] L. Jing, et al., A Text Clustering System based on *k*-means Type Subspace Clustering and Ontology, International Journal of Electrical and Computer Engineering, 2006.

[Yates 04] R. B. Yates, C. Hurtado, M. Mendoza, Query Recommendation Using Query Logs in Search Engines, EDBT 2004 Workshops, LNCS 3268, pp. 588–596, 2004.

[Almeida 04] R. B. Almeida, V. A. F. Almeida, A Community-Aware Search Engine, Proceeding of the 2004 World Wide Web Conference, 2004.