SEARCHING AND INDEXING WITH APACHE LUCENE

TECHNOLOGY REVIEW

TABLE OF CONTENTS

INTRODUCTION

Apache Lucene is an open-source search engine library created in Java as a core search library, named Lucene™ core, as well as PyLucene, a python binding for Lucene which is available free under the Apache License. It is associated to Apache Solr, and includes a number of sub-projects, such as Lucene.NET, Apache Tika, and Apache Nutch, now all top-level Apache projects.

This can be used in various use-cases which needs full-text search, especially in a cross-platform environment.

DETAILED ANALYSIS

Lucene is a Scalable, High-Performance Indexing solution which has less RAM requirement and fast incremental indexing. It is performant in querying large data and fetching the top matching files based on the metadata provided. It has good community support to find online help for the product as this has been used in various use cases and has many publicly available success stories.

Lucene has powerful and vast set of features which makes it a good choice for search applications both for simple and advanced applications.

Features supporting Common use cases

- fast, memory-efficient, and typo-tolerant suggesters
- sorting by any field
- allows simultaneous update and searching

Advanced features

- Ranked searching -- best results returned first
- Query types - phrase, wildcards, range, and proximity queries
- Fielded searching
- Multi index searching with merged results
- Flexible faceting, highlighting, joins and result grouping
- Pluggable ranking models

INDEXING

Lucene uses inverted index i.e., mapping of an entity to its metadata. The metadata holds the information about the matches of the term in all the files and the word counts of the term.

Lucene capable of managing dynamic collection of documents and supporting rapid updates to the index while adding/removing documents from the collection.

Document indexing involves constructing a document that contains the fields to be indexed, then adding that document to the index.

A Field stores the terms we want to index and search on. It stores a mapping of the name of the field and the corresponding value

ANALYZE

In this process, the text data is filtered and cleaned. The preprocessing steps involves steps like extracting words, removing stop words, converting to common letter case and tokenization. The indexing process results in an inverted index being stored for the search functionality.

SEARCHING

After indexing, the queries are processed through the IndexSearcher and parse the query, create a QueryParser and search the index for results. The results are returned as TopDocs which contain ScoreDocs, which contain the document IDs and the confidence scores of the results that match the query.

- IndexSearcher : Provides access to the index and several search methods that takes the input query and ouputs the TopDoc.
- Term : Basic searching unit and used in a TermQuery when searching
- Query: Lucene provides several types of Queries like Prefix, Wildcard, Phrase, and Fuzzy.
- QueryParser: Parses a human-readable query into system understandable Query object .
- TopDocs : Container for pointers to N search results. Each TopDoc contains a document ID and a confidence score.

The Lucene search API takes a search query and returns a set of documents ranked by relevancy. All searches are field specific because Lucene indexes terms and a term is composed of a field name and a token.

CONCLUSION

Apache Lucene is powerful tool for fetching the right information based on the query and metadata as it provides below rich feature set.

- Indexing - Organize all content for a quick access
- Query Parsing - Understand what user is looking for
- Search and browse - Quickly find the information of interest
- Relevance Ranking - Order the information in useful ways

Apache Lucene is a perfect text search implementation under below scenarios.

- Minimal heap space usage
- Search based on various search fields
- Simultaneous search and index process

When the index size grows large there might some scalability issues where the search might take longer but this is dependent on the data size, data variety, and the latency requirements of the use case.