Is the evolution of search algorithms finished?

Foidaș Andrei Ștefan

Group 933/2

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. Experimenting

1.1. Experimental modeling

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.

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

1.2. 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.

1.3. Rigorous experiment model

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.

2. Case Study

2.1. 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.

2.2. 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'
```

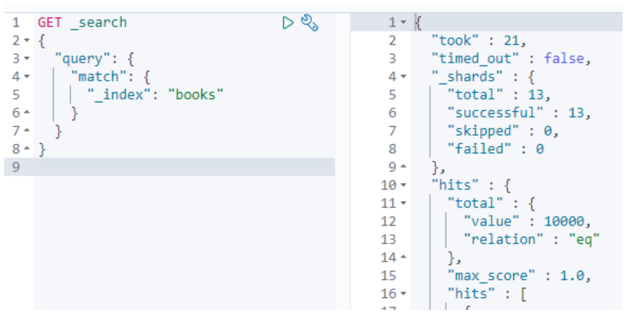
After we are done uploading the data, we need to reindex our data such that we can work wind index aliases and store that index more cost-effective. For our example we will create an index pattern of the form: **books*** that will help us retrieve the data more efficient from the search services.

2.3. 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 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 is our response, OpenSearch being the fastest with 21 milliseconds, followed by both instances of Elasticsearch with 25 milliseconds.

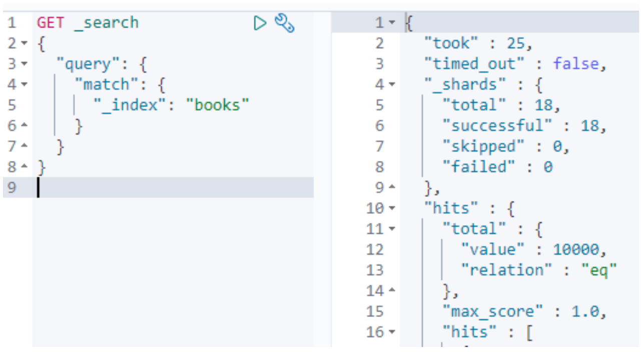
OpenSearch



```
GET _search
{
  "query": {
    "match": {
      "_index": "books"
    }
  }
}
```

```
{
  "took" : 21,
  "timed_out" : false,
  "_shards" : {
    "total" : 13,
    "successful" : 13,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
```

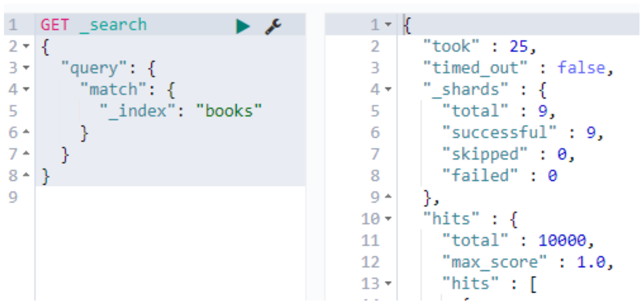
New Elasticsearch



```
GET _search
{
  "query": {
    "match": {
      "_index": "books"
    }
  }
}
```

```
{
  "took" : 25,
  "timed_out" : false,
  "_shards" : {
    "total" : 18,
    "successful" : 18,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
```

Old Elasticsearch

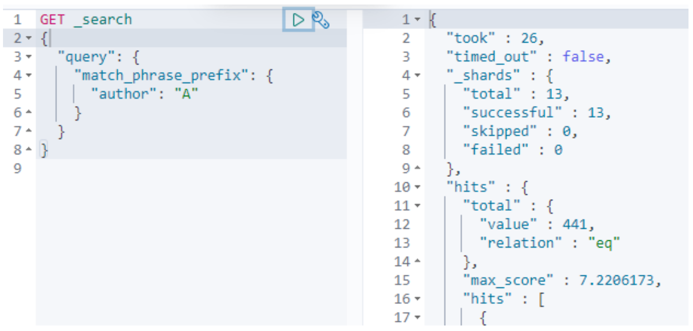


```
GET _search
{
  "query": {
    "match": {
      "_index": "books"
    }
  }
}
```

```
{
  "took" : 25,
  "timed_out" : false,
  "_shards" : {
    "total" : 9,
    "successful" : 9,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 10000,
    "max_score" : 1.0,
    "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

OpenSearch

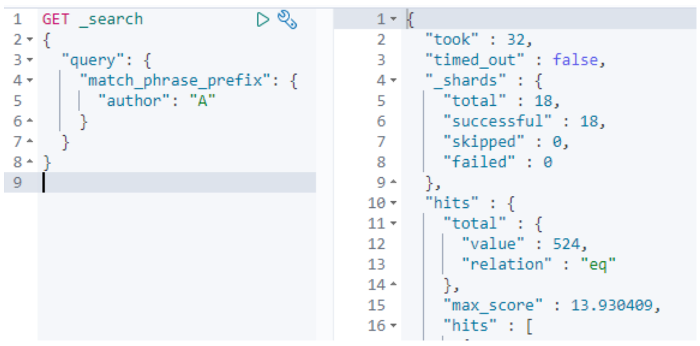


```
GET _search
{
  "query": {
    "match_phrase_prefix": {
      "author": "A"
    }
  }
}
```

```
{
  "took" : 26,
  "timed_out" : false,
  "_shards" : {
    "total" : 13,
    "successful" : 13,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 441,
      "relation" : "eq"
    },
    "max_score" : 7.2206173,
    "hits" : [
      {
```

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.

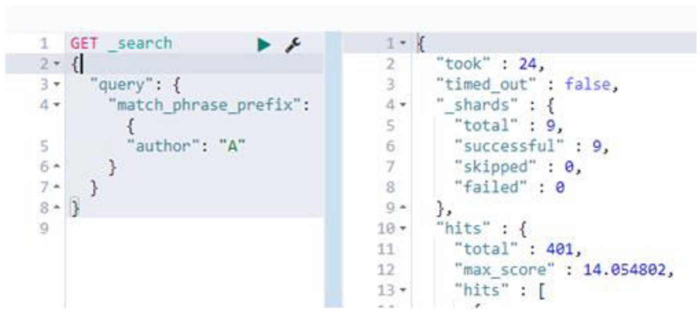
New Elasticsearch



```
GET _search
{
  "query": {
    "match_phrase_prefix": {
      "author": "A"
    }
  }
}
```

```
{
  "took" : 32,
  "timed_out" : false,
  "_shards" : {
    "total" : 18,
    "successful" : 18,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 524,
      "relation" : "eq"
    },
    "max_score" : 13.930409,
    "hits" : [
```

Old Elasticsearch

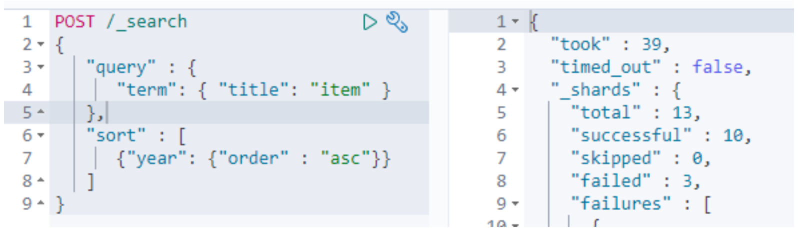


```
GET _search
{
  "query": {
    "match_phrase_prefix":
      {
      "author": "A"
      }
    }
}
```

```
{
  "took" : 24,
  "timed_out" : false,
  "_shards" : {
    "total" : 9,
    "successful" : 9,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 401,
    "max_score" : 14.054802,
    "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.

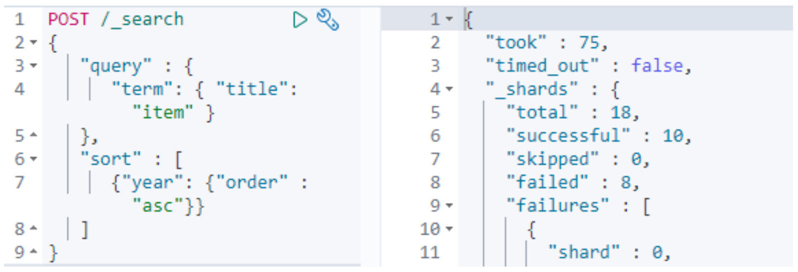
OpenSearch



```
POST /_search
{
  "query" : {
    "term": { "title": "item" }
  },
  "sort" : [
    {"year": {"order" : "asc"}}
  ]
}
```

```
{
  "took" : 39,
  "timed_out" : false,
  "_shards" : {
    "total" : 13,
    "successful" : 10,
    "skipped" : 0,
    "failed" : 3,
    "failures" : [
```

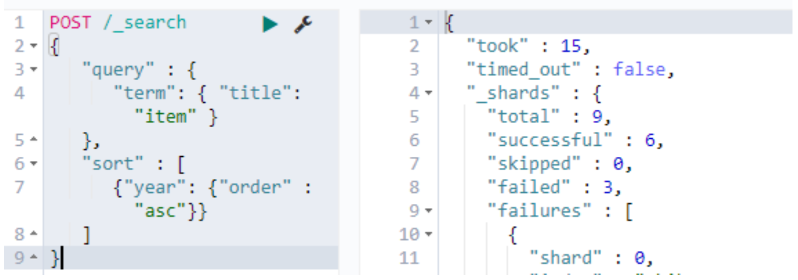
New Elasticsearch



```
POST /_search
{
  "query" : {
    "term": { "title":
      "item" }
  },
  "sort" : [
    {"year": {"order" :
      "asc"}}
  ]
}
```

```
{
  "took" : 75,
  "timed_out" : false,
  "_shards" : {
    "total" : 18,
    "successful" : 10,
    "skipped" : 0,
    "failed" : 8,
    "failures" : [
      {
        "shard" : 0,
```

Old Elasticsearch

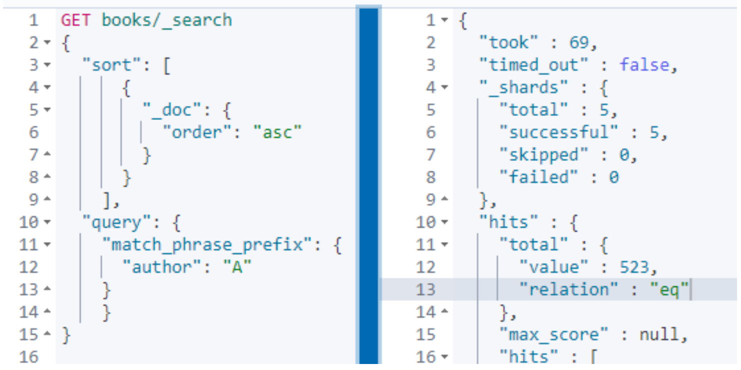


```
POST /_search
{
  "query" : {
    "term": { "title":
      "item" }
  },
  "sort" : [
    {"year": {"order" :
      "asc"}}
  ]
}
```

```
{
  "took" : 15,
  "timed_out" : false,
  "_shards" : {
    "total" : 9,
    "successful" : 6,
    "skipped" : 0,
    "failed" : 3,
    "failures" : [
      {
        "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 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.

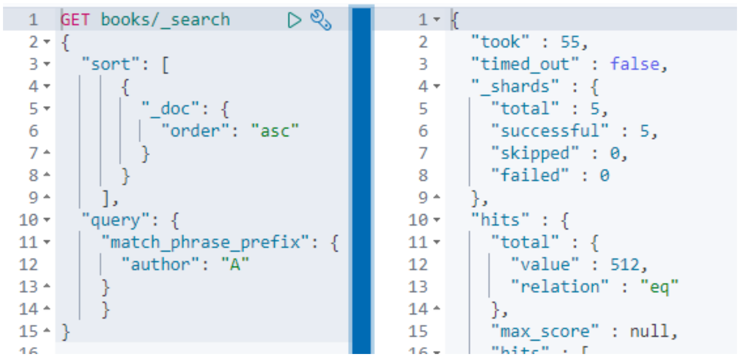
OpenSearch



```
GET books/_search
{
  "sort": [
    {
      "_doc": {
        "order": "asc"
      }
    }
  ],
  "query": {
    "match_phrase_prefix": {
      "author": "A"
    }
  }
}
```

```
{
  "took" : 69,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 523,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
```

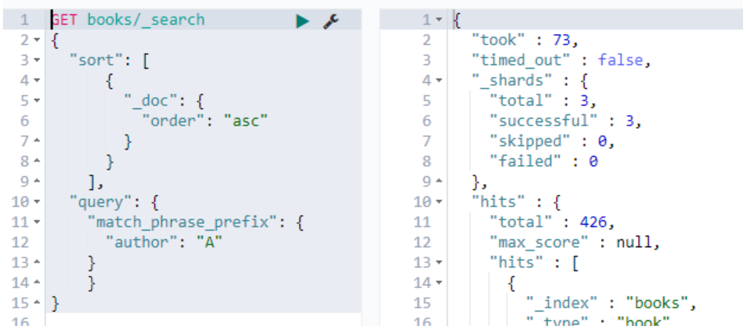
New Elasticsearch



```
GET books/_search
{
  "sort": [
    {
      "_doc": {
        "order": "asc"
      }
    }
  ],
  "query": {
    "match_phrase_prefix": {
      "author": "A"
    }
  }
}
```

```
{
  "took" : 55,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 512,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [
```

Old Elasticsearch

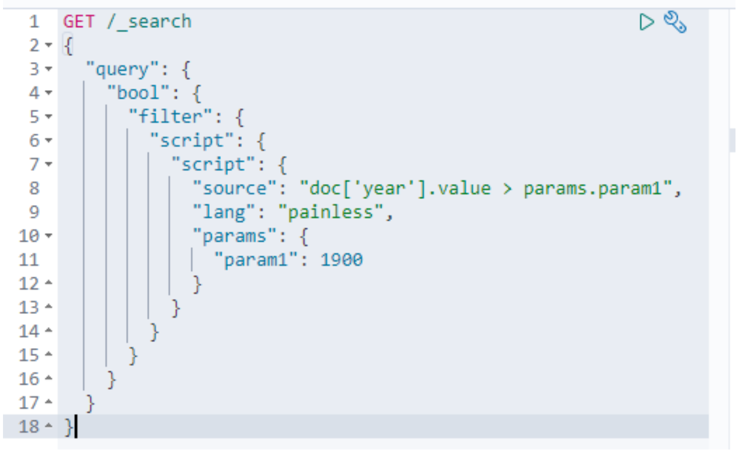


```
GET books/_search
{
  "sort": [
    {
      "_doc": {
        "order": "asc"
      }
    }
  ],
  "query": {
    "match_phrase_prefix": {
      "author": "A"
    }
  }
}
```

```
{
  "took" : 73,
  "timed_out" : false,
  "_shards" : {
    "total" : 3,
    "successful" : 3,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 426,
    "max_score" : null,
    "hits" : [
      {
        "_index" : "books",
        "_type" : "book"
```

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).

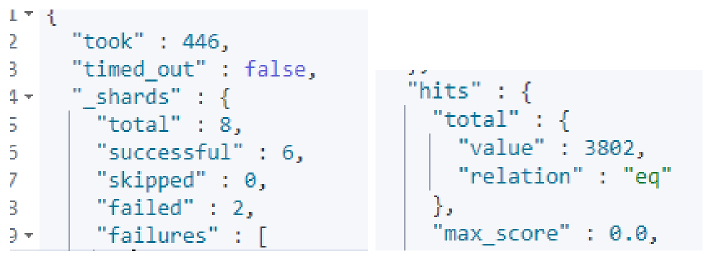
The script



```
GET /_search
{
  "query": {
    "bool": {
      "filter": {
        "script": {
          "script": {
            "source": "doc['year'].value > params.param1",
            "lang": "painless",
            "params": {
              "param1": 1900
            }
          }
        }
      }
    }
  }
}
```

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.

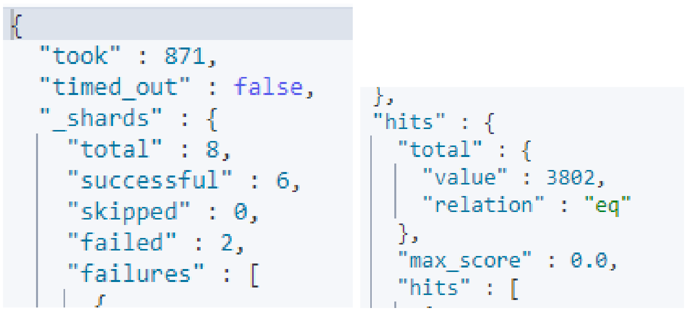
OpenSearch



```
{
  "took" : 446,
  "timed_out" : false,
  "_shards" : {
    "total" : 8,
    "successful" : 6,
    "skipped" : 0,
    "failed" : 2,
    "failures" : [
```

```
},
"hits" : {
  "total" : {
    "value" : 3802,
    "relation" : "eq"
  },
  "max_score" : 0.0,
```

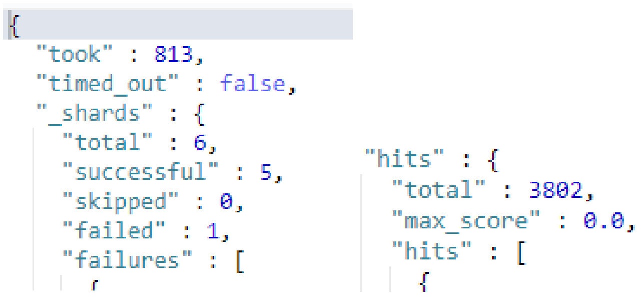
New Elasticsearch



```
{
  "took" : 871,
  "timed_out" : false,
  "_shards" : {
    "total" : 8,
    "successful" : 6,
    "skipped" : 0,
    "failed" : 2,
    "failures" : [
```

```
},
"hits" : {
  "total" : {
    "value" : 3802,
    "relation" : "eq"
  },
  "max_score" : 0.0,
  "hits" : [
```

Old Elasticsearch



```
{
  "took" : 813,
  "timed_out" : false,
  "_shards" : {
    "total" : 6,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 1,
    "failures" : [
```

```
"hits" : {
  "total" : 3802,
  "max_score" : 0.0,
  "hits" : [
    {
```

2.4. Conclusions

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.

3. Related work

Even though the differences may seem insignificant in such a small data set, when it comes to real life scenarios where petabytes of data need to be quickly accessed, that small distinction will make a difference. So looking into what kind of domain you should be choosing for your project is pretty important. But rather than taking their time comparing the older versions to the newer ones, most of the papers about Elasticsearch and OpenSearch are focused on the astonishing changes they brought in countless distinct fields.

In the paper [1] "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. [2] 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. [3] 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 messaged 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.

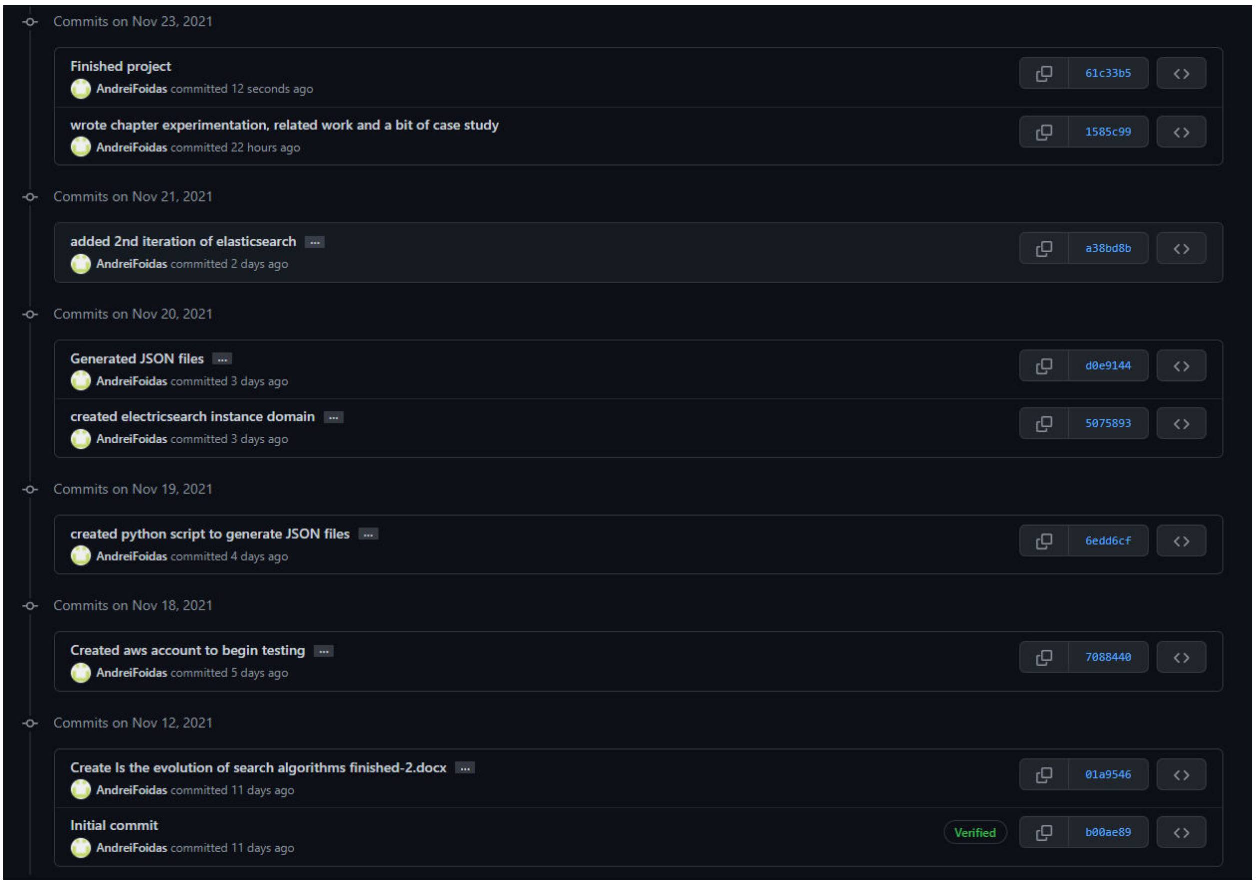
Bibliography

[1] 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)

[2] 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)

[3] 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 Nov 23, 2021

Finished project
AndreiFoidas committed 12 seconds ago
61c33b5

wrote chapter experimentation, related work and a bit of case study
AndreiFoidas committed 22 hours ago
1585c99

Commits on Nov 21, 2021

added 2nd iteration of elasticsearch ...
AndreiFoidas committed 2 days ago
a38bd8b

Commits on Nov 20, 2021

Generated JSON files ...
AndreiFoidas committed 3 days ago
d0e9144

created electricsearch instance domain ...
AndreiFoidas committed 3 days ago
5075893

Commits on Nov 19, 2021

created python script to generate JSON files ...
AndreiFoidas committed 4 days ago
6edd6cf

Commits on Nov 18, 2021

Created aws account to begin testing ...
AndreiFoidas committed 5 days ago
7088440

Commits on Nov 12, 2021

Create Is the evolution of search algorithms finished-2.docx ...
AndreiFoidas committed 11 days ago
01a9546

Initial commit
AndreiFoidas committed 11 days ago
Verified
b00ae89