Abhishek Shinde
CS 410: Text Information Systems

Technology Review

Introduction

Text Retrieval Systems, or Search Engines

Suppose you want to efficiently retrieve a relevant set of text documents from a large corpus of documents. These text document sources can comprise from a multitude of sources, such as tweets, articles, or emails, so the additional complexity of the structure of text data is clearly evident. A clear solution for this prevalent task is text retrieval systems. It is these systems, or search engines, that power daily searches we make to find pertinent information in a fast and applicable manner.

In this technology review, I will be discussing about Apache Lucene, which is a powerful Java-based search engine library. After examining this tool's capabilities, I will dive into an open source implementation of Apache Lucene: Apache Solr. The focus for the Apache Solr technology will be about the underlying data indexing and search workflow.

Body

Apache Lucene: A Modern Framework for Search Engines

Apache Lucene Core, one of the releases of Apache Lucene, consists of a Java search library, which is utilized by many search engines. There are two major components of this utility: indexing and searching. In terms of indexing, Lucene Core devises the core data structure as an inverted index, which is simply a mapping of keywords to pages [1]. In addition, the indexing is performant; the index size is significantly less than indexed text size. Incremental indexing is fast as well, matching that of batch indexing. For searching, the library offers ranked searching, with flexibility in the types of queries. Since ranking functions are critical for the searching implementation, Lucene Core offers integration of ranking models, such as *Okapi BM25* and *Vector Space Model*. Many other configurations are available through this library feature as well [2].

Check out the following documentation for the implementation of the Lucene API. The library consists of various packages, such as *codecs*, *search*, and *index*, which applications can leverage to build robust search engines. Additionally, it provides a sample workflow for developers to get started on creating an index and building queries. Many platforms have been built on top of Lucene Core, such as databases, search servers, and search applications [2].

Overview of Apache Solr

Out of the platforms constructed from Lucene Core, Apache Solr is one of the prominent enterprise search servers. There are several features that highlight this technology. First and

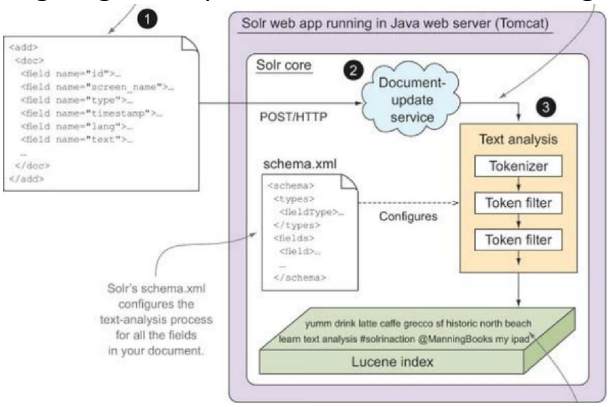
most important, it is primarily used as a search platform, in which the results from search are sorted by relevance to the user. To increase the usability and integration of this technology, developers can communicate with it through REST API. Most modern platforms implement restful services, so client communication with Solr is practical [3]. Solr is "enterprise ready", meaning that the server can be deployed to any machine and is scalable. *SolrCloud* is recommended for such use cases as it handles distribution and replication of indices across machines. According to the Solr guide, famous Internet sites like Ebay and Macy's use Solr [4].

This is a sample workflow of how to use Solr:
Build Schema -> Deploy Solr -> Supply Documents to Solr -> Apply and Use Search Feature

Defining Schema and Providing Indexing

In Solr, data is stored as documents. So, think of a document as a composure of some entity. For example, a car document would consist of model, color, price, etc. In Solr's representation, these are called fields which consist of field types. Now where does the index come into play? Well, these fields are added to the index, since after all it is an inverted index. This allows for quickly looking up documents that have the matched fields in them. To store the metadata and details for the field types, Solr updates and records this information in a schema file. Solr provides a schema API that allows reading and maintaining this file. Alternatively, there is a *Schemaless Mode* as well for a different approach [4].

Going back to the topic of indexing, Solr offers a multitude of ways to index data. These are *index handlers*, *Solr Cell with Apache Tika*, and *Java Client API*. Since JSON is one of the most popular file formats, I will be reviewing how JSON-formatted data can be indexed into Apache Solr through an index handler. So, in terms of mapping JSON into Solr's representation of data, there is an option to directly store without transforming the data. Solr offers sending update requests with parameters specifying how to split a JSON document and map the fields to the schema. Some useful parameters are *split*, *mapUniqueKeyOnly*, and *srcField*. There is an endpoint available to send these configurations. To simplify the mapping, wildcards can be used. Sometimes multiple documents need to be indexed at once, so Solr has the capability to bulk index [4]. The following diagram depicts the workflow of indexing [5].

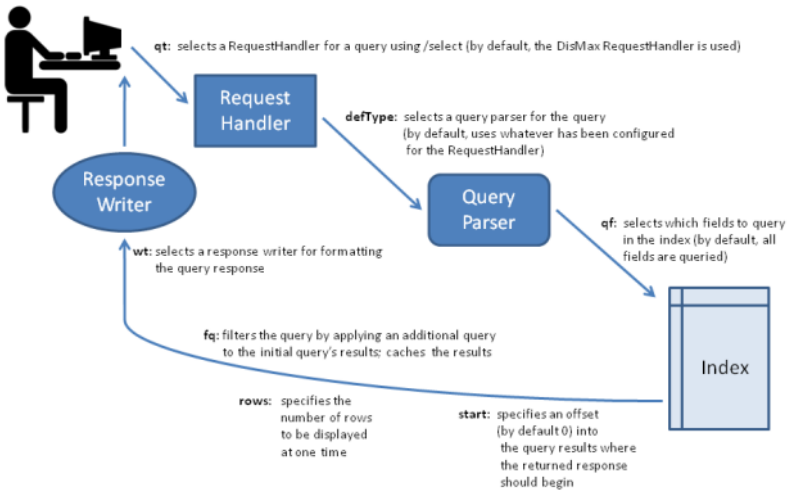


https://medium.com/techshots/introduction-to-apache-solr-d3e734a5c346

Abhishek Shinde
CS 410: Text Information Systems

Searching in Solr

After documents have been indexed into the Solr platform, it is ready to accept queries! A search query goes through several steps in the pipeline. First, it goes through a *request handler* that identifies processing logic to be use. The query string is then passed to the *query parser* component; this basically extracts the terms and parameters from the input. These parameters define how to either improve the query or return a preferred response structure. In addition, there are a variety of query parsers, such as the *DisMax query parser* that is similar to the one used by Google. Developers should select it based on priority for greater precision or tolerance to errors. The fields are passed onto the index and the results from it are sent to the *response writer* to finalize the result response. There are a variety of response writers, such as *JSON* and *XML Response Writers*. For improved performance, Solr provides specification of a filter query that allows for caching results for a given query. From the documentation, I found the following diagram helpful to visualize the pipeline [4].



https://lucene.apache.org/solr/guide/6_6/overview-of-searching-in-solr.html#overview-of-searching-in-solr

Conclusion

In this review, the high-level aspects of Apache Lucene were examined. Built on top of this framework, we have the Apache Solr software toolkit. Some of Solr's main features are addressed in this review, and then the two key workflows are assessed: indexing and searching. The takeaway is that Apache Solr is a robust, reliable, and practical solution for implementing a search feature into a system.

Abhishek Shinde
CS 410: Text Information Systems

References:

[1] Baeldung. “Introduction to Apache Lucene.” Baeldung, 2 Nov. 2018, www.baeldung.com/lucene.

[2] “Apache Lucene Core.” *Apache Lucene - Apache Lucene Core*, lucene.apache.org/core/.

[3] “Apache Solr - Overview.” *Tutorialspoint*, www.tutorialspoint.com/apache_solr/apache_solr_overview.htm.

[4] *Apache Solr -*, lucene.apache.org/solr/.

[5] Nigam, Sonam. “Introduction to Apache Solr.” *Medium*, TechShots, 19 May 2019, medium.com/techshots/introduction-to-apache-solr-d3e734a5c346.