From Apache Lucene to Elastic Stack: The Path of Evolution

Fengchao Wang

Introduction

Apache Lucene is a full-text search engine library (1). The design, implementation and usage are well documented in official documentations (2) as well as in blogs (3) and tutorials (4). Though a powerful library, much work needs to be done to conveniently utilize the functionality of Lucene. Elasticsearch is one of the most popular distributed search engines built on top of Lucene (5). From an engineer's perspective, this review compares Apache Lucene and Elasticsearch, focusing on the gap between the functions provided by Lucene and user demand. It covers the features of Apache Lucene, the limitations, the problem Elasticsearch solves as an out-of-the box search engine. Overall, it talks about how Elasticsearch gradually evolved from Lucene into Elastic Stack.

Apache Lucene: The Origin

As a full-text search library, Lucene primarily supports two operations: indexing and query. On high level, Lucene ingest documents by firstly process them through analyzers and tokenizers, then construct the inverted index (6). The indices are stored in files called "segment", which implements features of LSM tree to follow the "spill and merge" pattern for benefits in performance and update support (7). On the query side, Lucene supports typical query types in text retrieval, such as term, text, boolean, prefix and fuzzy (3). In recent versions of Lucene, it also supports Geospatial query (8).

You might have already noticed that though rich query types, Lucene does not support aggregations. Most real-world use cases require both real-time full-text search and multi-dimension aggregation. It is very hard to convince colleagues to use one storage system for search and another for aggregation at the same time since it introduces two copies of data and result in discrepancy in user experience unless making extra effort to synchronize. The ideal product for most scenarios is a system that performs full-text search and some aggregation regardless of the performance, at least.

Another gap between Lucene and user requirement is scalability and reliability. As a library, Lucene runs on single machine and manage segments on local storage. A single index has the limit of 2.14 billion documents (9). Local storage is more vulnerable to data loss. Running on a single machine also limits the performance of indexing and query. A distributed system that manages multiple Lucene shards can maximize the parallelism and replicate the storage segment to provide better scalability, reliability and performance.

Lastly, Lucene requires some boilerplate code to take care of the initialization, loading and configuration. Components to accept, handle and queue the query requests is also missing in Lucene. Most software engineers expect an out-of-the-box search system that have productivity in an hour or so. Let alone in a cloud native era, SAAS and serverless is gaining increasing adoption. Supporting ETL and visualization tools is also an important factor when choosing a data store. Next, I will talk about the evolution of Elastic Stacks, from Lucene to a whole ecosystem.

Elasticsearch: Distributed Lucene

Elasticsearch is core product of Elastic Stack. We can think of Elasticsearch as a "distributed Lucene", or a distributed system using Lucene as the local storage engine. The architecture follows a typical distributed storage system. Membership protocol and cluster coordination is self-developed, peer-to-peer protocol (10), partly because ZooKeeper was not published at the time of inception. Task and job scheduling use aggregator and worker pattern. Lucene is used as the physical storage engine on each node, and each Elasticsearch index is sharded into multiple Lucene indices that is distributed to multiple nodes. As a result, Lucene is managing multiple segments from multiple Elasticsearch indices on each node. These segments are replicated across nodes for performance and reliability. Due to the nature of Lucene segment, which is not splitable, the number of shards is fixed for an Elasticsearch index, unlike some storage systems that implements consistent hashing. This requires the user to plan well when setting the initial sharding number. All these are huge improvements from a bare library to an immediately available search engine.

Aggregation in Elasticsearch

Call back to aggregation. Elasticsearch implemented basic aggregation function with minimal performance. One cannot expect it to perform complex analytical aggregations like RDBMS or returns sub-second aggregation result like Druid and Dremio. It provides general aggregation functionality that covers most use cases with reasonable cost. The implementation consists of two components called 'fielddata' and 'doc_values' respectively, which is in fact columnar oriented indices of the same text data, represented in a different way (11). Lucene's inverted index solved the problem of "which document contains this word?". This data structure is not

designed to answer questions like “how many times does this word occurred in all documents?”. To answer the second question requires implementing a completely different data structure more often seen in traditional RDBMS. Like text queries, aggregation queries are accepted by a coordination node, then forward to worker/data nodes that performs local aggregation. Results are sent back to the coordinator node and coalesced to form the final result. Again, the data structure of Lucene nor Elasticsearch is designed to perform large scale aggregations. The performance is far from ideal for multi-dimension aggregations. For large cardinality estimation use case, Elasticsearch adopted HyperLogLog++ algorithm for estimation, which is also a common industry solution.

Deployment and Operation

One more problem remains to be solved: deployment and productionize. Operation and site reliability are as important and tricky as development. Developing in-house solution based on Lucene and managed to deploy to production is a challenging task. Elasticsearch solved this problem by providing it as service. There are multiple vendors of ES that provides service tiers from pure-cloud and on-premise services. Elastic Cloud and AWS Elasticsearch Service are two representative ones. They enable user to get Elasticsearch service instantly without the overhead of previsioning hardware and installing, configuring with complex automation tools. Elastic cloud has more flexibility by allowing user to control the deployment model, to the existing VPC and fine-tuning parameters. AWS Elasticsearch is a cloud managed service, and many functions are limited, such as scripting and re-indexing. These SAAS solutions greatly reduced the operation complexity.

The Ecosystem

Getting data in and out a storage engine is laborious yet important. Mature systems like RDBMS and Hadoop have rich toolkits to fulfill this task. Logstash is the tool designed for this purpose. Kibana, the dashboard built on top of Elasticsearch, is the visualization and analytical component. There are other components to support application performance tracing, time series data parsing and security auditing (12). These value-adding components solves actual problem that customers facing every day and further bridged the gap between Elasticsearch, a distributed search engine and problems in various domains.

Conclusion

Apache Lucene is laid the foundation of text search. Elasticsearch solved the scalability, reliability and aggregation problem in Lucene. Elasticsearch SAAS enabled user to use this service out-of-the-box without any overhead. Elastic Stack provides domain-specific solutions in visualization, ETL, application performance monitoring and security. Each step makes Lucene from solving an engineering problem closer to solving real world user problem.

Bibliography

1. **Lucene, Apache.** *Apache Lucene Official Website.* s.l. : https://lucene.apache.org/.

2. —. *Apache Lucene Documentation.* s.l. : https://lucene.apache.org/core/8_7_0/index.html.

3. **Baeldung.** *Introduction to Apache Lucene.* s.l. : https://www.baeldung.com/lucene.

4. **Edukera.** *Intro to Lucene.* s.l. : https://www.youtube.com/watch?v=vLEvmZ5eEz0.

5. **Elasticsearch.** *Elasticsearch Website.* s.l. : https://www.elastic.co/.

6. **Baeldung.** *Guide to Lucene Analyzers.* s.l. : https://www.baeldung.com/lucene-analyzers.

7. **MuLuo.** *Analyzing Lucene - Basic Concepts.* s.l. : https://zhuanlan.zhihu.com/p/35469104.

8. **Lucene.** *Lucene GeoPointdistanceQuery.* s.l. : https://lucene.apache.org/core/6_6_1/spatial/org/apache/lucene/spatial/geopoint/search/GeoPointDistanceQuery.html.

9. **JIRA, Solr.** *Clearly document the limit for the maximum number of documents in a single index* . s.l. : https://issues.apache.org/jira/browse/SOLR-3504#:~:text=Although%20the%20actual%20limit%20to,to%20approximately%202.14%20billion%20documents.

10. **Elasticsearch.** *A new era for cluster coordination in Elasticsearch.* s.l. : https://www.elastic.co/blog/a-new-era-for-cluster-coordination-in-elasticsearch.

11. —. *Elasticsearch Documentation.* s.l. : https://www.elastic.co/guide/en/elasticsearch/reference/current/fielddata.html.

12. —. *Elastic Observability.* s.l. : https://www.elastic.co/observability.

13. **Lucene, Apache.** *https://lucene.apache.org/core/.*