Technology Review: Apache Lucene

Author – Shaleen Mehrotra, shaleen3@illinois.edu

Introduction

Apache Lucene is an open-source Java library which is used as a data retrieval engine. It has powerful data retrieval capabilities such as whole-text indexing. It also provides many other types of indexing which satisfies various requirements of different types of search queries.
A lot of open-source projects use Lucene as their base. For e.g., Elasticsearch, Neo4j, etc. Elasticsearch and Neo4j, both are based on Lucene search engine. They leverage all the capabilities of Lucene and provide a convenient abstract layer over it for the users to interact and use the search engine easily. It is safe to say that if Elasticsearch and Neo4j are cars then Lucene is their engine.
This review will discuss about Lucene and how it is used in Elasticsearch and Neo4j.

Body

The 4 most powerful advantages of Lucene are –

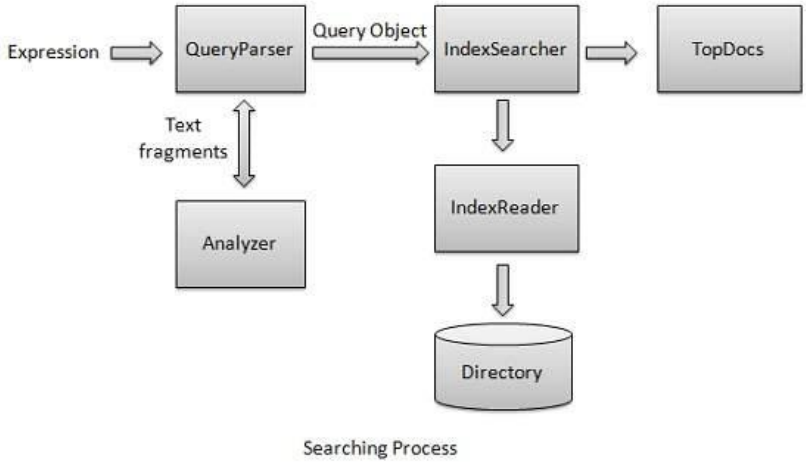
- Speed and high-performance indexing.
- Accurate and efficient search algorithms
- Open source and cross-platform
- Scalability

Indexing – When we have a huge number of documents and we need to find one specific document which contains a certain word that we are looking for, we look for an option that can do this search task quickly. Here is where indexing comes into the picture. To be able to search large amounts of text quickly, firstly we need to index that text and convert it into a format which will help us do the search rapidly. Lucene uses an inverted index data structure where each term is mapped to its metadata. This metadata contains information about the files that contain the term, the number of occurrences of the term, etc.
Unlike in relational database there is no primary key, row or column constraints in Lucene index. Indexing in Lucene is done using documents and fields. A document is a container that can contain one or more fields and a field stores the terms which we want to index and do our search on. There can be multiple documents in an index and each document is assigned a unique id, usually called as DocId. The index is stored as a part of a term dictionary. A term is the smallest unit of index in Lucene and a term dictionary is the basic index which is used to perform searches on the terms.

Searching – Once all the documents are indexed, the search functionality needs to be added. Each search query is parsed and the index is searched for the results. The returned results are in the form of top documents which contain the ids of all those documents and their confidence score. These documents are the results that matched the parsed query. Searching involves retrieving documents from an index using an IndexSearcher.

- **IndexSearcher** – It provides read-only access to the index. It exposes various methods that take the query and return the top and best documents (TopDocs) as the result. There is also an IndexWriter class which is used to create or update indexes.

- **Query** – Several types of queries are provided by Lucene, such as FuzzyQuery, BooleanQuery, WildcardQuery, PrefixQuery, TermQuery, and PhraseQuery. Every query has its own unique way of searching the index.
- **QueryParser** – It parses the query that is entered by a human (for ex: “arlington”) into a parsed query object. The query object is used for searching.
- **TopDocs** – It is a container for all the pointers which point to the top N search results. Each TopDoc contains a document ID and a document confidence score.



Searching Process

Image Source: Tutorials Point

Lucene in Elasticsearch

Elasticsearch is built on top of Lucene. Elasticsearch provides a JSON based REST API to use all the features of Lucene. It converts Lucene into a distributed search engine for scaling it up horizontally. Lucene is not built for distributed system, but it is Elasticsearch that provides this abstraction of distributed structure. Elasticsearch also provides queues, cluster management, cluster health API, data monitoring API, etc. In short, Elasticsearch extends Lucene and provides additional features beyond it.

Suppose we want to search for a specific term (For e.g., "Cake" or "Cookie"), then we will have to go over each shard in Elasticsearch and look for that term. This operation will take a lot of time. So, we will have to use an efficient data structure for this search. This is where Lucene's index comes into the picture.

- **Schema-Free**: Schema fields (name, value pairs) are not needed to be defined before. Elasticsearch creates a schema automatically at runtime as soon as the data is indexed.
- **Powerful Query Domain Specific Language**: JSON interface is used by Elasticsearch to read and write the queries on top of Lucene. Complex queries can be easily written even without knowing the Lucene syntax.
- **High Availability**: High distribution of Elasticsearch makes it easy to manage the data replication. High distribution means to have multiple copies of data in our Elasticsearch cluster, thus enabling high availability.

When an index is created in Elasticsearch, it is divided into one or more primary shards for scaling the data and splitting it into multiple nodes. Each shard is a separate instance of Lucene. Elasticsearch not only allows us to search but also allows us to store and analyze large volumes of data rapidly in almost real-time and gives back the results in milliseconds. It can achieve such fast search responses because it searches the index instead of searching the text directly.

The below table represents how documents are stored in an inverted Index in Elasticsearch which has Lucene underneath.

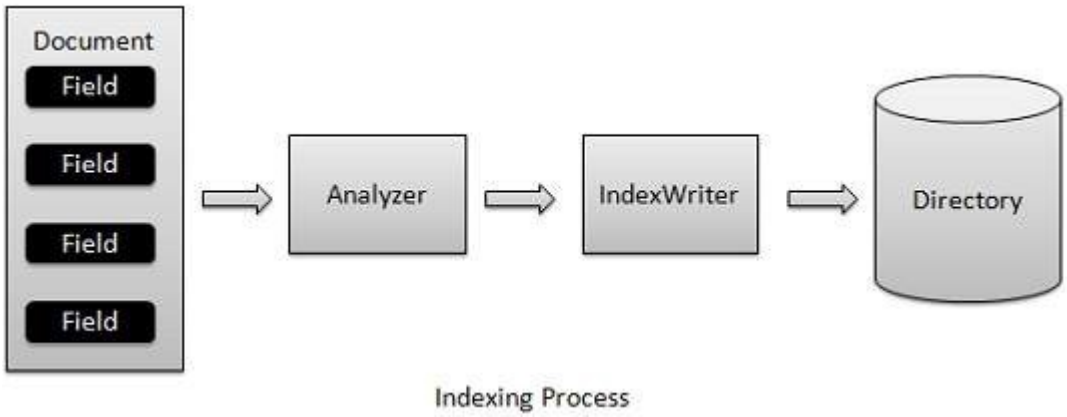
Documents	ID	Term	Document
The bright blue butterfly hangs on the breeze	1	butterfly	1
	2	blue	1, 2
	3	bright	1, 2
	4	breeze	1
Under blue sky, in bright sunlight, one need no search around	5	hangs	1
	6	need	2
	7	search	2
	8	sky	2

Lucene in Neo4j

Neo4j supports full-text search where the index in powered by Apache Lucene. The Lucene query language can be combined with Cypher (queries written in Neo4j) to manipulate the search results. Lucene indexes are created on particular node properties. This enables to use Lucene's query syntax to find nodes within a cypher query. To serve relevant results, Lucene adds more metadata and data structures to the index such as term frequency, doc frequency, term offsets, term positions, etc.

As Lucene has a white space analyzer, any words or names with spaces must become a new index entry. So, the words are split out based on whitespaces and this becomes the collection of index entries. This process is related to full-text search context.

- Full-text search index – There are 2 steps in Lucene indexing pipeline. The first step is the "Analyzer" step. This step is responsible for the preprocessing of the text. In the second step, the results of the "Analyzer" step are stored in the index by the "IndexWriter".
- Whenever a cypher query is used in Neo4j to do a search, it's Lucene behind the scenes that is converting the cypher to a Lucene query and does the search task.



Indexing Process

Image Source: Tutorials Point

Conclusion

Lucene is not a database; it is just a java library. It is used for relevant information retrieval, which cares about finding and describing data. It is not used as a database management system. It is an excellent building block for high-performance indices of our data. Search engines such as Elasticsearch and Neo4j are essentially just a wrappers on Lucene that use the good parts of Lucene for information retrieval and build their own layer on top of it to provide extra features and make it easy to use for people. Elasticsearch stores JSON binary large objects (blobs) inside a Lucene field, called "_source". Among all the java libraries, Lucene is very well documented.

By using Lucene's API, we can build our own index format. The systems that are built with Lucene often look like "NoSQL" databases.

Lucene should be used when we want to index textual documents of any length and search for text within those documents, which returns a ranked list of documents that matched the search query. The classic example is search engines, like Google, that uses text indexers like Lucene to index and query the content of web pages.

References

- I have used Elasticsearch and Neo4j extensively at my workplace. I used all the knowledge that I gained from that and added to this review.
- Tutorials Point – https://www.tutorialspoint.com/elasticsearch/elasticsearch_basic_concepts.htm, https://www.tutorialspoint.com/neo4j/index.htm
- Towards Data Science - https://towardsdatascience.com/exploring-the-full-text-search-index-in-neo4j-on-a-movies-dataset-3cddca69db7a
- Medium – https://medium.com/@karkum/introduction-to-apache-lucene-7d65f67f5231
- StackOverflow – https://stackoverflow.com/questions/27793721/what-is-the-difference-between-lucene-and-elasticsearch