IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 10, Issue 2 (Mar. - Apr. 2013), PP 41-45
www.iosrjournals.org

Searching and Analyzing Qualitative Data on Personal Computer

Mohit Bhansali[1], Praveen Kumar[2]
[1, 2](*Department of Computer Science & Engineering, Amity University, Noida, India)*

Abstract : *The number of files stored on our personal computer (PC) is increasing very quickly and locating information in this environment is difficult. In this paper we present a set building blocks for implementing a system which is composed of four modules i.e. indexing mechanism, analyzing text, index storing and searching mechanism. The implementation of these four modules are described in details and additionally, we provide implementation of user interface, and how they interact with each other.*
Keywords – *Desktop Search, Information Retrieval, indexing, searching, personal computer (PC). 3*

I. INTRODUCTION

With the development of computer technology, computer can complete many kinds of complicated tasks. Therefore, the number of files stored in the PC is increasing very quickly. At the same time, the number of various documents stored in the PC, such as digital photos, text files, video and audio files, increases in an amazing rate. However, a new problem arises; computer users have to spend much time searching the useful information in the ocean of the computer data, and sometimes, even ever seen or used files by users cannot be found. Therefore, the current problem which the users face is not how to save the file, but how to find and locate the file as quickly as possible. In other words, the traditional desktop information retrieval technology cannot meet the current needs of the users. This will inevitably lead to development of new technologies of desktop search engine. Desktop search engine is designed to help users find and locate the required information or documents from the PC effectively. Today, desktop search engine technologies become more popular in field of information retrieval.

The full-text search engine is the one that can search each word in documents. The full-text search engine first indexes for each document, then search system will search the index database according to the keywords which are inputted by users. The search results will return to users according to a certain sorting algorithm. The characteristic of the search engine is huge amount of information. It segments the whole content of the document and adds to the index database so that it has high recall ratio. The full-text search engine also has the characteristics of short cycle, rapid development and cheap costs.

Today there are often massive of documents stored in our PC's, we often spend much time on finding documents which are needed. The desktop search systems index the large number of documents, so that they can locate the needed documents immediately. The desktop search systems solve the problems of the difficulty of finding the right document. With the number of open-source search engine tools, we can design our personal desktop search engine conveniently and efficiently.

Lucene is an open-source full-text search engine tool which is excellent and popular. The following will introduce the characteristics and the basic frameworks of Lucene.

Lucene is a full-text search engine. You can search a lot of documents including specified words. The characteristics of Lucene is [8]: High performance of search; High scalability of target documents; Morphological analyzer; Phrase search, regular expressions, attribute search, and similarity search; Multilingualism with Unicode; Independent of the file format and repository; Intelligent web crawler; Simple and powerful API.

II. RELATED WORK

Several search and retrieval systems make extensive use of the semantically relationships that can be inferred on the desktop. Haystack [1] for example is a Java-based, open source system that aims to cater for different users' preferences and needs by giving them control and flexibility over storage and retrieval. It also emphasizes the relationship between a particular person and her corpus. It creates RDF connections between documents with similar content and then exploits this information for facilitating browsing and information access. Some basic search facilities came with Magnet [2], additional Haystack component, yet relying on database querying approaches, which are complementary to ours.

The Gnowsis system [3] is a Semantic desktop prototype which aims to integrate desktop applications such that documents are linked across applications and the data managed on desktop computers using Semantic Web technology. And users are able to use their PC as a small personal semantic web. A number of adapters read data from different sources and make this information available as RDF. Created metadata is stored in a local RDF database and can be viewed through a browser.

Other algorithms focus more on the ranking scheme that on the semantically inferable connections on the desktop. Meza et al. [4] develop a ranking technique for the possible semantic associations between the entities of interest for a specific query. They define an ontology for describing the user's interest and use this information to computer weights for the links among the semantic entities. Some desktop search tools have been proposed to retrieve all web pages and local files that were viewed in the past such as Yahoo! Desktop Search (YDS) [15] and Copernic Desktop Search (CDS) [14].

III. THE IMPLEMENTATION OF OUR SEARCH SYSTEM

In this section we will introduce our search system with the toolkits of Lucene API in NetBeans environment. The jar packages needed: lucene-core-3.0.1. JDK 1.7 version as the java runtime environment is also needed.

The function of the desktop search system is to analyze and index all the data in the local computer and also can provide full-text searching.

Lucene mainly includes two functions:

- Builds index database, and indexes the plain texts;
- According to user's query, searches index database which already is established.
-

3.1 Indexing Mechanism

Lucene is a multi-purpose information retrieval library for adding indexing and search capabilities to various applications. Its indexing process breaks down into three main operations: converting data to text, analyzing and storing it into its index structures. As most full-text search engines, Lucene implements an extended version of the vector space model, which supports fragmentation of the vector space into separate namespaces (denoted as "fields" in the actual implementation). Thus, a term does not only represent indexed words of the document's body, but also states the context in which the word appears. Finally, Lucene provides an efficient way to search for phrases which is not directly supported by the vector space model. Lucene maintains a "positional index" which contains for each term the exact position in each document. It seems easy to complete this operating process in which indexing a document may only call several methods of Lucene API. However, it actually hides ingenious and complex indexing process behind the simple process. Indexing process is shown in fig. 1.

We may see that indexing process is divided into three main stages from fig. 1, which are pretreatment, analyzing text, index storing.

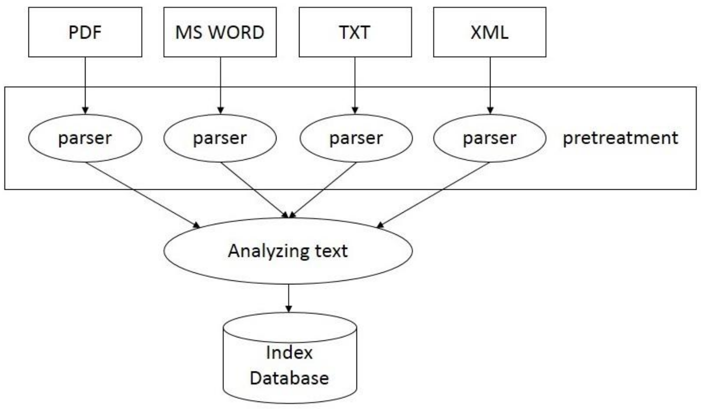


Figure 1: Lucene Indexing Mechanism

3.1.1 Pretreatment

Because Lucene can only index the plain text documents, before indexing data, the data for indexing must be converted into the text character stream which is a format that Lucene can handle. The above process is called pretreatment, namely the pretreatment is used to extract the text information from the non-text documents. After that, the data extracted is used to create Lucene's Document and corresponding Field.

3.1.2 Analyzing Text

After creating Field's Document, Lucene should not directly carry out indexing operation, but analyze data [9]. Analysis, in Lucene, is the process of converting field text into its most fundamental indexed representation, terms. These terms are used to determine what documents match a query during searches. An analyzer tokenizes text by performing any number of operations on it, which could include extracting words, discarding punctuation, removing accents from characters, lowercasing (also called as normalizing), removing common words, reducing words to a root form (stemming), or changing words into the basic form (lemmatization). This process is called tokenization, and the chunks of text pulled from a stream of text are called tokens. Tokens, combined with their associated field name, and terms [5].

3.1.3 Index Storing

Indexing is the process of extracting text from data, tokenizing it and then creating an index structure (inverted index) that can be used to quickly find which pages contain a particular word. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. After establishing terms, Lucene will call IndexWriter's addDocument (Document) method, and store data to index database with an inverted index data structure.

Example of indexing code is as follows:

```
public class SimpleIndexer {
        int index (File indexDir, File dataDir, String suffix) throws Exception {
                IndexWriter indexWriter = new IndexWriter (
                                FSDirectory.open(indexDir),
                                new SimpleAnalyzer (),
                                true,
                                indexWriter.MaxFieldLength.LIMITED);
                writer.setUseCompoundFile (false);
                indexDirectory (indexWriter, dataDir, suffix);
                indexWriter.optimize ();
                indexWriter.close();
        }
public void indexDirectory (IndexWriter indexWriter, File dataDir, String suffix) throws Exception {
        File[] files = dataDir.listFiles();
        for (int i = 0; i < files.lenght; i++) {
                File f = files[i];
                if (f.isDirectory()) {
                        indexDirectory (indexWriter, f, suffix);
                }
                else {
                        indexFileWriter (indexWriter, f, suffix);
                }
        }
}
private void indexFileWriter (IndexWriter indexWriter, File f, String suffix) throws IOException {
        Document doc = new Document ();
        doc.add(new Field("contents", new FileReader(f)));
        doc.add(new Field("filename", f.getName(), Field.Store.YES, Field.Index.ANALYZED));
        indexWriter.addDocument (doc);
}
}
```

3.2 Searching Mechanism

Once the indexing is done, the inverted files are saved. Searching involves searching through the indexed content. With regard to search engine, the deepest experience for users is search, which directly decides on user's satisfaction degree. Searching process is show in fig. 2:

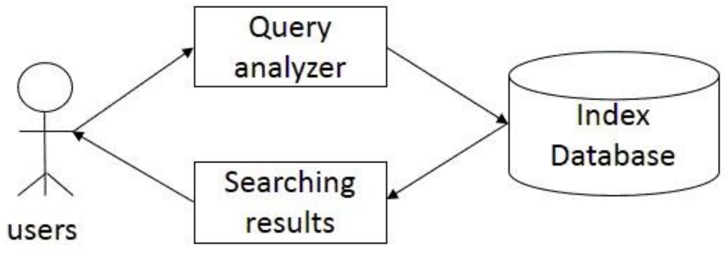


Figure 2: Lucene Searching Mechanism

Fig. 2 shows that Lucene searching process is divided into three steps: first, Lucene sends query to searcher (Lucene.search). After that, searcher calls query analyzer for parsing the query. At the moment, Lucene need to determine what language analyzer will be used; second, according to this query parsed, searcher searches index database built; three, Lucene returns searching hits (Lucene.hits) to users.

Example of searching code is as follows:

```
public class SimpleSearcher {
        File indexDir = new File ("E:/index/");
        String quertStr = "pdf";
        int maxHits = 100;
public void searchIndex () throws Exception {
        Directory directory = FSDirectory.open (indexDir);
        IndexSearcher searcher = new IndexSearcher(directory);
        MultiFieldQueryParser parser = new MultiFieldQueryParser (Version_30, new String
{"contents", "filename"}, new SimpleAnalyzer());
        Query query = parser.parse (queryStr);
        TopDocs topDocs = searcher.search(query, maxHits);
        ScoreDoc[] hits = topDocs.scoreDocs;
        for (int i = 0; i < hits.length; i++) {
                int docId = hits[i].doc;
                Document d = searcher.doc(docId);
                System.out.println(d.get("filename"));
        }
        System.out.println ("Found: " +hits.length);
}
}
```

IV. WHY LUCENE?

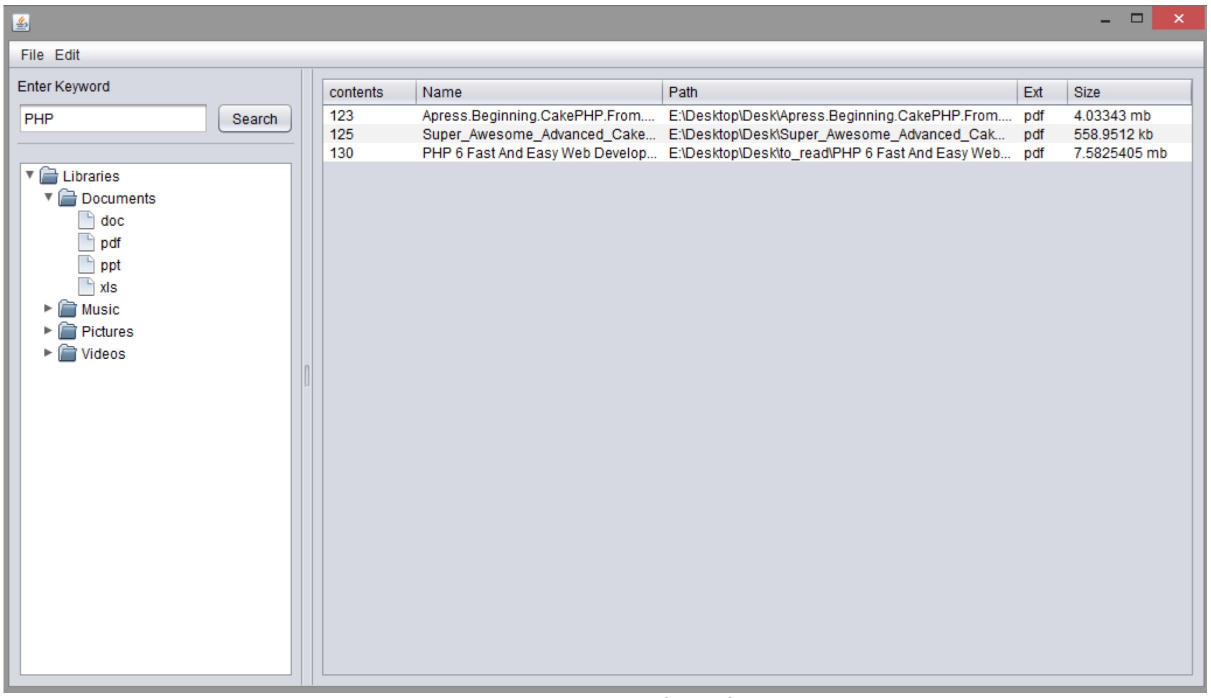
As a full-text search engine, Lucene has the following outstanding advantages [7]:

- The index file format is independent of application platform. Lucene defines a set of 8-bit byte-based index format. Making compatible system or the application of different platform to share index file created.
- Based on the traditional full-text search engine's inverted index. Lucene implements the block index. With the block index, Lucene can create the new small index file for the new files in order to improve the indexing speed, then merged with the existing index to achieve the purpose of optimization.
- The excellent object-oriented system architecture reduces the difficulty of learning the expansion of Lucene, and facilitates the expansion of new features.
- Lucene designs a text analysis interface which independent of language a document formats. The indexer completed the creation of index files by receiving the token flow. The users only need to achieve the interface of textual analysis to extend new language and file formats.
- Lucene default implements a powerful query engine. User can add powerful query capabilities to their systems even do no need to write any code. Lucene also default implements Boolean operations, fuzzy query, group query and so on.

As an excellent full-text search engine, the architecture of Lucene has strong object-oriented features [6]. Firstly, Lucene defines a platform-independent index file format. Secondly, Lucene designs abstract classes for the core of the system. The implementation of the part of the platform is designed as the implementation of abstract classes, in addition, the relevant part of the specific platforms such as the file storage is also packaged as classes, though the procedures of the object-oriented treatment, finally the search engine systems reached the goal of low coupling and high efficiency, and easy to be secondary developed. There are a lot of search systems based on Lucene such as Zilverline, Red-Piranaha, LIUS and so on.

V. USER INTERFACE OF OUR SYSTEM

The interface allows users to input specify query, displaying the result and configuration of the system. Because system works from a local index, query results can be returned very quickly, allowing a highly interactive and iterative query strategy. The user interface of the system is as follows:



contents	Name	Path	Ext	Size
123	Apress.Beginning.CakePHP.From....	E:\Desktop\Desk\Apress.Beginning.CakePHP.From....	pdf	4.03343 mb
125	Super_Awesome_Advanced_Cake...	E:\Desktop\Desk\Super_Awesome_Advanced_Cak...	pdf	558.9512 kb
130	PHP 6 Fast And Easy Web Develop...	E:\Desktop\Desk\to_read\PHP 6 Fast And Easy Web...	pdf	7.5825405 mb

Figure 3: User Interface of our System

Figure 3 shows the user interface of our system after searching the query “PHP”. There are the input box and search button on the top of user interface. On the left side, it is the display interface of different file types, including documents, music, pictures, videos etc. The right side is the search result, which shows each item in the list of search results in details, including filename, file path, extension, and file size.

VI. CONCLUSION

This paper presents the structure of desktop search engine system and its components include indexing, analyzing, index storing and searching. We have designed, deployed and evaluated a desktop full-text search system based on Lucene that provides unified access to information. After the implementation of the system, we have tested the indexing and searching capabilities of the system. Lucene is not an integrated application program of full-text information retrieval, but it is a high reliable and extensible toolkit. It can embedded into many applications provide full-text information retrieval system.

REFERENCES

[1] Karger, David R., et al. "Haystack: A customizable general-purpose information management tool for end users of semistructured data." *Proc. of the Conference of Innovative Data Systems Research*, 2005.

[2] Sinha, Vineet, and David R. Karger. "Magnet: Supporting navigation in semistructured data environments." *Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM*, 2005.

[3] Sauermann, Leo, and Sven Schwarz. "Gnowsis adapter framework: Treating structured data sources as virtual rdf graphs." *The Semantic Web–ISWC 2005(2005)*: 1016-1028.

[4] Aleman-Meza, Boanerges, et al. "Context-aware semantic association ranking." *Proceedings of Semantic Web and Database Workshop*. Vol. 3. 2003.

[5] Otis Gospodnetic,Erik Hatcher. *Lucene in Action*. Manning Publications, 2006.

[6] Li, Shengdong, et al. "Study on efficiency of full-text retrieval based on lucene."*Information Engineering and Computer Science,* 2009. *ICIECS* 2009. International Conference on. IEEE 2009.

[7] Tian, Wen, Zhou Ya, and Huang Guimin. "Research and implementation of a desktop full-text search system based on Hyper Estraier." *Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on. IEEE*, 2010.

[8] Lucene: http://lucene.apache.org/.

[9] Gospodnetic, Otis. "Parsing, indexing, and searching XML with Digester and Lucene." *Journal of IBM Developer Works* (2003).

[10] Wei Zhao, "The design and research of Literary retrieval system based on Lucene," *Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference on , vol.8, no., pp.4146,4148*, 12-14 Aug. 2011.

[11] Hristidis, Vagelis, Heasoo Hwang, and Yannis Papakonstantinou. "Authority-based keyword search in databases." *ACM Transactions on Database Systems (TODS)* 33.1 (2008): 1.

[12] Yan Hongyin; Qi Xuelei, "Design and implementation of intranet search engine system," Mechatronic Science, *Electric Engineering and Computer (MEC), 2011 International Conference on , vol., no., pp.302,304*, 19-22 Aug. 2011.

[13] Yong Zhang; Jian-lin Li, "Research and Improvement of Search Engine Based on Lucene," *Intelligent Human-Machine Systems and Cybernetics, 2009. IHMSC '09. International Conference on , vol.2, no., pp.270,273, 26-27* Aug. 2009.

[14] Copernic Desktop Search – The search engine for your PC. Available online: http://www.copernic.com/en/products/desktop-search/

[15] Yahoo! Desktop Search. Available online: http://desktop.yahoo.com/