

Technology Review: Apache Lucene – Implementation of Elasticsearch

Introduction:

Google posted an article on their research blog called “Speed Matters” with the results that slowing down a search-results page by even 100 to 400 milliseconds impacted the number of searches a user was willing to perform. Users subject to the slower loads would perform 0.2% to 0.6% fewer searches.

Apache Lucene is an open-source library (java based) which provides the standard for search and indexing performance. Lucene is the search core for Elasticsearch.

In this review, we will try to understand how Elasticsearch efficiently uses Lucene platform for faster searches & how can it be implemented for data search.

Why Elasticsearch?

Elasticsearch is an open source, distributed, RESTful, full text search engine built on top of Apache Lucene. On top of what Lucene already provides, Elasticsearch adds an HTTP interface, meaning application doesn't need to be built using java anymore and is distributed by default, meaning you won't have any trouble scaling your operations to thousands of queries per second.

Elasticsearch provides near real time search and analytics for all types of data. Whether you have structured or unstructured text, numerical data or geospatial data, Elasticsearch can efficiently store and index it in a way that supports fast searches.

While Lucene would need java to be installed on every machine where search need to be performed, Elasticsearch resolves this problem by its default distributed outlook.

How is Elasticsearch faster?

Elasticsearch's speed is attributed to two main factors:

1. How the data is stored and represented
Data in Elasticsearch is indexed, an index can be thought of as an optimized collection of documents and each document is a collection of fields, which are the key-value pairs that contain your data. By default, Elasticsearch indexed all data in every field and each indexed field has a dedicated, optimized data structure. The ability to use per-field data structures to assemble and return search results is what makes Elasticsearch so fast.
2. Instead of searching the text directly, Elasticsearch searches an index instead.
This is like retrieving pages in a book related to a keyword by scanning the index at the back of book, as opposed to searching every word of every page of the book. This type

of index is called an inverted index, because it inverts a page-centric data structure (page -> words) to a keyword centric data structure (word -> pages)

Elasticsearch Implementation:

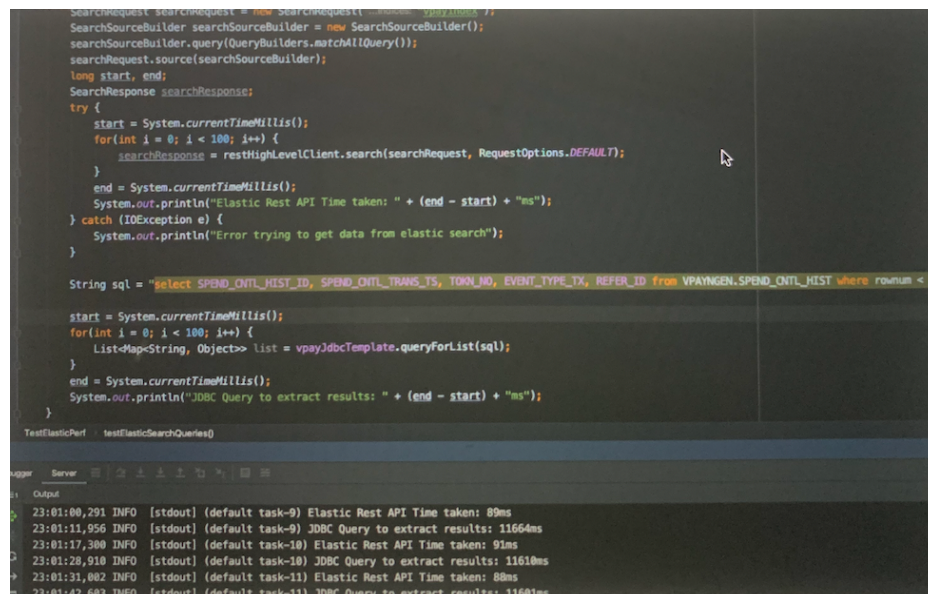
Elasticsearch can be downloaded from <https://www.elastic.co/downloads/elasticsearch>
Kibana is a free and open user interface that lets you visualize Elasticsearch data and navigate the Elastic stack.

Note: We need data to be ingested in Elasticsearch so that it can index the data. We can ingest the data manually through Elasticsearch rest APIs or connect to other data sources using plugins.

Logstash is a free and open server-side data processing pipeline that ingests data from a multitude of sources, transforms it and then sends it to Elasticsearch. If there is huge data, you can batch the data ingestion so that it does not error out.

Once data is ingested and indexed, power of Elasticsearch can be fully utilized to query the data. Elasticsearch also has a JAVA REST client too trigger elastic REST APIs.

When a basic performance test is performed to compare an elastic query vs Java JDBC query for 10 records – JDBC average was around 116ms per request bs elastic search was impressive 90ms (25% faster)!! We can get more insights using better performance tests with remote elastic server and huge amounts of data where searching complex queries are always slow in oracle database with its relational model.



```
SearchRequest searchRequest = new SearchRequest("index");
SearchSourceBuilder searchSourceBuilder = new SearchSourceBuilder();
searchRequest.source(searchSourceBuilder);
long start, end;
SearchResponse searchResponse;
try {
    start = System.currentTimeMillis();
    for(int i = 0; i < 100; i++) {
        searchResponse = restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);
    }
    end = System.currentTimeMillis();
    System.out.println("Elastic Rest API Time taken: " + (end - start) + "ms");
} catch (IOException e) {
    System.out.println("Error trying to get data from elastic search");
}

String sql = "select SPEND_CNTL_HIST_ID, SPEND_CNTL_TRANS_TS, TORN_NO, EVENT_TYPE_TX, REFER_ID from VPAYNGEN.SPEND_CNTL_HIST where rownum < 1";

start = System.currentTimeMillis();
for(int i = 0; i < 100; i++) {
    List<Map<String, Object>> list = vpayJdbcTemplate.queryForList(sql);
}
end = System.currentTimeMillis();
System.out.println("JDBC Query to extract results: " + (end - start) + "ms");

TestElasticPerf testElasticSearchQueries()

Output
23:01:00,291 INFO [stdout] (default task-9) Elastic Rest API Time taken: 89ms
23:01:11,956 INFO [stdout] (default task-9) JDBC Query to extract results: 11664ms
23:01:17,300 INFO [stdout] (default task-10) Elastic Rest API Time taken: 91ms
23:01:28,910 INFO [stdout] (default task-10) JDBC Query to extract results: 11610ms
23:01:31,002 INFO [stdout] (default task-11) Elastic Rest API Time taken: 88ms
23:01:42,603 INFO [stdout] (default task-11) JDBC Query to extract results: 11601ms
```

Conclusion:

Elasticsearch has lot other features like boosting the score of search results, support fuzziness, perform aggregations like finding the average of all the documents for a field that has numeric values etc. Elasticsearch on Lucene platform offers a reliable, fast searching & scalable solution for large data.