Introduction of Apache Lucene

Qin Geng qingeng2@illinois.edu

ABSTRACT

Apache Lucene is high-performance, full-featured text search engine library written entirely in Java. It is suitable for nearly any application that requires full-text search, especially cross-platform. This paper focus on Lucene's main features, including text analysis, indexing, and query. With knowing these features of Apache Lucene, along with a big picture of its architecture, scientist and developers can execute better and more precise full-text search from both academy and developments.

Keywords

Information Retrieval, Open Source, Apache Lucene, Full-text retrieval; Search engine

1. INTRODUCTION

Apache Lucene is an open-source search software which releases a java-based core library, named Lucene^{TM}, and a python binding, named PyLucene [1]. Apache Lucene is supported and maintained by Apache Software Foundation (ASF) and is licensed under the Apache Software License v2 [2]. Nowadays, Lucene is widely used and powering search on many popular websites, applications and devices, such as Twitter, Netflix, Instagram, etc [3]. It can also be used from various programming languages including Object Pascal, Perl, C#, C++, Python, Ruby and PHP [4].

Apache LuceneTM's indexing and searching capabilities also make it attractive for any number of uses—development or academic. In this paper, we will focus on Lecene's main features include text analysis, indexing, and query. This paper will also introduce the big picture of Lucene's architecture, as well as its specific modules like segment, text analysis chains, and indexing chain.

2. BACKGROUND

Lucene was originally written by Doug Cutting in 1999 as a means to learnings Java[5]. It was initially available for download at the SourceForge website, but was donated subsequently to the Apache Software Foundation (ASF) in 2001 [6]. In ASF, Lucene joined the Jakarta family of open-source Java and became its own top-level Apache project in February 2005.

From the earliest days, Lucene implemented a modified vector space model that supports incremental modifications to the index and supported a variety of query types. As it's been updated to higher and higher levels, more query types have been added, including support for regular expressions, complex phrases, spatial distances, etc.

3. LUCENE^{TM} FEATURES

Lucene offers powerful features like scalable and high-performance indexing of the documents and search capability through a simple API. It utilizes powerful, accurate and efficient search algorithms written in Java. Most importantly, it is a cross-platform solution. Therefore, it's popular in both academic and commercial settings due to its performance, reconfigurability, and generous licensing terms. Lucene consists of a number of features which can be concluded into three major categories: text analysis, indexing, and queries.

3.1 Text Analysis

The text analysis in Lucene focuses on taking in content in the form of documents to be indexed or queries to be searched and converting them into an appropriate internal representation that can then be used as needed. We can see the text analysis process for both input content and queries in Figure 1, Lucene's Architecture. The analysis process consists of three tasks which are chained together to operation on incoming content: 1) optional character filtering and normalization (e.g. removing diacritics), 2) tokenization, and 3) token filtering (e.g. stemming, lemmatization, stopword removal, n-gram creation).

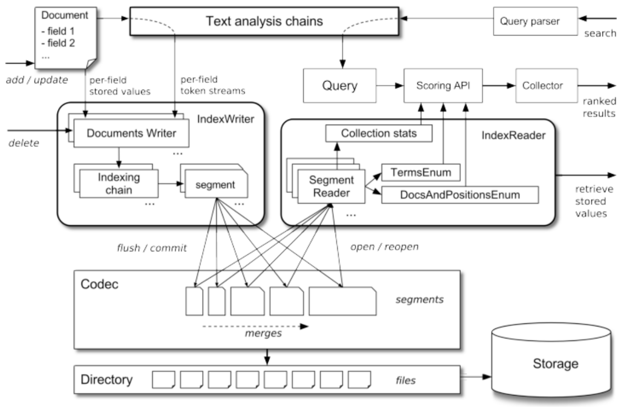

Figure 1 Lucene's Architecture

As shown in the Figure 2, the text analysis takes place in the text analysis chain which produces a stream of tokens from the input data in a field. Tokens in the analysis chain are represented as a collection of "attributes". In addition to the expected main "term" attribute that contains the token value there can be many other attributes associated with a token, such as token position, starting and ending offsets, token type, arbitrary payload data (a byte array to be stored in the index at the current position)

There are multiple built-in analyzers. For example, StandardAnalyzer (analyses based on basic grammar, removes stop words like "a", "an" etc. Also converts in lowercase), SimpleAnalyzer (breaks the text based on no-letter character and converts in lowercase), WhiteSpaceAnalyzer (breaks the text based on white spaces), and more other analyzers for using and customizing.

3.2 Indexing

Lucene uses an "inverted indexing" of data – instead of mapping pages to keywords, it maps keywords to pages just like a glossary at the end of any book. This allows for faster search responses, as it searches through an index, instead of searching through text directly [7]. At indexing time, analysis creates tokens that are ultimately inserted into Lucene's inverted index, while at query time, tokens are created to help form appropriate query representations.

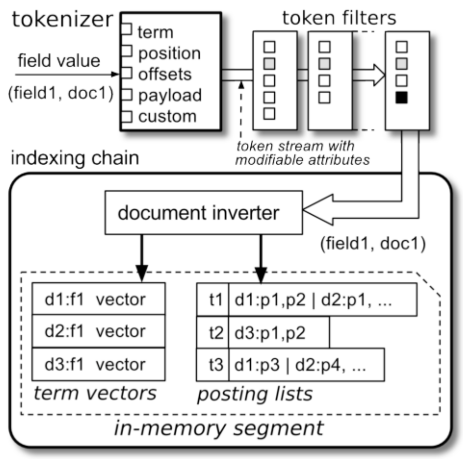

Figure 2 Indexing process

3.2.1 Document

Documents are modeled in Lucene as a flat ordered list of fields with content. Fields have name, content data, float weight (used later for scoring), and other attributes, depending on their type, which together determine how the content is processed and represented in the index. There can be multiple fields with the same name in a document, in which case they will be processed sequentially. Documents are not required to have a unique identifier (though they often carry a field with this role for application-level unique key lookup) – in the process of indexing documents are assigned internal integer identifiers.

3.2.2 Field

There are two broad categories of fields in Lucene documents – those that carry content to be inverted (indexed fields) and those with content to be stored as-is (stored fields). Fields may belong to either or both categories (e.g. with content both to be stored and inverted). Bothe indexed and stored fields can be submitted for storing / indexing, but only stored fields can be retrieved – the inverted data can be accessed and traversed using a specialized API.

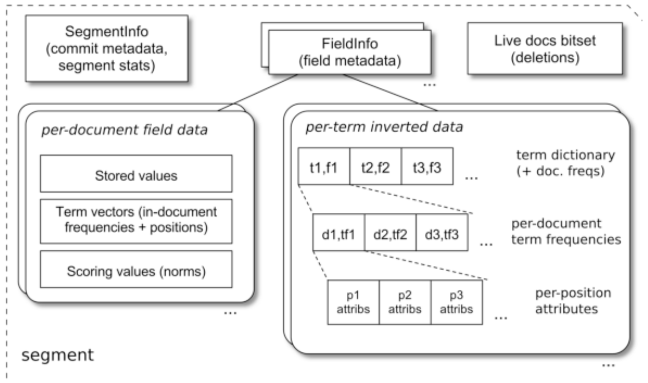
SegmentInfo (commit metadata, segment stats)
FieldInfo (field metadata)
Live docs bitset (deletions)
per-document field data
Stored values
Term vectors (in-document frequencies + positions)
Scoring values (norms)
per-term inverted data
t1,f1
t2,f2
t3,f3
term dictionary (+ doc. freqs)
d1,tf1
d2,tf2
d3,tf3
per-document term frequencies
p1 attribs
p2 attribs
p3 attribs
per-position attributes
segment

Figure 3 Structure of a Lucene segment

3.2.3 Indexing

The resulting token stream is finally processed by the indexing chain and the supported attributes (term value, position, offsets and payload data) are added to the respective posting lists for each term (Figure 3). Also, at this stage documents are assigned their internal document identifiers, which are small sequential integers (for efficient delta compression). These identifiers are ephemeral – they are used for identifying document data within a particular segment, so they naturally change after two or more segments are merged.

3.2.4 Index Updating

When new documents are submitted for indexing, their fields undergo the process described in the Indexing section, and the resulting inverted and non-inverted data is accumulated in new in-memory index extents called "segments" (Figure 3), using a compact in-memory representation. Periodically these in-memory segments are flushed to a persistent storage, whenever they reach a configurable threshold – for example, the total number of documents, or the size in bytes of the segment.

3.3 Queries

On the search side, Lucene supports a variety of query oprions, along with the ability to filter, page and sort results as well as perform pseduo relevance feedback. For querying, Lucene provides over 50 different kinds of query representations, as well as several query parsers and a query parsing framework to assist developers in writing their own query parse.

3.3.1 Query Types

Lucene uses Query objects to perform searches instead of enforcing a particular query language. Complex queries can be expressed by several queries provided as building blocks. Developers can also construct their own queries programmatically or via a Query Parser. Lucene provides different type of concrete queries, like TermQuery, PrefixQuery, WildcardQuery, PhraseQuery, FuzzyQuery, and BooleanQuery [8].

Term is a basic unit for searching, containing the field name together with the text to be searched for, and TermQuery is the simplest of all queries consisting of a single term. PrefixQuery, we can tell from the name, is searching a document with a "Starts with" word. We can use wildcards "*" or "?" for searching in WildcardQuery, and PhraseQuery is used to serach a sequence of texts in a document. When we want to search for something similar, but not necessarily identical, we can use FuzzyQuery. BooleanQuery is used to execute complex searches, like combining two or more different types of queries.

3.3.2 Query Evaluation

The inverted segments are processed sequentially for efficiency when a Query is executed. For each index segment, the Query generates a Scorer which is essentially an enumerator over the matching documents with an additional score() method. Scorers will compute the score for each query, typically by passing raw index statistics (term frequency for example) to the Similarity.

4. CONCLUSIONS

This paper presented a big view of Lecune's architecture, as well as details on its specific features like text analysis, indexing, and query. These features make Lucene one of the top level search-based applications in industry today, and the ASF open community are collectively working to better it. By overhauling the underpinnings of Lucene to be more flexible and pluggable, along with greatly improving the efficiency and performance, Lucene is well suited for continued commercial success as well as better positioned for experimental research work.

5. REFERENCES

[1] Apache Lucene – Welcome to Apache Lucene, Accessed 11/02/2021, http://www.apache.org.

[2] Apache License, Version 2.0, Accessed 11/02/2021, http://www.apache.org/licenses/LICENSE-2.0

[3] A. Bialecki, R. Muir, G. Ingersoll, "*Apache Lucene 4*", Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval.

[4] LuceneImplementations,

Accessed 11/02/2021, https://cwiki.apache.org/confluence/display/lucene/LuceneImplementations

[5] Better Search with Apache Lucene and Solr,

Accessed 11/02/2021, https://www.yumpu.com/en/document/view/51026903

[6] Apache Subversion Initial Lucene Revision, The Apache Software Foundation, Accessed 11/02/2021. http://svn.apache.org/viewvc?view=revision&revision=149570

[7] Q. Wang, W. Wu and Y. Gu, "The Application of Lucene in Information Leakage Monitoring and Querying System," 2010 2nd International Conference on Information Engineering and Computer Science, 2010, pp. 1-4.

[8] Introduction to Apache Lucene. Accessed 11/01/2021. https://www.baeldung.com/lucene