Comparing Apache Solr and Elasticsearch search servers

Nikola Luburić, Dragan Ivanović
University of Novi Sad, Faculty of Technical Sciences, Novi Sad
{nikola.luburic, dragan.ivanovic}@uns.ac.rs

***Abstract* – The paper presents a comparative analysis of the leading two platforms for developing information retrieval systems, Apache Solr and Elasticsearch. We briefly examine other similar solutions, but focus on the previously mentioned solutions as they provide greater functionality with better performance. Our goal was to examine both systems, including what they offer and how they are used. We examine expert opinions on both systems, as well as concrete use cases. After that we make a comparative analysis focusing on many aspects, from usability to working at scale. Finally we conclude which system works better for which use case.**

INTRODUCTION

With the development of information communication technology more and more data is being written and stored digitally. While collecting data has become less of a problem, extracting useful information from massive volumes of digital documents has become a large issue. Relational databases, which were the previously leading solutions for data storage, aren't designed for such scale and big data searches. In order to solve this issue, a search engine needs to be built that efficiently stores, indexes and searches data, so that the end user can quickly access the information he\she needs.

Search engine indexing is a process in which data is collected, parsed and stored in order to support fast and accurate information retrieval [1]. Searching is a process which includes query processing and information retrieval based on the query. In order to be efficient, the process uses previously formed indexes, so as to avoid scanning every document in the corpus [2].

The previously mentioned functionality is essential for solving the problem of efficient information retrieval, but it is not sufficient in and of itself. The system needs to provide an easy to use, intuitive user interface, with which the user can utilize the search capabilities. The system needs to take into account not only the user's lack of knowledge about the underlying structure of the data set, but also his\her inability to form precise queries. Another issue, related to indexing and searching in general, is that the result set for a given query will be imprecise, regardless of the quality of the query or the implementation of the search system. The previously mentioned issues can be solved by introducing a ranking system, which will sort the results for a given query by relevance.

As information retrieval has become a serious problem, many solutions have arisen over the years trying to address it. In this large set of solutions it is hard to choose the right tool without previous experience.

This paper represents a comparative analysis of the leading systems which solve the problems mentioned above. The main motivation behind this paper is to provide developers and members of the scientific community an overview of the best solutions for developing information retrieval systems, as well as give insight for the best use cases of both solutions.

During our research we preformed manual inspection of various solutions, and in the end isolated the two systems Apache Solr [3] and Elasticsearch [4]. Solutions such as Sphinx [5] and Xapian [6], Whoosh [7], while still in active development, lack functionality compared to Solr and Elasticsearch, while solutions such as Swish-E [8] have stopped with development altogether.

In the following chapter we take a look at examples where Solr and Elasticsearch have been utilized. In chapter three we analyze both systems separately, while chapter four focuses on the actual comparison. This includes comparing the differences in design, offered functionalities, ease of use and resource consumption. Finally, we list the use cases for each system and make predictions about the future of these two systems.

RELATED WORK

We examined several solutions which used Apache Solr or Elasticsearch as their primary search engine. This includes solutions produced by the scientific community as well as several major organizations in the industry.

In [9] Atanassova and Bertin present an information retrieval system for scientific papers using Solr. Their approach provides a new way to access relevant information in scientific papers by utilizing semantic facets. Faceted search allows the user to visualize multiple categories and to filter the results according to these categories. In [10] Cuff and colleagues show a significant improvement in the search function of their CATH system when switching to Solr. CATH, which stands for class, architecture, topology and homology, is a hierarchical protein domain classification system.

Many organizations, such as Helprace, Jobreez, Apple, Inc., AT&T, AOL, reddit, etc. use Solr for search and faceted browsing [11]. Helprace uses Solr to power its search engine and search suggestions, while Jobreez uses Solr to search for jobs across 25 000 sources. AT&T uses Solr to run local searches on its yellowpages, while AOL utilizes Solr to power most of its channels.

In [12] Kononenko and colleagues describe the significant improvement in performance they got with their software analytics dashboard tool. This performance boost is solely due to switching from a traditional relational database to Elasticsearch. In [13] Thompson and

colleagues describe their use of Elasticsearch in querying graphical music documents. As far as organizations go, notable users include CERN [14], GitHub [15], Stack Exchange [16], Mozzila [17], etc. CERN uses Elasticsearch to efficiently manage and search through the various logs their devices produce. In 2013 GitHub started using an Elasticsearch cluster which indexes code as it gets pushed to the repository. This change marked an increase in search result relevancy and general search performance.

Lastly, there are organizations such as Foursquare which use both Solr and Elasticsearch in their infrastructure [18].

As both Solr and Elasticsearch are leading solutions for information retrieval this paper aims to compare the two systems. There are several articles written by experts in the industry which compare these systems [19-22], and this paper represents a synthesis of those articles, as well as our own experience.

Analyzed Systems

Before examining Solr and Elasticsearch we take a look at Apache Lucene [23] which is the underlying information retrieval library for both systems. Lucene is a free, open source and independent library which has been widely recognized for its utility in the implementation of Internet search engines and local searching. The primary function of this library is indexing and searching and Lucene result ranking uses a combination of the Vector Space model and the Boolean model of information retrieval [24] to determine how relevant a given document is to a user's query.

Apache Solr is an open source enterprise search server originally written in Java. It runs as a standalone full text search server, using Lucene for indexing and search functionalities. The system exposes a REST-like API and most of the interaction between the user and Solr is done over HTTP. By sending HTTP PUT and POST requests it is possible to send documents for storage and indexing, while the HTTP GET request allows the user to retrieve results based on queries. The data sent and retrieved supports several formats, including XML, JSON and CSV. Not only does Solr store, index and search data, it also offers additional features, including analytics of the indexed data. Unlike Lucene which offers indexing and searching, but lacks the needed infrastructure to be a standalone application, Solr is a web application which can be deployed on any servlet container. This allows Solr to be used as a tool by people from various professions, as was shown in the related works section.

Similarly to Solr, Elasticsearch is also an open source enterprise search server written in Java. It provides a distributed full text search engine, with a REST-like web interface and uses JSON for the document format. It provides scalable search, has near real-time search and supports multitenancy. A significant feature of this system is massive distribution and high availability. Elasticsearch allows users to start small and scale horizontally as they grow. These clusters are resilient and will detect new or failed nodes and reorganize data automatically, to ensure that it stays safe and accessible. More so than Solr, this system offers real time advanced analytics of the indexed data. Elasticsearch can be used as a standalone system by people of various professions.

It should be noted that both systems evolved together and learned from another, reaching a point where they are very similar to one another.

Comparative Analysis

Before diving into the comparative analysis it should be noted that both solutions in their current versions (Solr at 5.3.1 and Elasticsearch at 1.7.2 at the time the comparison took place) are similar systems, in the sense that they offer near-equal functionalities and have similar performance. It should also be noted that while some difference does exist, both systems can be suitable solutions for most common information retrieval needs. As both technologies are mature and stable and have a strong community behind them, most of the time the decision whether to use one solution over the other will come down to preference and the foreknowledge of the team.

The main difference between these solutions derives from their cores which significantly differ from one another. Looking at the distributions, Solr takes more space on the hard drive. This is primarily owed to the fact that the standard Solr distribution includes functionality, other than the base, which may or may not be useful to the end user, such as Map-Reduce, a testing framework, a web application which represents a GUI monitoring tool, etc. Unlike Solr, Elasticsearch's core consists of only the base code and documentation, which is why it needs one third of the space that Solr needs. Fig. 1 shows the composition of the web application archives of both systems.

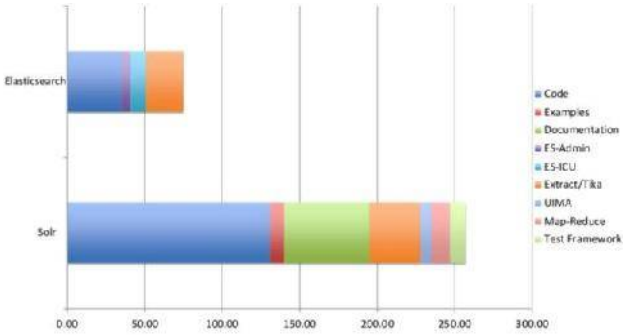

Figure 1. Solr and Elasticsearch .war composition[1]

Elasticsearch starts from the premise that the end user will always need the minimum functionality that Lucene offers and not much else. By configuring the system and using additional tools like Logstash, Kibana and Marvel, the user can expand the basic functionality to suit his needs. While Solr also offers a wide variety of plugins, its core includes modules which aren't always necessary. A problem that both systems have, but is more evident with Elasticsearch, is the lack of a centralized orchestration tool, for plugin and dependency management. If the user wants to create an Elasticsearch cluster he must manually install Elasticsearch and all the needed plugins on each node.

The second difference can be found in the cluster management subsystems of these two solutions. Solr relies on Apache ZooKeeper [25] which is a mature and tested technology, but more often than not offers little more than

[1] Source: http://www.slideshare.net/arafalov/solr-vs-elasticsearch-case-by-case

the much simpler built-in cluster management subsystem of Elasticsearch. ZooKeeper adds complexity to the system, as it is a separate component that needs to be managed. ZooKeeper also requires three nodes to form a cluster, while Elasticsearch can form a cluster with only one node.

Both Solr and Elasticsearch handle document preprocessing in a similar fashion. Both systems have configuration files in which analyzers, tokenizers and filters are declared. The declared components are used both during file and query preprocessing. Fig. 2 shows an example document containing the preprocessing configuration in Solr, while fig. 3 shows a similar configuration for Elasticsearch.

```
<fieldType name="nametext" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.KeepWordFilterFactory" words="keepwords.txt"/>
    <filter class="solr.SynonymFilterFactory" synonyms="syns.txt"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

Figure 2. Solr preprocessor configuration file

```
index :
    analysis :
        analyzer :
            standard :
                type : standard
                stopwords : [stop1, stop2]
            myAnalyzer1 :
                type : standard
                stopwords : [stop1, stop2, stop3]
                max_token_length : 500
            # configure a custom analyzer which is
            # exactly like the default standard analyzer
            myAnalyzer2 :
                tokenizer : standard
                filter : [standard, lowercase, stop]
        tokenizer :
            myTokenizer1 :
                type : standard
                max_token_length : 900
            myTokenizer2 :
                type : keyword
                buffer_size : 512
        filter :
            myTokenFilter1 :
                type : stop
                stopwords : [stop1, stop2, stop3, stop4]
            myTokenFilter2 :
                type : length
                min : 0
                max : 2000
```

Figure 3. Elasticsearch preprocessor configuration file

Apart from the built-in preprocessing components, both systems allow for creation and use of custom components. It is even possible to move the preprocessing to an entirely separate system, and this is where the two systems differ. While Solr recommends a tight coupling between preprocessing and indexing and searching, Elasticsearch takes a more modular approach, and recommends a separate system (e.g. Logstash) for preprocessing. This approach increases the system complexity by introducing another moving part, but avoids bottlenecks by allowing each subsystem to scale separately.

In the context of document preprocessing it is also worth noting how both systems handle language recognition. Solr has this functionality built-in, while Elasticsearch requires a plugin. Several good solutions exist, and coupled with such a library, Elasticsearch handles this issue as well as Solr.

Both Solr and Elasticsearch can index digital documents such as PDF, MS Word document, etc. Once again, this is part of Solr, while Elasticsearch uses an external module. Both systems rely on Apache Tika for this functionality [23].

Highlighting is another feature both systems handle well, offering a high degree of flexibility in the configuration of summary creation and management. Solr handles this using the `hl` object [24]. By accessing the object's fields (e.g. `hl.formatter`, `hl.snippets`, etc.) the highlighter can be configured. Elasticsearch highlighter configuration is done by sending a `highlight` object [25] with the query request. This object contains most of the fields that Solar's `hl` object has. Fig. 4 shows an example of such an object. As the image shows, it is possible to form a query for the highlighter, making highlighting independent from searching.

```
"highlight" : {
    "order" : "score",
    "fields" : {
        "content" : {
            "fragment_size" : 150,
            "number_of_fragments" : 3,
            "highlight_query": {
                "bool": {
                    "must": {
                        "match": {
                            "content": {
                                "query": "foo bar"
                            }
                        }
                    },
                    "should": {
                        "match_phrase": {
                            "content": {
                                "query": "foo bar",
                                "phrase_slop": 1,
                                "boost": 10.0
                            }
                        }
                    },
                    "minimum_should_match": 0
                }
            }
        }
    }
```

Figure 4. Highligh object in an Elasticsearch query request

Elasticsearch has a few advantages when compared to Solr. The main advantage this system has comes from its simplicity. Elasticsearch is easier to install, setup and use. The REST-like services which work with JSON are not only simple to use, but are more aligned with the current trends in the industry that Web 2.0 has brought. This does simplify things for software developers, but it should be

noted that other industries use these systems as well. The JSON-based query language that Elasticsearch uses is also arguably simpler than the HTTP requests that need to be formed in order to query Solr.

Another difference comes from the general direction that both systems are moving towards. While Solr remains specialized for document indexing and searching, the Elasticsearch team puts significant efforts into improving and expanding their data analytics subsystem. This might very well be the most significant difference between these two systems.

When analyzing performance we found that both systems showed similar results on datasets of medium size. Elasticsearch did, however, surpass Solr when testing analytic queries, which was expected.

Finally, another key difference can be seen in the metrics that both systems offer. While Solr does provide key metrics, Elasticsearch (in its core, as well as by utilizing plugins) offers significantly more metrics.

CONCLUSION

The paper examines two leading solutions for information retrieval, Apache Solr and Elasticsearch, and presents a side by side comparison of these two systems. After manually inspecting both systems and researching the papers and articles on the subject we conclude that both systems are good choices when it comes to document indexing and searching.

Over the years both solutions learned from one another and significant improvements in both systems are partially due to the competition. While Solr still seems to be the more common solution for classic enterprise systems, Elasticsearch's simplicity, flexible design and modular architecture make this system a great choice for both prototyping and large, scalable information retrieval solutions. Elasticsearch offers far better data analytics, and when combined with Logstash and Kibana, the ELK stack surpasses Solr in many areas, including preprocessing, analytics and visualization.

Even though both teams continue to upgrade and develop new features, the fact of the matter is that Elasticsearch has a fresh, compact core, created after the various drawbacks of Solr were noticed. Solr hasn't stood still and while many improvements were made it can be concluded that Elasticsearch will in time surpass this system. This coupled with the fact that Elasticsearch relies on one man to approve or decline changes to the system, while Solr requires that every new features goes through a more rigid protocol of evaluation means that Elasticsearch, as it stands, will have quicker and more frequent updates.

The only real downside of Elasticsearch in its current version is that it lacks a centralized tool for managing the nodes of a cluster. This can easily lead to misconfiguration in clusters with many nodes, as a separate installation of the core Elasticsearch instance and all of its plugins is needed every time a node is added to the cluster. Without version control, installing updates for parts of the system present another problem.

While Elasticsearch does seem to be the go to solution for use cases where serious analytics are needed this does not mean Solr should be abandoned. The ELK stack might be a slightly more suitable solution, but reworking a system which already utilizes Solr will more often than not be pointless. Likewise, teams who have experience with Solr shouldn't switch to a new system without serious consideration, as both systems are near equal in most cases.

ACKNOWLEDGMENT

Results presented in this paper are part of the research conducted within the Grant No. III-47003, Ministry of Education, Science and Technological Development of the Republic of Serbia.

REFERENCES

[1] Z. G. Gonzalez, M. Kelly, T. E. Murphy Jr, and M. Nisenson, International Business Machine Corporation, 2012. *Search Engine Indexing*. U.S. Patent Application 13/713,765.

[2] S. T. Kirsch, W. I. Chang, and E. R. Miller, Infoseek Corporation, 1999. *Real-time document collection search engine with phrase indexing*. U.S. Patent 5,920,854.

[3] Apache Software Foundation, *Solr*, http://lucene.apache.org/solr/, retrieved: 20.10.2015.

[4] S. Banon, *Elasticsearch*, https://www.elastic.co/, retrieved: 20.10.2015.

[5] A. Aksyonoff, *Sphinx*, http://sphinxsearch.com/, retrieved: 20.10.2015.

[6] Xapian Team, *Xapian*, http://xapian.org, retrieved: 20.10.2015.

[7] M. Chaput, *Whoosh*, https://bitbucket.org/mchaput/whoosh/wiki, retrieved: 20.10.2016.

[8] K. Hughes, *Swish-E*, http://swish-e.org/, retrieved: 20.10.2015.

[9] I. Atanassova, and M. Bertin, Faceted Semantic Search for Scientific Papers. *PLoS Biology*, *2*, pp.426-522.

[10] A. L. Cuff, I. Sillitoe, T. Lewis, A. B. Clegg, R. Rentzsch, N. Furnham M. Pellegrini-Calace, D. Jones, J. Thornton, and C. A. Orengo, 2011. Extending CATH: increasing coverage of the protein structure universe and linking structure with function. *Nucleic acids research*, *39*(suppl 1), pp.D420-D426.

[11] Apache Wiki, *PublicServers*, https://wiki.apache.org/solr /PublicServers, retrieved: 20.10.2015.

[12] O. Kononenko, O. Baysal, R. Holmes, and M.W. Godfrey, 2014, May. Mining modern repositories with elasticsearch. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp.328-331.

[13] J. Thompson, A. Hankinson, and I. Fujinaga, 2011. Searching the Liber Usualis: Using COUCHDB and ELASTICSEARCH to query graphical music documents. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*.

[14] G. Horanyi, *Needle in a haystack*, https://medium.com/@ghoranyi /needle-in-a-haystack-873c97a99983#.autnzq6bt, retrieved: 20.10.2015.

[15] T. Pease, *A Whole New Code Search*, https://github.com/blog /1381-a-whole-new-code-search, retrieved: 20.10.2015.

[16] N. Carver, *What it takes to run Stack Overflow*, http://nickcraver.com/blog/2013/11/22/what-it-takes-to-run-stack-overflow/, retrieved: 20.10.2015.

[17] P. Alves, *Firefox 4, Twitter and NoSQL Elasticsearch*, http://pedroalves-bi.blogspot.rs/2011/03/firefox-4-twitter-and-nosql.html, retrieved: 20.10.2015.

[18] H. Karau, A. Alix, *foursquares now uses Elasticsearch*, http://engineering.foursquare.com/2012/08/09/foursquare-now-uses-elastic-search-and-on-a-related-note-slashem-also-works-with-elastic-search/, retrieved: 20.10.2015.

[19] K. Tan, *Apache Solr vs Elasticsearch - The Feature Smackdown*, http://solr-vs-elasticsearch.com/, retrieved: 20.10.2015.

[20] O. Gospodnetić, *Solr or Elasticsearch – that is the question*, http://www.datanami.com/2015/01/22/solr-elasticsearch-question/, retrieved: 20.10.2015.

[21] R. Sonnek, *Realtime Search: Solr vs Elasticsearch*, http://blog.socialcast.com/realtime-search-solr-vs-elasticsearch/, retrieved: 20.10.2015.

[22] A. Rafalovitch, *Solr vs. Elasticsearch – Case by Case*, http://www.slideshare.net/arafalov/solr-vs-elasticsearch-case-by-case, retrieved: 20.10.2015.

[23] Apache Software Foundation, *Lucene*, https://lucene.apache.org, retrieved: 20.10.2015.

[24] C. D. Manning, P. Raghavan, and H. Schütze, 2008. *Introduction to information retrieval*, Cambridge: Cambridge university press.

[25] Apache Software Foundation, *Apache ZooKeeper*, https://zookeeper.apache.org/, retrieved: 20.10.2015.

[26] Apache Software Foundation, *Apache Tika*, https://tika.apache.org/, retrieved: 20.10.2015.

[27] *Solr HighlightingParameters*, https://wiki.apache.org/solr/HighlightingParameters, retrieved: 20.10.2015.

[28] *Elasticsearch Highlighting*, https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-highlighting.html, retrieved: 20.10.2015.