ISSN:2320-0790

An International Journal of Advanced Computer Technology

ElasticSearch

An advanced and quick search technique to handle voluminous data

Manda Sai Divya[1], Shiv Kumar Goyal[2]

[1] Master of Computer Application
Vivekananda Educational Society's Institute Of Technology, Chembur
Mumbai, Maharashtra
msaidivya.mca@gmail.com

[2] *Deputy Head of the Department, Master of Computer Application*
Vivekananda Educational Society's Institute Of Technology, Chembur
Mumbai, Maharashtra
shiva_goyal1@rediffmail.com

Abstract: A horizontally-scalable, distributed database built on Apache's Lucene that delivers a full-featured search experience across terabytes of data with a simple yet powerful API.

Keywords: Lucene, Big data, Mapping, Indexing, Searching, Key-Value pairs

I. INTRODUCTION

Elasticsearch is a new database built to handle huge amounts of data volume with very high availability and to distribute itself across many machines to be fault- tolerant and scalable, all the while maintaining a simple but powerful API that allows applications from any language or framework access to the database.

A horizontally-scalable, distributed database built on APACHE'S LUCENE that delivers a full-featured search experience across terabytes of data with a simple yet powerful API.

Many companies use elasticsearch to help them deploy powerful search capabilities in their applications that are easy to set up, scalable and built for the cloud.

ABOUT LUCENE

Apache's Lucene is an open-source Java library for text search. The Lucene project has been growing for more than a decade and has now become the standard reference for how to build a powerful yet easy to integrate, open- source search library.

Lucene, as a search library, must be wrapped with an interface to allow its features to be used by an application. Many such interfaces have been built for different platforms and use cases, e.g. SOLR.

An interface like SOLR, however, is designed for a world in which a single server can handle the full workload of indexing and querying the data. When the data volume begins to increase past a limit, SOLR (and similar interfaces to Lucene) become unwieldy to use: the same problems of sharding, replication, and query dispatching that occur in RDBMS systems begin to occur again in this context. And just as various methods exist for dealing with these difficulties in the RDBMS world, various tools exist for shard creation and distribution around SOLR.

But just as the right solution to big data databases means moving away from RDBMS into NoSQL technologies, the right solution to scaling Lucene is to move away from tools like SOLR and use a tool built from the ground-up to work with terabytes of data in a horizontally scalable, distributed, and fault-tolerant way: Elasticsearch!

II. LUCENE – ELASTICSEARCH - BIGDATA

Elasticsearch is best thought of as an interface to Lucene designed for BIGDATA from the ground up. The complex feature set that Lucene provides for searching data is directly available through Elasticsearch, as Lucene is ultimately the library that's used for indexing and querying data. This also means that plugins that work with Lucene will work with Elasticsearch out of the box.

The features that Elasticsearch itself provides around Lucene are designed to make it the perfect tool for full-text search on big data.

III. UNDERSTANDING ELASTICSEARCH

A. Basic features [1]

1) *REST API:* ElasticSearch stores/retrieves objects via a REST API. Convenient PUT, POST, GET, and DELETE APIs are provided that implement version checks (optionally on PUT), generate ids (optionally on POST), and allow you to read your own writes (on GET). This is what makes it a key value store. [2]

2) Key Value Store: In ElasticSearch, every piece of data has a defined index and type. You can think about an index as a collection of documents or a table in a database. However, here the documents added to an index have no defined structure and field types.
Objects have a type and go in an index. So, from a REST point of view, the relative uri to any object is /{index}/{type}/{id}. You create indices and types at runtime via a REST API.

3) *Multi-tenancy:* You can create, update, retrieve and delete indices. You can configure the sharding and replication on a per index basis. That means ElasticSearch is multi tenant and quite flexible.

4) Mapping: ElasticSearch indexes documents you store using either a dynamic mapping, or a mapping you provide (recommended). That means you can find back your documents via the search API as well.

Sharding & Replication: Better availability and performance are achieved through the replicas (copies of index parts).

B. Search API in Elasticsearch

This is where Lucene comes in. Unlike the GET, search does not allow you to read your own writes immediately because it takes time for indices to update, and replicate and doing this in bulk is more efficient.

The search API is exposed as a _search resource that is available in at the server level (/_search), index level (/{index}/_search, or type level (/{index}/{type}/_search). So you can search across multiple indices, because ElasticSearch is replicating and sharding, across multiple machines as well.

When returning search results, ElasticSearch includes a _source field in the result set that by default contains the object associated with the results. This means that querying is like doing a multi-get, i.e. expensive if your documents are big, your queries ar expensive, and your result sets are large. This means that you have to carefully manage how you query your dataset.

The search API supports the GET and POST methods. Post exists as a backup for clients that don't allow a json body as part of a GET request. The reason you need one is that ElasticSearch provides a domain specific language (json based, of course) to specify complex queries. You can also use the Lucene query language with a q=query parameter in the GET request but it's a lot less powerful and only useful for simple stuff.

C. Cluster in Elasticsearch

ElasticSearch is clustered by default. That means if you start two nodes in the same network, they will hook up and become a cluster. This doesnot require any special configuration.

ElasticSearch can work as a standalone, single-search server. Nevertheless, to be able to process large sets of data and to achieve fault-tolerance, it can be run on many cooperating servers. [1] Collectively, these servers are called as a CLUSTER and each of them is called a node.

Large amounts of data can be split across many nodes via index sharding (splitting it into smaller individual parts) and ElasticSearch replicates across whatever nodes are available in the network. Typically, you configure it in different ways for running in different environments.

ElasticSearch is built for big deployments in e.g. Amazon AWS, Heroku, or your own data center. That means it comes with built in monitoring features, a pluggable architecture for adapting to different environments and a lot of other stuff you need to run in such environments. This is a nice contrast to SOLR, which doesn't do any of these things out of the box.

Application Support

Elasticsearch isn't just a search engine; it's a full-fledged database, and you can build an entire frontend application on top of it.
Elasticsearch supports multiple indices (databases) and multiple mappings (tables) per index. This feature, combined with the complex document structure Elasticsearch allows, lets you build the complex data models that support applications.

And, in addition to being able to execute rich search queries across the data, Elasticsearch allows the more "traditional" operations that define an application database:

listing records, creating records, updating records, and deleting records. These features give you what you need to build a traditional database-driven, read/write application on top of the same database that lets you do full-text search and complex queries, all with horizontal scalability built-in from the ground up.

*D. **Working with JSON over HTTP***

JSON over HTTP has effectively become the "lingua franca" (bridge language) of services in a system. Even though Elasticsearch is a Java solution, JSON over HTTP makes it really easy for people to develop in Ruby, Perl, and other languages.

HTTP is the wire format and JSON is the payload. It's easily consumable by any language. In addition, you can now state a response in JSON and stream it directly to the browser.

When you give elasticsearch a search request, you can also ask for a histogram, like the number of tweets per day in the past year. It is returned in a structure that is ready to be thrown into any charting library. It's not just HTTP and JSON, it's the data or object structure—which make it easy to consume.

Indexing in ElasticSearch

ElasticSearch is able to achieve fast search responses because, instead of searching the text directly, it searches an index instead.

This is like retrieving pages in a book related to a keyword by scanning the index at the back of a 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 uses Apache Lucene to create and manage this inverted index.

In ElasticSearch, a Document is the unit of search and index.

An index consists of one or more Documents, and a Document consists of one or more Fields.

In database terminology, a Document corresponds to a table row, and a Field corresponds to a table column.

E. Query DSL

The Query DSL is ElasticSearch's way of making Lucene's query syntax accessible to users, allowing complex queries to be composed using a JSON syntax.

The main structure of a query is roughly:

```
curl -X POST
"http://localhost:9200/blog/_search?pretty=true" -d '
{"from": 0,
"size": 10,
"query" : QUERY_JSON,
FILTER_JSON,
FACET_JSON,
SORT_JSON
}'
```

IV. ELASTICSEARCH VS OTHER OPEN SOURCE SEARCH ENGINES [3]

ElasticSearch makes data exploration very easy.

First of all, there is a strong principle or architectural notion that things should be easy and simple. When people start to set-up, deploy, and use elasticsearch, it is easy to set-up 2, 4, 6, 10—as many nodes as you want. It's easy to set up a cluster too.

When you start developing, it's easy to start using documents as JSON documents. The API makes it easy to use different languages like Java, Ruby, Perl, Python, and more. In runtime, elasticsearch manages distribution—adding a node is quite easy and data is redistributed automatically.

If you think about search as a process, elasticsearch goes beyond free-text. Users and developers want valuable information from their data regardless of the form. While elasticsearch does free-text search very well, you also want structured search, analytics, aggregations, facets over the data, and more. All these are tied together nicely. For example, let's say you are indexing a social string from Twitter like a lot of our customers do. You can easily set it up and ask questions like, "find all the tweets about the president." This is a free-text search over a bunch of tweets. Then you can say, "Find all the tweets about the president when tweeted from Idaho in the past month." We sprinkle a bit more structure here by adding the location and time period. Then, we can ask it do the same thing, but break it down into number of tweets per day to see a trend over time. So, we end up with a metric and several dimensions to show something of value. After the initial query is set up, we can easily change the name of the president to any other person, and elasticsearch reflects the result—set in real-time. Then, we can change from a person to a topic like "flu epidemic," and the result-set is reflected in real-time again.

For developers, it's very powerful how easy data exploration becomes with elasticsearch.

V. ELASTICSEARCH BASIC WORKFLOW

The workflow can be explained in brief as follows:

- Documents are uploaded or stored; they may be of any type and any size and in any number.
- Then the JSON Builder converts these documents from their respective type to JSON documents.

- Now, it's the duty of the Tokenizer, to break down the data into individual words.

- These words are indexed and mapping is also done so as to group the similar type of words into one mapping type. This ensures the faster retrieval of text as per the query fired by the user.

- The parser will parse the query and accordingly search and retrieve the searched text from the indexed documents.

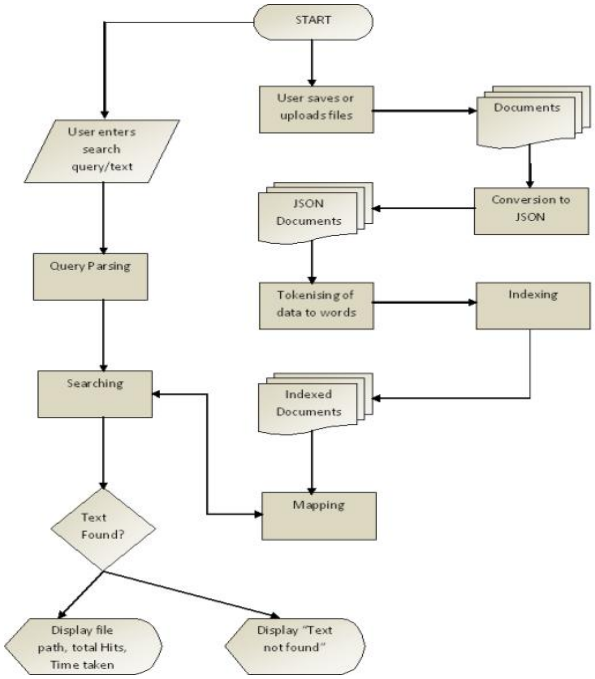

Fig.1. ElasticSearch Basic Workflow

VI. ELASTICSEARCH: THE COMPANY!

Shay Banon created Compass in 2004. [4] While thinking about the third version of Compass he realized that it would be necessary to rewrite big parts of Compass to "create a scalable search solution". [4] So he created "a solution built from the ground up to be distributed" and used a common interface, JSON over HTTP, suitable for programming languages other than Java as well. Shay Banon released the first version of ElasticSearch in February 2010.

Elasticsearch, the company behind the popular real-time search and analytics open source project, was highlighted byThoughtWorks, the global technology consultancy, as a leading go-to search platform in its bi-annual Technology Radar report. [5]

Read by thousands of technology leaders, this inclusion is validation of Elasticsearch, which has more than 2.5 million downloads to date. Companies around the world are using Elasticsearch to explore and understand large sets of data easier and more cost effectively than with other solutions.

The 2013 Technology Radar report recognized Elasticsearch across a number of criteria including: [5]

- **Ease of Use:** The Elasticsearch platform is an extensible, multi-tenanted, and horizontally scalable search solution. It allows complex data structures to be indexed and retrieved quickly and simply.

- **Ease of Operation**: The platform provides an elegant model for operation with automatic discovery of peers in a cluster, failover, and replication. Elasticsearch can be extended with a plugin system from which new functionality can easily be added.

- **Credible Community:** The users surrounding the Elasticsearch open source tool are quite vibrant as illustrated by the number of client libraries available in languages like Java, C#, Ruby, and JavaScript.

VII. SUMMARY

Data flows into your system all the time. The question is … how quickly can that data become an insight? With Elasticsearch, real-time is the only time.

Elasticsearch allows you to start small, but will grow with your business. It is built to scale horizontally out of the box. As you need more capacity, just add more nodes, and let the cluster reorganize itself to take advantage of the extra hardware.

Elasticsearch uses Lucene under the covers to provide the most powerful full text search capabilities available in any open source product. Search comes with multi-language support, a powerful query language, support for geo-location, context aware did-you-mean suggestions, auto complete and search snippets.

Complex real world entities can be stored in Elasticsearch as structured JSON documents. All fields are indexed by default, and all the indices can be used in a single query, to return results at breath taking speed.

It is continuously evolving and there are new versions coming up in the days to come for text analytics.

It is ultimately going to herald a new dawn in the field of full-text search as well as text analytics.

REFERENCES

[1] "The Power of Elastic Search", A white paper by Infochimps.

[2] "ElasticSearch Server", Rafat Kuc, Marek Rogozinski.

[3] "Q&A with Shay Banon: 10 "Bonsai Cool" Things About elasticsearch" - 2013, Adam Bloom.

[4] "The Future of Compass", Shay Banon.

[5] "Elasticsearch Platform Recommended as "Adopt" by ThoughtWorks", Report: Technology Radar 2013.

[6] "Elastic search or other Lucene for HBase?" - June 2010, Otis Gospodnetic.

[7] " Elastic Search: Distributed, Lucene-based Search Engine" May 2010, Sematext Blog.

[8] "ElasticSearch at Berlinbuzzwords" 2010.