

Is the evolution of search algorithms finished?

Foidaş Andrei Ștefan

Group 933/2

Abstract

One fact cannot be denied, technology has drastically changed our lives. It has significantly reduced the effort needed to perform particular tasks and vastly increased efficiency and productivity. And computers especially play a very important role in the vast majority of fields in today's world. One of the main parts of computers is storing large amounts of data and solves to solve complex and hard problems. Information is a crucial aspect, and in this era of quick evolving and huge data, it is illogical to analyze it with classical algorithms or relational databases.

The main purpose of this paper is to emphasize on the idea that search algorithms are always on the verge of evolution and that there will always be an increasing demand for search algorithms to perform faster and on much larger data sets. We will be showing some ways in which search algorithms have evolved throughout the years but also showcase the beginning of a possible next generation of searching.

In this paper we will be using Amazon's OpenSearch and Elasticsearch services, a search engine that is capable of storing petabytes of data and perform searching, sorting and other operations in near real-time. To showcase one of the most powerful search engines we are using a small dataset of 10.000 elements and executing some test queries. The tests will be applied to distinct implementations and versions of the Elasticsearch principle with the goal of presenting the way this algorithm becomes more efficient with time.

The expected results will provide answers to the research questions as well as proving the efficiency and correctness of the newly and evolved search algorithms. Furthermore it will show that with the increasing demand for the current versions of the algorithms, they will have to continually evolve and new discoveries will have to be made, similar to the case of Quantum Computers.

Classification

AMS Mathematical Subject Classification

90B40 Search theory

ACM Computing Reviews Categories and Subject Descriptors

F.2 ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY

F.2.2 Nonnumerical Algorithms and Problems

Sorting and searching

1. Introduction

One of the most basic tasks a computer scientist faces is that of designing a search algorithm to solve a given problem, whether it's a simple linear search, NP problem or a query. With technology evolving so fast and demanding better performance times, a few questions arise:

- Are classical search algorithms deprecated, or still usable?

The time complexities of classical search algorithms are usually in the logarithmic range, which makes them usable in most scenarios, thus they are still widely used. Still, upgrades were needed, and achievable: many searching algorithms were improved and in various ways to fulfill the demands of a problem in a more efficient way.

- Can such algorithms be evolved, or did we hit the peak performance when it comes to searching?

Kfir Wolfson and Moshe Slipper in [1] "Evolving Efficient List Search Algorithms" talk about how they employed genetic programming to evolve an iterative algorithm for searching for a given key in an array of integers, but when it comes to sublinear search problems, the process of evolution poses a greater challenge and the results turn out to be an implementation of binary search.

Other approaches still exist when it comes to evolving search algorithms: [2] Rather than solving a problem by applying a variety of search algorithms, an idea would be deriving a language composed of potentially useful basic search primitives and applying generic programming to create search algorithms. Being an instance of meta-evolution: evolving search algorithms at the top level and applying each search algorithm to a problem instance at level two, it proposes a unique approach to finding an effective search strategy for a given problem.

One can also [3] apply a method where Graph Structured Program Evolution is applied for constructing a method of evolving search algorithms. Those algorithms are used for benchmark function optimization and for template matching problems.

- What is the next generation of searching, does it exist?

With the advent of Quantum Computers, search algorithms adapted too. [4] - [6] Quantum Computers are not limited to just two states. Qubits, the basic unit of quantum computing have the power to exist in more than one state at a time. While the classical computers only perform operations by manipulation of classical bits having two values 0 and 1, quantum bits can represent data in multiple states. This property of inheriting multiple states at a time gives quantum computers tremendous power over classical computers. With this power, the algorithms designed on quantum computers to solve search queries can yield results significantly faster than the classical algorithms. This is how algorithms like Grover's algorithm can reduce the time complexity of some of the classical algorithm problems from N to \sqrt{N} .

But since we live in an era where every millisecond is important when it comes to waiting, those classical algorithms can't fulfill the demand of searching millions of petabytes fast enough. New big data instruments, architectures and designs have appeared to drastically boost the productivity and efficiency of searching tasks. Specifically, Elasticsearch, an open source Apache Lucene based full-text search engine that provides near real-time search ability. It is a Java based search engine that is designed keeping cloud environment in mind solves issues of scalability, search in real-time, and efficiency that relational databases were not able to address.

In the experimentation section we will compare and analyze Amazon's OpenSearch Service to its predecessor, Amazon Elasticsearch Service. We will experiment on three different domains: one older version of Elasticsearch, the latest version of Elasticsearch and OpenSearch. Elasticsearch and OpenSearch are both search engines and data storage systems. They provide scalable search, fast access and response to very large volumes of data, as well as data visualization tools.

2. Original Contribution

The main goal of this study is to show that search algorithms are constantly evolving, performing faster and with increased accuracy.

The experiment is focused on directly comparing the three search domains through a series of different types of searches and sorting tests. We will not only look at the data resulted from the queries, but we will differentiate the search domains from multiple perspectives: indexes used, elapsed time, number of hits and successful finds such that meaningful conclusions can be derived. We will use the exact same queries on the exact same data set so we can get the most accurate results.

At the end of the experiment we will be able to not only the differences between the search engines, but see how they managed to evolve, what they prioritize now and which of suits a test field better.

The whole experimentation part is what sets this paper apart from other papers from this area of work since it explores in detail the performance difference of distinct search engines and search engine versions when it comes to various fields related to searching and sorting.

3. Experimenting

3.1. Experimentation data

To try to simulate the real usages of such search engines, I created a Python script that generates large JSON files with various data types. We will conduct a bundle of different search tests on files of different lengths to try to figure out which search engines excels and in which domain.

When it comes to data validation, the search engines will accept only valid JSON objects or files, with a specific "bulk" format, hence any invalid data will be instantly rejected. Moreover, the search engines will return information related to the state of the response such as how many items fetched were successful, skipped and failed.

3.2. Creating search domains

With the help of the Amazon Web Services we will generate three search domains with similar settings but different versions: one older version of the Elasticsearch Service, the newest version of Elasticsearch and one instance of the OpenSearch Service. All three domains are deployed for development and testing, using an instance of t3.small.search with three nodes and can be accessed publicly.

3.3. Uploading data

In order to conduct the tests we desire, we need some data to work with. Both OpenSearch and Elasticsearch represent documents in JSON format, so naturally we will upload data to the search domains using JSON files. But in order to upload the data, the JSON file needs to have a specific bulk format:

- For OpenSearch versions:

```
{"index": {"_index": "index", "_id": "id"}}
{"field1": "field1", {"field2": "field2" ...}
```

- For Elasticsearch versions:

```
{"index": {"_index": "index", "_type": "type", "_id": "id"}}
{"field1": "field1", {"field2": "field2" ...}
```

In order to generate JSON files with the specific bulk format I created a Python script that generates two files with the same JSON objects but in the two different bulk formats. The script can generate files with a specific number of elements and with different valid, randomly generated fields such as names, words, lists and numbers. It is important that the JSON files have a trailing newline.

For this experiment I generated files containing 10.000 objects: booksOS.json for OpenSearch and booksES.json for Elasticsearch. They contain the following information about books: the author, a couple of themes, the year it was written, a few characters from the book and its title. Note that all the information was randomly generated and it doesn't exist in reality.

For uploading the randomly generated JSON files we will use a common HTTP client, **curl**, for brevity and convenience. The following command will be used in a local directory to upload a file into a search domain:

```
curl -XPOST -u 'master-user:master-user-password' 'domain-endpoint/_bulk'
--data-binary @file-name.json -H 'Content-Type: application/json'
```

After we are done uploading the data, we need to reindex our data such that we can work with index aliases and store that index more cost-effective. For our example we will create an index pattern of the form: **books*** that will help us retrieve the data more efficient from the search services.

3.4. Comparing test results

While performing the experiments we can observe how the three search engines tend to fetch data using distinct number of shards: OpenSearch usually uses 13 shards to retrieve data, the new Elasticsearch uses 18 shards, and the old Elasticsearch uses just 9 shards. We will see how by organizing the data into larger number of shards can be both beneficial but can bring some downsides too.

The first test is a simple one: we will fetch all the books from the domains. We receive, as expected 10.000 documents (we can notice this in the “hits”: “value”: 10.000 sections). To no surprise, the time elapsed is very similar, we

OpenSearch



```
1 GET _search
2 {
3   "query": {
4     "match": {
5       "_index": "books"
6     }
7   }
8 }
9

1 {
2   "took": 21,
3   "timed_out": false,
4   "_shards": {
5     "total": 13,
6     "successful": 13,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 10000,
13      "relation": "eq"
14    },
15    "max_score": 1.0,
16    "hits": [
```

are working with a rather small dataset so we will not notice very big differences. The section “took” represents the milliseconds it took the search engines to give us our response, OpenSearch being the fastest with 21 milliseconds, followed by both instances of Elasticsearch with 25 milliseconds.

New Elasticsearch

```
1 GET _search
2 {
3   "query": {
4     "match": {
5       "_index": "books"
6     }
7   }
8 }
9
```

```
1 {
2   "took" : 25,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 18,
6     "successful" : 18,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 10000,
13      "relation" : "eq"
14    },
15    "max_score" : 1.0,
16    "hits" : [
```

Old Elasticsearch

```
1 GET _search
2 {
3   "query": {
4     "match": {
5       "_index": "books"
6     }
7   }
8 }
9
```

```
1 {
2   "took" : 25,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 9,
6     "successful" : 9,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : 10000,
12    "max_score" : 1.0,
13    "hits" : [
```

Now we will look into how the domains can manage a search where they have to match a certain prefix. Here they will fetch all the books where the author has a name starting with the letter “A”. Now we start to notice small fluctuations in the results we are receiving: the new version of Elasticsearch managed to return the most number of books with the given criteria, 524, followed by OpenSearch with 441 and the last one was the old version of Elasticsearch with only 401 documents. The elapsed time was again pretty similar between the three search engines. Having the data split into more shards can benefit when it comes to searching while matching a certain pattern.

OpenSearch

```
1 GET _search
2 {
3   "query": {
4     "match_phrase_prefix": {
5       "author": "A"
6     }
7   }
8 }
9
```

```
1 {
2   "took" : 26,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 13,
6     "successful" : 13,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 441,
13      "relation" : "eq"
14    },
15    "max_score" : 7.2206173,
16    "hits" : [
17    {
```

New Elasticsearch

```
1 GET _search
2 {
3   "query": {
4     "match_phrase_prefix": {
5       "author": "A"
6     }
7   }
8 }
9
```

```
1 {
2   "took" : 32,
3   "timed_out" : false,
4   "_shards" : {
5     "total" : 18,
6     "successful" : 18,
7     "skipped" : 0,
8     "failed" : 0
9   },
10  "hits" : {
11    "total" : {
12      "value" : 524,
13      "relation" : "eq"
14    },
15    "max_score" : 13.930409,
16    "hits" : [
```

Old Elasticsearch

```
1 GET _search
2 {
3   "query": {
4     "match_phrase_prefix":
5       "author": "A"
6   }
7 }
8
9
10 {
11   "took" : 24,
12   "timed_out" : false,
13   "_shards" : {
14     "total" : 9,
15     "successful" : 9,
16     "skipped" : 0,
17     "failed" : 0
18   },
19   "hits" : {
20     "total" : 401,
21     "max_score" : 14.054802,
22     "hits" : [
```

When it comes to failures and error handling, having more shards will result in more of them failing and can delay or affect the end result. In the test where we include a sorting functionality on document fields (which is not available for the Amazon Free Tiers) we can see that the most affected domain was the new Elasticsearch having almost half of its shards end in failures, while the OpenSearch only had a quarter fail and the old Elasticsearch a third. The end result in this case was not affected; they all managed to retrieve one document which had the word “item” in the title.

OpenSearch

```
1 POST /_search
2 {
3   "query": {
4     "term": { "title": "item" }
5   },
6   "sort": [
7     { "year": { "order": "asc" } }
8   ]
9 }
10
11 {
12   "took" : 39,
13   "timed_out" : false,
14   "_shards" : {
15     "total" : 13,
16     "successful" : 10,
17     "skipped" : 0,
18     "failed" : 3,
19     "failures" : [
```

New Elasticsearch

```
1 POST /_search
2 {
3   "query": {
4     "term": { "title":
5       "item" }
6   },
7   "sort": [
8     { "year": { "order":
9       "asc" } }
10 ]
11
12 {
13   "took" : 75,
14   "timed_out" : false,
15   "_shards" : {
16     "total" : 18,
17     "successful" : 10,
18     "skipped" : 0,
19     "failed" : 8,
20     "failures" : [
21     {
22       "shard" : 0,
```

Old Elasticsearch

```
1 POST /_search
2 {
3   "query": {
4     "term": { "title":
5       "item" }
6   },
7   "sort": [
8     { "year": { "order":
9       "asc" } }
10 ]
11
12 {
13   "took" : 15,
14   "timed_out" : false,
15   "_shards" : {
16     "total" : 9,
17     "successful" : 6,
18     "skipped" : 0,
19     "failed" : 3,
20     "failures" : [
21     {
22       "shard" : 0,
```

We can run one type of sorting on the free tier search engines, searching based on **_doc**. It is the most efficient sort order but it has no real use-case. It is only helpful when you want to use deep pagination. This test helps us see how the search engines are

OpenSearch

```
1 GET books/_search
2 {
3   "sort": [
4     {
5       "_doc": {
6         "order": "asc"
7       }
8     }
9   ],
10   "query": {
11     "match_phrase_prefix": {
12       "author": "A"
13     }
14   }
15 }
16
17 {
18   "took" : 69,
19   "timed_out" : false,
20   "_shards" : {
21     "total" : 5,
22     "successful" : 5,
23     "skipped" : 0,
24     "failed" : 0
25   },
26   "hits" : {
27     "total" : {
28       "value" : 523,
29       "relation" : "eq"
30     },
31     "max_score" : null,
32     "hits" : [
```

managing using the default number of shards (5 for OpenSearch and the new Elasticsearch and 3 for the old Elasticsearch) since the sorting is the first request and the `_doc` field is set by default for any kind of document uploaded to a search engine. We can see that when it comes to sorting, OpenSearch outperforms the new Elasticsearch when it comes to number of elements found and sorted but by a very small difference. The old Elasticsearch performs the worst with a difference of almost 100 hits.

In the last experiment we will test how the three domains run a script. The script on the right is pretty simple, it just filters all the books that have the year greater than 1900. This script searches through the entire domain, hence the few failures for data that does not have a year field, and for all three domains it returns the same number of elements (3802 elements).

The only distinctions appear when it comes to the number of the shards used: OpenSearch and the new Elasticsearch use 8 shards to fetch data while the old Elasticsearch uses only 6; and when it comes to the elapsed time: OpenSearch manages to execute the script and retrieve the data twice as fast as the Elasticsearch instances.

New Elasticsearch

```
1 GET books/_search
2 {
3   "sort": [
4     {
5       "_doc": {
6         "order": "asc"
7       }
8     }
9   ],
10  "query": {
11    "match_phrase_prefix": {
12      "author": "A"
13    }
14  }
15 }
```

```
1 {
2   "took": 55,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": {
12      "value": 512,
13      "relation": "eq"
14    },
15    "max_score": null,
16  }
```

Old Elasticsearch

```
1 GET books/_search
2 {
3   "sort": [
4     {
5       "_doc": {
6         "order": "asc"
7       }
8     }
9   ],
10  "query": {
11    "match_phrase_prefix": {
12      "author": "A"
13    }
14  }
15 }
```

```
1 {
2   "took": 73,
3   "timed_out": false,
4   "_shards": {
5     "total": 3,
6     "successful": 3,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 426,
12    "max_score": null,
13    "hits": [
14      {
15        "_index": "books",
16        "type": "book"
```

The script

```
1 GET /_search
2 {
3   "query": {
4     "bool": {
5       "filter": {
6         "script": {
7           "script": {
8             "source": "doc['year'].value > params.param1",
9             "lang": "painless",
10            "params": {
11              "param1": 1900
12            }
13          }
14        }
15      }
16    }
17  }
18 }
```

OpenSearch

```
1 {
2   "took": 446,
3   "timed_out": false,
4   "_shards": {
5     "total": 8,
6     "successful": 6,
7     "skipped": 0,
8     "failed": 2,
9     "failures": [
10    {
11      "index": "books",
12      "type": "book",
13      "reason": "script compilation failed"
14    }
15  ]
16 },
17 "hits": {
18   "total": {
19     "value": 3802,
20     "relation": "eq"
21   },
22   "max_score": 0.0,
23 }
```

```
{
  "took" : 871,
  "timed_out" : false,
  "_shards" : {
    "total" : 8,
    "successful" : 6,
    "skipped" : 0,
    "failed" : 2,
    "failures" : [
      {
        "type" : "search_phase_execution_exception",
        "reason" : "search_phase_execution_exception[no hits found]",
        "index_uuid" : "98661000-9866-437c-8000-000000000000",
        "index_name" : "test",
        "type_uuid" : "98661000-9866-437c-8000-000000000000",
        "type_name" : "test",
        "reason_uuid" : "98661000-9866-437c-8000-000000000000",
        "reason_name" : "test",
        "index_type_uuid" : "98661000-9866-437c-8000-000000000000",
        "index_type_name" : "test",
        "reason_type_uuid" : "98661000-9866-437c-8000-000000000000",
        "reason_type_name" : "test"
      }
    ]
  },
  "hits" : {
    "total" : {
      "value" : 3802,
      "relation" : "eq"
    },
    "max_score" : 0.0,
    "hits" : [

```

```
{
  "took" : 813,
  "timed_out" : false,
  "_shards" : {
    "total" : 6,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 1,
    "failures" : [
      {
        "type" : "org.elasticsearch.index.shard.ShardExceedsSizeLimitException",
        "reason" : "Shard size exceeds the limit of 100mb"
      }
    ]
  },
  "hits" : {
    "total" : 3802,
    "max_score" : 0.0,
    "hits" : [
      {
        "_type" : "tweet",
        "_id" : "1",
        "score" : 0.0,
        "source" : {
          "text" : "A tweet"
        }
      }
    ]
  }
}
```

After conducting the experiment the evolution of the Elasticsearch becomes very clearly visible and the new OpenSearch (even if it's only the first version) shows potential of possibly overtaking Elasticsearch in the future. There are still some areas where Elasticsearch will be the better option so some research should be done in order to see which one portrays a project better.

In the paper [7] “Large-Scale Image Retrieval with Elasticsearch” it is described how Elasticsearch was used to obtain outstanding performance improvement when it comes to content-Based Image Retrieval in large archives by transform CNN features into textual representations and indexing them such that the Elasticsearch Service can operate on them.

With the growth of the Earth Science Information Partner (ESIP) federation, came also the growth of diverse data sets to enable understanding of the Earth as a system. The diversity and need of combining the information presents a challenge to finding useful data for a given study. [8] To address this issue, the ESIP is developing a federated space-time query framework, based on the OpenSearch convention. The novelty of OpenSearch was that the space-time query interface became both machine callable and easy enough to integrate into the web browser's search box. This changed offered great flexibility, simplicity and queries would run in real-time.

These search engines can be used to solve critical real life problems too. [9] Mayo Clinic, a nonprofit American academic medical center focused on integrated health care, education, and research, used to generate 0.7-2.2 million HL7 V2 messages on a daily basis, which couldn't be real-time stored or analyzed even with multiple RDBMS-based systems. Using Big Data technology coupled with the Elasticsearch technology, the platform could process over 62 million HL7 messages per day while Elasticsearch indexes could provide ultrafast test searching (0.2 seconds per query) on an indexes of up to 25 million HL7-derived JSON documents.

In the end, we could clearly observe that the evolution of search algorithms is not finished, nor it will be in the near future. There is still room for improvement when it comes to the next generation of search algorithms such as Elasticsearch and search for Quantum Computers, and if we analyze the way technology evolved up to this point we can safely predict that in the future there will be more and more demands for faster and more efficient search algorithms working on even larger data sets.

Bibliography


- [1] Kfir Wolfson, Moshe Sipper: Evolving Efficient List Search Algorithms, pp. 158-169, EA 2009: Artificial Evolution (2009)
- [2] Brian J. Ross: Searching for Search Algorithms: Experiments in Meta-search, COSC TR CS-01-01 (2002)
- [3] Shinichi Shirakawa, Tomoharu Nagao: Evolution of Search Algorithms Using Graph Structured Program Evolution, pp. 109-120, EuroGP 2009: Genetic Programming (2009)
- [4] Avery Leider, Sadida Siddiqui, Daniel A. Sabol, Charles C. Tappert: Quantum Computer Search Algorithms: Can We Outperform the Classical Search Algorithms?, pp. 447-459, FTC 2019: Proceedings of the Future Technologies Conference (FTC) (2019)
- [5] Lov K. Grover: Quantum Mechanics helps in searching for a needle in a haystack (1997)
- [6] Giuseppe Amato, Paolo Bolettieri, Fabio Carrara, Fabrizio Falchi: Large-Scale Image Retrieval with Elasticsearch, SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (2018)
- [7] G. L. Long: Grover algorithm with zero theoretical failure rate (2001)
- [8] Lynnes, C., Beaumont, B., Duerr, R. E., Hua, H.: Federated Space-Time Query for Earth Science Data Using OpenSearch Conventions, American Geophysical Union, Fall Meeting 2009, abstract id. IN33F-04 (2009)
- [9] Dequan Chen, Yi Chen, Brian N. Brownlow, Pradip P. Kanjamala, Carlos A. Garcia Arredondo, Bryan L. Radspinner, Matthew A. Raveling: Real-Time or Near Real-Time Persisting Daily Healthcare Data Into HDFS and Elasticsearch Index Inside a Big Data Platform, IEEE Transactions on Industrial Informatics (2016)



Project timeline

⌵

Commits on Dec 17, 2021

finished.


 AndreiFoidas committed 17 seconds ago



 0c31ade 

⌵

Commits on Dec 16, 2021

worked on introduction and abstract


 AndreiFoidas committed 16 hours ago



 0b71f11 

⌵

Commits on Dec 14, 2021

original contribution, experimenting, result & conclusion


 AndreiFoidas committed 3 days ago


 e3f9924 

⌵

Commits on Dec 13, 2021

started final lab

 AndreiFoidas committed 4 days ago

 4677499 