



Analysis of Text Based Data Retrieval System

¹Saurabh S. Wani, ²Prof. Y. N. Patil

Computer Department

Dr. Babasaheb Ambedkar Technological University

Lonere, Dist. Raigad

Maharashtra, India

Abstract: This paper presents the analysis of text based data retrieval system. This paper also introduces the implementation of text retrieval system using Apache Lucene Technology. It gives the significance of Term frequency and Inverse Document Frequency in Lucene's scoring Formula. This analysis is useful for implementation of system which stores index for local files as well as data from email accounts. Lucene is a Java library which includes token scanning, token parsing, frequency count and document inverting that can be easily added to any application. In this paper, we have presented techniques to improve performance of Lucene by modifying certain parameter of document scoring formula. Lucene's performance also can be improved by modifying algorithm for incremental indexing and parallel processing. The purpose of developing such system is to reduce manual efforts of searching to greater extent. The strength of this system is its portability and security.

Keywords: Indexing, Searching, Parsing, Clustering, Lucene, Sentiment analysis

I. Introduction

With the growth in computer storage capacity, the amount of information stored on personal computers has increased significantly. With wide spread of internet, there has been tremendous increase in amount of information available on user computer. At the same time, text databases are rapidly growing due to the increasing amount of information available in electronic forms, such as electronic publications, e-mail, Social Networking website data and World Wide Web (which can also be viewed as a huge, interconnected, dynamic text database). Due to this increased data, it has become tedious work and time consuming process to search a particular file in the sea of files on hard drives for desired data. This has led to the development of Text/Document retrieval system which helps to locate files on a desktop related to query entered by user.

This system works on Lucene Technology. So, the results which are displayed to user are in sorted manner which helps user to go through results in a properly coordinated manner [2]. Lucene's emphasis is on information retrieval. Indexing and Searching steps have to be supplemented with parsing and analysis in order to achieve the best search results. The whole process of information retrieval can be divided into several sequential steps. The complete system consists of three major steps, they are [1]:

1. Indexing
2. Searching
3. Ranking

This system improves Lucene's performance by using some techniques like Parallel processing, multi-threading and Incremental indexing.

II. Related Work

To start new thing, it is necessary and essential to have knowledge about all existing tools related to our topic, their features and their working so that we can easily get new idea or identify superiority of our tool among all other. There are various tools available for document processing, indexing, token parsing, searching etc. Some of these tools with their features are listed below:-

- Lemur Toolkit – It supports construction of basic text retrieval system using language modeling, VSM, LSI, probabilistic model. It provides functionality for document processing, indexing, retrieval, summarization, document clustering, and evaluation [9].
- The Dragon Toolkit – It is a JAVA development package for Text Mining. It provides facility for text retrieval, classification, clustering, summarization and topic modeling [10].
- Xapian – It is C++ search engine library based on the probabilistic Information retrieval model. It also provides basic document processing, indexing, retrieval facilities [11].
- Lucene – It is free open source search library written in JAVA. It is developed in Stanford University by Doug Cutting. Lucene is available in various languages like CLucene in C++, Lucene.Net in .Net, Lucene4c in C and many more. It contains core functionality for document processing, indexing and searching. It supports Boolean query, phrase query, range query etc. [12].

We are focusing on analysis of document retrieval process done by Lucene. We are selecting Lucene for our project because of its amazing collection of properties like easily tunable and extensible, functionality for customizing scoring formula.

III. Architecture

Lucene is a free open source information retrieval library, originally created in Java by Doug Cutting. It is supported by Apache Software Foundation and is released under the Apache Software Licence. Lucene is having high performance, easily extensible and tuneable. It can index any file in text format. Lucene is format independent, which means it can index any file provided data stored in files can be extracted in text format [2]. Simple architecture of this system is as follows:

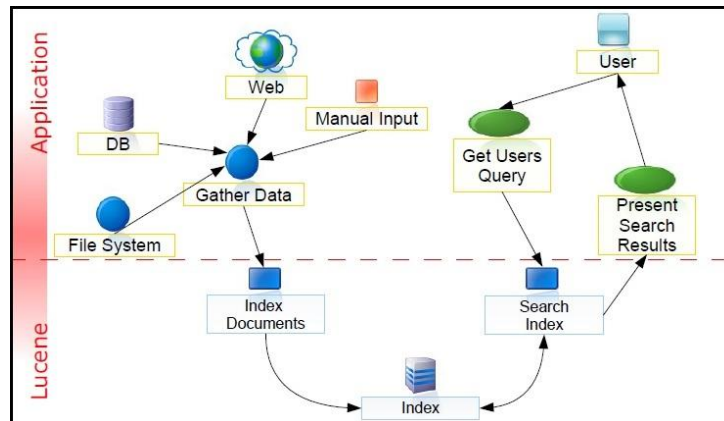


Figure 1: Simple System Architecture

System gathers data from various sources like local file system, e-mails, social networking websites etc. That data is sent to Lucene for processing. Lucene indexes that data and store index files. When user enters search query, that query is forwarded to Lucene to search into index. Lucene's primary goal is to facilitate information retrieval. This system is divided into following sequential steps.

A. Document Analysis

Lucene indexes Document objects that are represented by collection of field objects. Each field object is a name and value pair. Lucene provides following core ready to use analysers:-

- Whitespace Analyser – This split the input text into tokens on whitespace characters.
- Simple Analyser – It divides input text into tokens on non-alphabetic characters. It results in discarding all non-alphabetic characters like phone numbers, dates etc. It also performs lower casing each token.
- Stop Analyser – It divides text at no letters and lowercase and removes stop words.
- Keyword Analyser – This analyser performs no operation on the input text and treats the entire text as a single token.
- Standard Analyser – It's Lucene's most sophisticated core analyser. It removes stop words, punctuations and lowercases the token [2].

B. Indexing

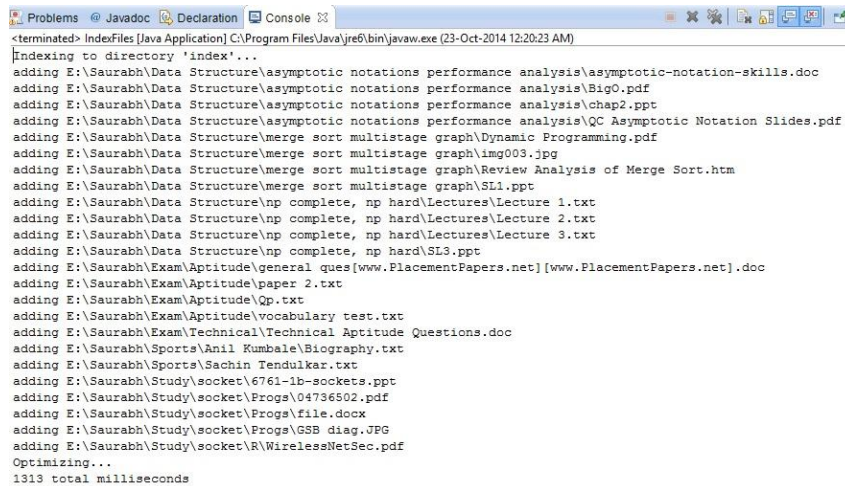
The first step of indexing is acquiring the content. Sometimes parsers need to be built to extract required content from files that contain mark-ups. For example, to parse PDF files we can use Pdftbox.

The next step is to build documents from the contents gathered in previous steps. The raw content that need to be indexed has to be translated into units (usually called documents) used by the search application. The document typically consists of several separately named fields with values, such as title, body, abstract, author etc. [2].

Lucene provides facility to boost the document for searching. Boosting can be done at indexing time or searching time. The sentences in the documents are not indexed as it is. Sentences are broken down into individual words called as token. This process is performed at document analysis time.

The process of document analysis depends on analyser used in application. Lucene provides an array of built-in analysers to perform document analysis step. It also allows customizing the analyser or for creating arbitrary analyser chains combining Lucene tokenizers and token filters.

The final step is to index the document. During this step, the document is added to the index. Following image shows how Lucene adds document to index and time required for indexing the document [2].



```

<terminated> IndexFiles [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (23-Oct-2014 12:20:23 AM)
Indexing to directory 'index'...
adding E:\Saurabh\Data Structure\asymptotic notations performance analysis\asymptotic-notation-skills.doc
adding E:\Saurabh\Data Structure\asymptotic notations performance analysis\BigO.pdf
adding E:\Saurabh\Data Structure\asymptotic notations performance analysis\chap2.ppt
adding E:\Saurabh\Data Structure\asymptotic notations performance analysis\QC Asymptotic Notation Slides.pdf
adding E:\Saurabh\Data Structure\merge sort multistage graph\Dynamic Programming.pdf
adding E:\Saurabh\Data Structure\merge sort multistage graph\img003.jpg
adding E:\Saurabh\Data Structure\merge sort multistage graph\Review Analysis of Merge Sort.htm
adding E:\Saurabh\Data Structure\merge sort multistage graph\SL1.ppt
adding E:\Saurabh\Data Structure\np complete, np hard\Lectures\Lecture 1.txt
adding E:\Saurabh\Data Structure\np complete, np hard\Lectures\Lecture 2.txt
adding E:\Saurabh\Data Structure\np complete, np hard\Lectures\Lecture 3.txt
adding E:\Saurabh\Exam\Aptitude\general ques[www.PlacementPapers.net][www.PlacementPapers.net].doc
adding E:\Saurabh\Exam\Aptitude\paper 2.txt
adding E:\Saurabh\Exam\Aptitude\Qp.txt
adding E:\Saurabh\Exam\Aptitude\vocabulary test.txt
adding E:\Saurabh\Exam\Technical\Technical Aptitude Questions.doc
adding E:\Saurabh\Sports\Anil Kumbale\Biography.txt
adding E:\Saurabh\Sports\Sachin Tendulkar.txt
adding E:\Saurabh\Study\socket\6761-1b-sockets.ppt
adding E:\Saurabh\Study\socket\Progs\04736502.pdf
adding E:\Saurabh\Study\socket\Progs\file.docx
adding E:\Saurabh\Study\socket\Progs\GSB diag.JPG
adding E:\Saurabh\Study\socket\R\WirelessNetSec.pdf
Optimizing...
1313 total milliseconds

```

Figure 2: Lucene indexing

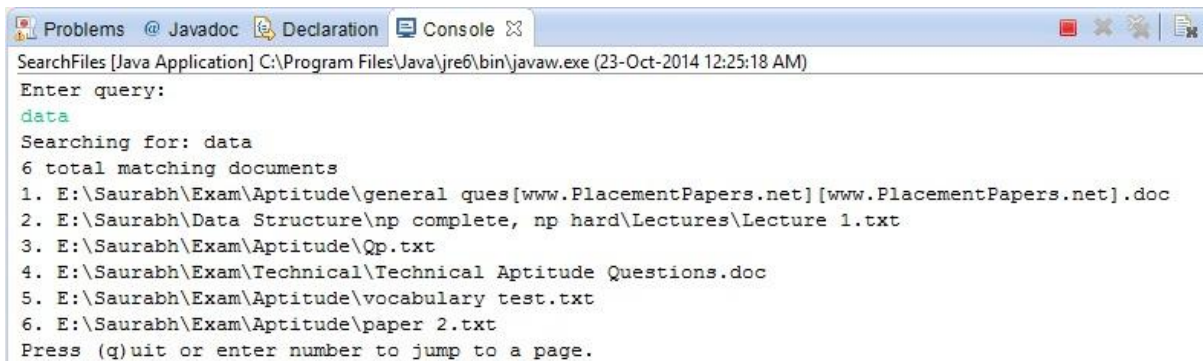
C. Searching

Once index is created for all documents in system, Searching operation can be performed efficiently and It is the process of looking up words in an index to find documents where they appear.

To perform searching, we need to know search query. To obtain search query, user is presented with GUI, where he or she provides search request. Lucene provides facility to boost the query or terms in query. This request is then translated into an appropriate query object for the search engine. Lucene provides a query parser package to process the request into a query object according to common search syntax.

After creating query object from user search request, application performs searching to retrieve documents matching the query, sorted in the requested sort order [2].

Once a set of matched documents that match the query are retrieved by searching into index, it is displayed to user as per relevance ranking, sorted in the right order.



```

SearchFiles [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (23-Oct-2014 12:25:18 AM)
Enter query:
data
Searching for: data
6 total matching documents
1. E:\Saurabh\Exam\Aptitude\general ques[www.PlacementPapers.net][www.PlacementPapers.net].doc
2. E:\Saurabh\Data Structure\np complete, np hard\Lectures\Lecture 1.txt
3. E:\Saurabh\Exam\Aptitude\Qp.txt
4. E:\Saurabh\Exam\Technical\Technical Aptitude Questions.doc
5. E:\Saurabh\Exam\Aptitude\vocabulary test.txt
6. E:\Saurabh\Exam\Aptitude\paper 2.txt
Press (q)uit or enter number to jump to a page.

```

Figure 3: Lucene Searching

D. Lucene Extensions

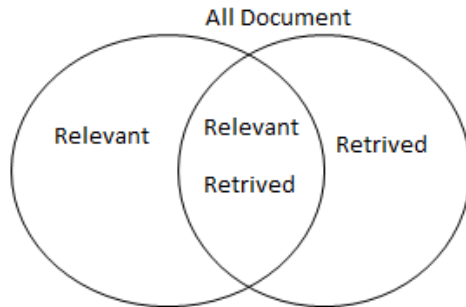
Lucene is easily extensible and tunable. Beyond Lucene's core functionality, there are a number of extension modules available that can be integrated to Lucene. These include the spellchecker and highlighter modules among others. The example of Lucene extensions includes wordnet, Tika, Pdfbox etc.

IV. Mathematical Analysis

Searching is the process of looking up words in an index to find documents where they appear [1]. The Search efficiency of any search application can be defined using precision and recall metrics. Precision is the fraction of retrieved documents that are relevant to the search. Recall in information retrieval is the fraction of the documents that are relevant to the query that are successfully retrieved [2].

Recall measures how many of the relevant documents in a collection have actually been found. For example, a 80% recall rate means that 80% of all relevant documents in a collection have been found, and 20% have been missed. Recall measures how well the search system finds relevant documents.

Precision measures how many of the documents retrieved are actually relevant, that is, how much of the result set is on target. For example, a 75% precision rate means that 75% of the documents retrieved are relevant, while 25% of those documents have been misidentified as relevant. Precision measures how well the system filters out the irrelevant documents [2].



$$\text{Precision} = \text{Relevant Retrieved} / \text{Total retrieved}$$

$$\text{Recall} = \text{Relevant retrieved} / \text{Total Relevant}$$

Figure 4: Relevancy

A perfect retrieval system would retrieve only the relevant documents and no irrelevant documents. Every time a document matches during a search, Lucene calculates a score (a numeric value of relevance) for document. The score reflects level of relevance of document to query. Higher scores reflect stronger relevance to user search query.

The score is computed for each document (d) matching each term (t) in a query (q) as follows [1]:

$$\text{score}(q, d) = (\text{coord}(q, d) \cdot \text{queryNorm}(q) \cdot \sum (tf(t \in d) \cdot idf(t)^2 \cdot \text{boost}(t, \text{field} \in d) \cdot \text{lengthNorm}(t, \text{field} \in d)))$$

Sr. No.	Functions	Explanations
1	tf (t ∈ d)	Normalized number of times the term appears in a document (term frequency)
2	idf (t)	Total number of documents/number of documents containing the term (inverse document frequency)
3	coord (q,d)	How many of the query terms are found in the document (coordination factor)
4	queryNorm(q)	Normalizing factor assuring comparable scores between queries (normalization value for query)
5	boost(t.field ∈ d)	Index time boost, specified during indexing
6	lengthNorm(t.field ∈ d)	Normalizing factor assigning shorter fields bigger boost (normalization value of a field)

Table 1: List of functions used in formula [2]

The computed score is a raw score, which is a non-negative floating-point number. It is a good manner to normalize the scores by dividing all scores by the maximum score for the query.

A. Significance of TF and IDF

To understand significance of TF-IDF, let us consider search query as "Relevant Retrieved" and we have four document with TF for Relevant Retrieved as follows:

	Term Frequency (TF)		
	Relevant	Retrieved	Total
First 5 document retrieved	2	24	26
Results for relevant retrieved	8	5	13
Precision for retrieved document	4	15	19
Relevant retrieved definition	9	8	17

Table 2: Term Frequency table

We take number of occurrences of term in those documents, sum them and order them. With only TF consideration, we get following ranking result for above query

	Term Frequency (TF)		
	Relevant	Retrieved	Total
First 5 document retrieved	2	24	26
Precision for retrieved document	4	15	19
Relevant retrieved definition	9	8	17
Results for relevant retrieved	8	5	13

Table 3: Sorted Term Frequency table

For the given query, last and second last document must get higher ranking over other two documents. But these results are not sorted as per expectation. So, consider following IDF (Inverse Document Frequency):

TERM	IDF
Relevant	11
Retrieved	3

Table 4: IDF table

By considering above TF and IDF, we get ranking for each document as follows:

	TF * IDF		
	Relevant	Retrieved	Total
Relevant retrieved definition	9 * 11	8 * 3	123
Results for relevant retrieved	8 * 11	5 * 3	103
First 5 document retrieved	2 * 11	24 * 3	94
Precision for retrieved document	4 * 11	15 * 3	89

Table 5: Sorted TF * IDF table

For this reason, Lucene scoring formula considers TF-IDF.

B. Example

Suppose we have two documents D1, D2 and query Q as follows:

D1: please retrieve document containing word document.

D2: retrieve file

Q: let's retrieve document

Then score for above two documents can be calculated as follows:

$$\text{coord}(Q, D) = \frac{\text{overlap between } Q \text{ and } D}{\text{maximum overlap}}$$

$$\text{coord}(Q, D1) = 2/3, \text{coord}(Q, D2) = 1/3$$

$$\text{queryNorm}(Q) = \frac{1}{\sqrt{\text{sum of square weight}}}$$

$$\text{sum of square weight} = (q.\text{getBoost}())^2 \cdot \sum_{t \text{ in } Q} (idf(t) \cdot t.\text{getBoost}())^2$$

$$t.\text{getBoost}() = 1, q.\text{getBoost}() = 1$$

$$\text{sum of square weight} = \sum_{t \text{ in } Q} (idf(t))^2$$

$$\text{queryNorm}(Q) = 1/(0.5945^2 + 1^2)^{1/2} = 0.8596$$

$$tf(t \text{ in } d) = \sqrt{\text{frequency}}$$

$$tf(\text{let's}, D1) = 0, tf(\text{retrieve}, D1) = 1, tf(\text{document}, D1) = 2^{1/2} = 1.4142$$

$$tf(\text{let's}, D2) = 0, tf(\text{retrieve}, D2) = 1, tf(\text{document}, D2) = 0$$

$$idf(t \text{ in } d) = \ln(N/(n_t + 1)) + 1$$

$$idf(\text{let's}) = 0, idf(\text{retrieve}) = \ln(2 / (2+1)) + 1 = 0.5945, idf(\text{document}) = \ln(2 / (1+1)) + 1 = 1$$

$$\text{norm}(D) = \frac{1}{\sqrt{\text{number of terms}}}$$

$$\text{norm}(D1) = 1/6^{1/2} = 0.4082,$$

$$\text{norm}(D2) = 1/2^{1/2} = 0.7071$$

$$\text{Score}(Q, D1) = 2/3 * 0.8596 * (1 * 0.5945^2 + 1.4142 * 1^2) * 0.4082 = 0.4135$$

$$\text{Score}(Q, D2) = 1/3 * 0.8596 * (1 * 0.5945^2) * 0.7071 = 0.0716$$

Thus, both document D1 and D2 are relevant to query but D1 is having first rank as it is having higher document score (more relevant) than document D2.

V. Conclusion

With the help of Lucene, we can develop text based data retrieval system which is very helpful for faster and efficient information retrieval for user. We are developing same for the text retrieval. With this, user can fetch data from Social networking sites, email accounts, local file system etc., retrieve text from it and index that data. User can search for any post, document, email from that indexed data with this system. Search results are sorted according to relevancy. Scoring formula of Lucene can be customized to change result order.

VI. References

- [1] Lucene in Action, Second Edition by Erik Hatcher, Michel McCandless
- [2] Lucie Molková, Indexing very large text data, Master's thesis, in Brno, Spring 2011.
- [3] Chang Tien Lu, Manu Shukla, Siri H. subramanya, Yamin wu, Performance evaluation of desktopsearch engine, Virginia Polytechnic Institute and State University, USA, IEEE 2007.
- [4] Performance Analysis and Optimization on Lucene – David Chi-Chuan Su.
- [5] Andrzej Bialecki, Robert Muir, Grant Ingersoll, Apache Lucene 4, Lucid Imagination.
- [6] David Chi-Chuan Su, Performance Analysis and Optimization on Lucene, 609 Escondido Road, Stanford, CA 94305
- [7] Eric Schreiner, High Speed Searching with Lucene presented at Colorado Software Summit, october 2008
- [8] <http://wiki.apache.org/lucen=java/Improve-IndexingSpeed>
- [9] <http://www.lemurproject.org/>
- [10] <http://dragon.ischool.drexel.edu/>
- [11] <http://xapian.org/>
- [12] <http://lucene.apache.org/core/>