CS410 Technology Review: Apache Lucene

Evan Luo

Introduction:

Apache Lucene is a free, open-source information retrieval library originally written in Java. Created by Doug Cutting in 1999, it is currently supported by the Apache Software Foundation. It is also available in various programming languages, such as C++, Python, Perl, Ruby. Its powerful features and cross platform compatibility allow it to be suitable for any application that requires full-text search. This review will cover the main features, capabilities and extensions, and pros and cons of Apache Lucene.

Details and Features:

Lucene performs very fast full-text searching by utilizing *indexing*, and it uses a method called *inverted indexing*. Instead of having a classic index where for every document you have the full list of terms it contains, *inverted indexes* do it the other way around. For every term in the documents, you have a list of all the documents that contain that term, which makes it hugely more convenient when performing full-text searching.

The indexing process in Lucene is split into multiple steps:

1. Feed with text documents or sources.
2. For every document it analyzes the text and splits it into terms. Meanwhile it can perform all kinds of analysis in the plain text.
3. For every term of the documents, it creates the inverted lists.
4. The index is ready to be searched.

The basic components used for indexing are Directories, Documents, Fields, Terms, and Analyzer. A Directory is where Lucene uses in your file system to store everything that is necessary for the index. A Document is an object that represents the document you want to index. Documents are indexing components, and not the actual text sources. Because it represents an individual physical text source, it is the building unit of the Index. The Document objects are populated with a collections of Field items. A Field is a pair of (name, value) items. So, when creating a new Document object, it must be filled with that kind of pairs. A Term represents a word from the text. Terms are extracted from the analysis and tokenization of the Fields' values; thus, Term is the unit of searching. Term is composed of two elements, the text word and the name of the Field this word appeared into. An Analyzer is responsible for taking plain text and converting it to searchable Terms. It can perform various kinds of text and word analysis like stemming, stop words filtering, text normalization, and synonym expansion.

For searching, the basic components are QueryBuilder, Query, IndexReader, IndexSearcher, TopDocs, and ScoreDoc. A Query object is a query passed to the Index. A QueryBuilder object is used to create the Query object. IndexReader opens the Index and accesses it before doing a search. The IndexSearcher is a class used to search a single Index. The TopDocs class represents the hits that satisfy the query. The ScoreDoc represents a hit for a query. It consists of the ID of the Document that satisfied the query, and the score that the Document achieved in the query. The scoring formula here is used to provide a relevance measure for the retrieved Document. This helps to characterize the good and bad Documents and ensures that the good ones are close as possible.

Although Lucene itself is only an indexing and search library, numerous applications and projects have been created to extend its capabilities, such as the following examples:

- Apache Nutch: An open-source project to provide functions such as parsing, data retrieval, querying and clustering.
- Apache Solr: An open-source search engine. Also can be used as a document-based NoSQL database with transactional support that can be used for storage purposes.
- CrateDB: An open-source distributed SQL database management system that integrates a fully searchable document-oriented data store.
- Elasticsearch: An open-source search engine. It acts as a server that can process JSON requests and give back JSON data.
- OpenSearch: A distributed, open-source search and analytics suite used for real-time application monitoring, log analytics, and website search.

Lucene provides rapid indexing; with proper optimization it could index a large amount of data in a short amount of time. With its extensibility, it could be easily integrated into web crawlers. It also provides numerous sorting options for results based on the search field and relevance, making it extremely flexible. With its ability to provide quick and accurate results, it has no issues in being performant.

Although being a powerful and efficient tool, Lucene still has a few downsides. For example, it is difficult to port a project to a cluster-based environment on the cloud. The learning curve for the tool is rather steep, with limited centralized documentation. Scalability issues also occur; the performance degrades when the index gets larger, though this is not a Lucene-specific issue (happens on any indexing system).

Overall, the disadvantages are only minor issues, and does not affect Lucene of being a strong candidate for developing search applications.

Conclusion:

By providing an API to a search engine library, Apache Lucene demonstrates impressive extensibility and powerful mechanics. It is scalable, providing high-performance indexing. It is also efficient and accurate. Although written in Java, implementation in other programming languages is also available. But the best part is that it is available as an open-source software, meaning that it is free to use. As a search engine, Lucene makes a perfect fit within a software ecosystem with needs for massive scalability.

References:

1. https://www.lucenetutorial.com/basic-concepts.html
2. https://en.wikipedia.org/wiki/Apache_Lucene
3. https://lucene.apache.org/
4. http://www.baeldung.com/lucene