ielab at the Open-Source IR Replicability Challenge 2019

Harrisen Scells
The University of Queensland
Brisbane, Australia
h.scells@uq.net.au

Guido Zuccon
The University of Queensland
Brisbane, Australia
g.zuccon@uq.edu.au

ABSTRACT

We present our participating Docker image in the SIGIR Open-Source IR Replicability Challenge 2019. Our participation includes an Elasticsearch-based Docker image conforming to the specifications laid out by the challenge, a collection parser, and a topic parser. Our Docker image is capable of indexing and searching several commonly used Information Retrieval test collections including robust04 (Trec 2004 Robust Track Topics – disks 4/5), core17 (TREC New York Times Annotated Corpus), and core18 (TREC Washington Post Corpus). The Docker image and associated tools we present in this paper provide a solid foundation for replicable Information Retrieval experiments using Elasticsearch.

KEYWORDS

Systematic Reviews, Query Formulation, Boolean Queries

1 INTRODUCTION

The main contribution of our participation in the Open-Source IR Replicability Challenge 2019 (OSIRRC 2019) is an Elasticsearch-based Docker image. Our Docker image provides an environment for performing replicable Information Retrieval (IR) experiments using Elasticsearch. However, Elasticsearch is not designed to be used in academic or research settings, e.g., by default, it cannot output TREC run files, or issue TREC topics as queries; it is difficult to configure fine-grain aspects of retrieval models, and it is difficult to extend in terms of additional retrieval models. These are all challenges which have been identified in initiatives such as the Elastic4ir[1] presentation as part of the SIGIR LIARR 2017 Workshop [2] and the IR In Practice session at AFIRM 2019[2]. Despite these limitations, we were able to package Elasticsearch conforming to the majority of the specifications in this challenge.

The contributed Docker image is capable of indexing and searching three test collections: robust04 — Trec 2004 Robust Track Topics — disks 4/5, core17 — TREC New York Times Annotated Corpus, and core18 — TREC Washington Post Corpus. We also make two other contributions which support indexing and searching using Elasticsearch. One of the main selling points of Elasticsearch is the ability to be schema-less (i.e., automatic mapping of document fields and data types to the underlying Lucene index). As a consequence of this, documents in Elasticsearch are represented as (and indexed as) JSON objects. This presents a challenge for indexing standard IR test collections, as they commonly use a variety of file formats such as XML, JSON, WARC or plain text – or even purpose-based formats, e.g. the TREC document format. Due to this, the second contribution of our participation in OSIRRC 2019 is a tool for processing IR test collection file formats into Elasticsearch bulk indexing operations. Finally, because Elasticsearch is a commercial, off-the-shelf search engine, not built for research purposes, it does not directly support issuing TREC topics as queries. Due to this, the third contribution of our participation in OSIRRC 2019 is a tool for directly issuing TREC topic files to Elasticsearch. These two tools are described in more detail in the following sections.

The OSIRRC 2019 uses a *jig* to issue commands to *hooks* within participating Docker images. We implement the main hooks of the jig, described in Section 2. We also report the arguments we pass to the jig in order for it to use the implemented hooks in Section 3. Finally, we report baseline results from indexing and searching the three collections compatible with our Docker image in Section 4.

Like other participants in the challenge, our Dockerfile, associated tools, and documentation is made available on GitHub at https://github.com/osirrc/ielab-docker.

2 HOOK IMPLEMENTATIONS

The OSIRRC 2019 uses a jig[3] to communicate with the Docker image from participating teams. The jig supports five different hooks that a participating Docker image can implement. Our Docker image supports the `init`, `index`, and `search` hooks. Our implementation of these hooks are described in the following subsections. We also examine the unimplemented hooks and how these would work if implemented in future work.

2.1 `init`

The `init` hook is used to prepare a participating Docker image prior to indexing. We found, however that using this hook significantly slowed development time, and instead decided to include all of our setup steps in our Dockerfile. When using a Dockerfile, listed commands can be cached between image builds. When compiling code, downloading external artefacts, and copying large files, these processes take time away from development when in the `init` hook.

Our Dockerfile uses a multi-stage build to reduce the total size of the image, counteracting what would be a downside to using it. In the first stage, development libraries are installed, Elasticsearch is downloaded and decompressed (our participating Docker image uses Elasticsearch 7.0.0, which was the latest version at the time development of the image began), and our command-line applications for parsing and searching are compiled. In the following stage, only the necessary runtime applications are installed (e.g., Python), and the artefacts from the previous stage are copied.

[1] https://github.com/ielab/elastic4IR
[2] http://ielab.io/tutorials/ir-in-practice.html
[3] https://github.com/osirrc/jig

2.2 index

The index hook is used to index one or more collections. There are several steps before a collection can be indexed in this phase. Firstly, depending on the collection, files must be decompressed or removed (e.g., removing README files, changelogs, tools). Next, Elasticsearch must be started. Because Elasticsearch is designed to be run as a service, it must be started in the background. Our index hook then waits until the Elasticsearch web service is reachable (by repeatedly making HTTP requests), and then waits until the cluster health level of Elasticsearch is at least *yellow* (i.e., all shards have been allocated). API requests to Elasticsearch will fail if the web service is available but the cluster health is *red* (i.e., there are shards still waiting to be allocated or there is a problem with Elasticsearch). This was a major pain point during development of our system and lead to many unexpected issues.

At this stage, Elasticsearch is available for indexing documents. Documents in the collection path (provided by the jig) are transformed by our TREC collection parser [4] and indexed in bulk. When indexing different collections, we do not attempt any document cleaning or pre-processing (e.g., stemming). We generally index titles into one field, then the content of the document from all other fields (e.g., paragraphs, sub-headings) under the content part of the document (e.g., TEXT/body/contents, etc.) into a single text field. Examples of how collection documents are transformed into a representation suitable to be indexed in Elasticsearch are described in detail in the following sections.

2.2.1 robust04. Documents in this collection are in a XML-like format (in fact we parse them using a generic XML parser). One aspect of this file format that caused the XML parser to fail are the unquoted attributes on elements (e.g., <F P=100>, the 100 should be wrapped in quotation marks to be valid XML). To overcome this, we simply removed all attributes from every element. When transforming the files in the robust04 collection, we index everything in the <TEXT> tag as a single string, ignoring structure. An example of this transformation is presented in Figure 1.

2.2.2 core17. The documents for core17 are in a standardised XML format. We take a similar approach to indexing these documents to the robust04 documents. Unlike the robust04 documents, there is a clear title in the header of the core17 documents which we extract. We then proceed to index everything between the <body> tags as a single string, ignoring structure. An example core17 document and what we transform it into as a JSON representation suitable for indexing in Elasticsearch is presented in Figure 2.

2.2.3 core18. The documents in the core18 collection are already in a format suitable for Elasticsearch to index (i.e., they come as a JSON object). One challenge we faced here, however is that the main content of these documents is stored as an array of individual JSON objects, all with the same field name (content), but with different data-types for the values. This issue is illustrated in Figure 3. This is a problem because Elasticsearch infers the types of values to create a mapping based on the first document indexed. When Elasticsearch encounters a value that does not conform to the data-type specified by the mapping, it will fail to index the document. To overcome this issue, we simply cast all values to a string for this field before indexing.

```
<DOC>
<DOCNO> FBIS3-61525 </DOCNO>
<HT>      "jpust004___94105" </HT>


<HEADER>
<AU>   JPRS-UST-94-004 </AU>
Document Type:JPRS
Document Title:Science &amp;amp; Technology

</HEADER>

<ABS>  Central Eurasia </ABS>
<DATE1>   26 January 1994 </DATE1>
<F P=100> LIFE SCIENCES </F>
<F P=101> MEDICINE AND PUBLIC HEALTH </F>
<H3> <TI>   Environmental Monitoring System of Natural Water
    Conditions </TI></H3>
<F P=102>  947C0156B Kiev GIDROBIOLOGICHESKIY ZHURNAL in Russian
Vol 29 No 3, May-Jun 93 pp 88-95 </F>

<F P=103> 947C0156B </F>
<F P=104>  Kiev GIDROBIOLOGICHESKIY ZHURNAL </F>


<TEXT>
Language: <F P=105> Russian </F>
Article Type:CSO

<F P=106> [Article by S.V. Kreneva, Azov Scientific
    Research Institute </F>
of Fishery Industry, Rostov na Donu; UDC [593.17:574.63](08)(28)]
[Abstract] The widening gap between the rates at which new
contaminants appear the maximum...

</TEXT>

</DOC>
```

(a) Contents of robust04 document FBIS3-61525.

```
{
    "DocNo": "FBIS3-61525",
    "Text": "Language: Russian Article Type:CSO [Article
             by S.V. Kreneva, Azov Scientific Research
             Institute of Fishery Industry, Rostov na Donu;
             UDC [593.17:574.63](08)(28)]   [Abstract] The
             widening gap between the rates at which new
             contaminants appear the maximum..."
}
```

(b) Elasticsearch representation of robust04 document FBIS3-61525.

Figure 1: Example transformation of a document in robust04 into a representation suitable to be indexed in Elasticsearch.

2.3 search

The search hook is used to perform ad-hoc searches on a previously indexed collection. This hook uses TREC topics as queries, and as such, these topic files for a collection must be parsed. Our topic parsing and searching tool [5] reads a topic file, issues the title of the topic as an Elasticsearch "query string" query, and parses the Elasticsearch API response into a TREC run file to be evaluated automatically by the jig.

[4] https://github.com/osirrc/ielab-docker/tree/master/cparser

[5] https://github.com/osirrc/ielab-docker/tree/master/tsearcher

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nitf SYSTEM "http://www.nitf.org/IPTC/NITF/3.3...">
<nitf change.date="June 10, 2005" change.time="19:30">
<head>
<title>ART: DAVID SALLE'S WORKS SHOWN AT THE WHITNEY</title>
<meta content="23" name="publication_day_of_month"/>
<meta content="1" name="publication_month"/>
<meta content="1987" name="publication_year"/>
<meta content="Friday" name="publication_day_of_week"/>
<meta content="Weekend Desk" name="dsk"/>
<meta content="20" name="print_page_number"/>
<meta content="C" name="print_section"/>
<meta content="3" name="print_column"/>
<meta content="Arts" name="online_sections"/>
<docdata>
<doc-id id-string="6135"/>
<doc.copyright holder="The New York Times" year="1987"/>
<identified-content>
<classifier class="indexing_service">ART</classifier>
<classifier class="indexing_service">REVIEWS</classifier>
<classifier class="indexing_service">ART SHOWS</classifier>
<org class="indexing_service">WHITNEY MUSEUM OF AMERICAN...</org>
<person class="indexing_service">SMITH, ROBERTA</person>
<person class="indexing_service">SALLE, DAVID</person>
<classifier class="online_producer">Review</classifier>
<classifier class="online_producer">Top/Features/Arts</classifier>
</identified-content>
</docdata>
</head>
<body>
<body.head>
<hedline>
<hl1>ART: DAVID SALLE'S WORKS SHOWN AT THE WHITNEY</hl1>
</hedline>
<byline class="print_byline">By ROBERTA SMITH</byline>
<byline class="normalized_byline">SMITH, ROBERTA</byline>
</body.head>
<body.content>
<block class="lead_paragraph">
<p>LEAD: THE exhibition of David Salle's work that has just ...</p>
</block>
<block class="full_text">
<p>For American art of the 1980's, Salle's painting stands...</p>
</block>
</body.content>
</body>
</nitf>
```

(a) Contents of core17 document 6135.

```
{
    "DocNo": "6135",
    "Title": "ART: DAVID SALLE'S WORKS SHOWN AT THE WHITNEY",
    "Text": "ART: DAVID SALLE'S WORKS SHOWN AT THE WHITNEY By
            ROBERTA SMITH SMITH, ROBERTA LEAD: THE exhibition
            of David Salle's work that has just...For American
            art of the 1980's, Salle's painting stands..."
}
```

(b) Elasticsearch representation of core17 document 6135.

Figure 2: Example transformation of a document in core17 into a representation suitable to be indexed in Elasticsearch.

2.4 Unimplemented Hooks

The two unimplemented hooks are `train` and `interact`. As future work, if the `train` hook were to be implemented, we would use it to optimise retrieval function parameters (e.g., the $K1$ and B parameters of BM25, similar to work by Jimmy et al. [4]). Furthermore, if the `interact` hook were to be implemented, we would use it to expose the REST API of Elasticsearch. This would allow one to connect other tools such as Kibana, and explore the data further.

```
{
    "content": "Homicides remain steady in the Washington region",
    "mime": "text/plain",
    "type": "title"
}
```

(a) Data-type of `content` is a string.

```
{
    "content": 1483230909000,
    "mime": "text/plain",
    "type": "date"
}
```

(b) Data-type of `content` is an integer.

Figure 3: Example of JSON objects in the content section in a core18 document which contain different data-types for the same field (`content`).

```
python3 run.py prepare \
  --repo osirrc2019/ielab \
  --collections robust04=/path/to/disk45=trectext
```

(a) robust04

```
python3 run.py prepare \
  --repo osirrc2019/ielab \
  --collections core17=/path/to/NYTcorpus=nyt
```

(b) core17

```
python3 run.py prepare \
  --repo osirrc2019/ielab \
  --collections core18=/path/to/WashingtonPost.v2=wp
```

(c) core18

Figure 4: Prepare commands issued to the jig used to index each collection in our Elasticsearch Docker image.

We are also aware of optional parameters in the jig that can be used to pass configuration items into the Docker image. If we were to exploit these optional parameters in future work, we would use them to specify aspects of Elasticsearch such as sharding settings and document pre-processing using the `index` hook, or the fields to consider when scoring documents or the retrieval model using the `search` hook.

3 JIG ARGUMENTS

Here we provide the arguments passed to the jig in order to index and search the three collections compatible with our Docker image. The following two sections describe the two jig commands and the arguments we pass to them to obtain our baseline results.

3.1 `prepare`

The `prepare` jig command prepares a Docker image for performing experiments. This command calls the `init` and `index` hooks of the Docker image. Figure 4 presents the arguments passed to the jig in order to prepare our Elasticsearch Docker image to run experiments.

```
python run.py search \
  --repo osirrc2019/ielab \
  --output out/ielab \
  --qrels qrels/qrels.robust04.txt \
  --topic topics/topics.robust04.txt \
  --collection robust04
```

(a) robust04

```
python run.py search \
  --repo osirrc2019/ielab \
  --output out/ielab \
  --qrels qrels/qrels.core17.txt \
  --topic topics/topics.core17.txt \
  --collection core17
```

(b) core17

```
python run.py search \
  --repo osirrc2019/ielab \
  --output out/ielab \
  --qrels qrels/qrels.core18.txt \
  --topic topics/topics.core18.txt \
  --collection core18
```

(c) core18

Figure 5: Search commands issued to the jig used to search each collection in our Elasticsearch Docker image.

Run	MAP	P@30	nDCG@20
robust04 (BM25, k=1000)	0.1826	0.3236	0.3477
core17 (BM25, k=1000)	0.0831	0.2333	0.1779
core18 (BM25, k=1000)	0.1899	0.2760	0.3081

Table 1: Results of runs for the robust04, core17, and core18 collections using our Elasticsearch Docker image. Each run uses the default Elasticsearch BM25 scorer for ranking, and each run retrieves a maximum of 1,000 documents (k).

3.2 search

The `search` command issues queries from TREC topic files and outputs a TREC run file which is evaluated by the jig. This command calls the `search` hook of the Docker image. Figure 5 presents the arguments passed to the jig in order to run the search experiments on the collections compatible with our Elasticsearch Docker image.

4 RESULTS

Table 1 presents the evaluation results for each collection indexed using our Elasticsearch Docker image. The evaluation measures reported in this table are chosen based on the measures reported by the jig. Although all participants report the same evaluation measures it is important to note that one cannot fairly compare between systems and these results can only be used to demonstrate the replicability of our system. We elaborate on this detail in the following section, and present a possible solution.

5 DISCUSSION & CONCLUSIONS

Our contribution at the OSIRRC 2019 is an Elasticsearch Docker image capable of indexing and searching the robust04, core17, and core18 collections. While we had many difficulties getting Elasticsearch to work in this setup, and correctly parsing and indexing documents, we were able to implement the main hooks for the jig.

The jig of the OSIRRC 2019 introduced a standardised framework for anyone to implement hooks to a search system, to allow for replicability. However, due to the lack of standardisation of the fine-grain parametrisation of the systems, it becomes difficult to control systems comparisons. What the jig introduced is a way to *perform* experiments, however, it lacked a standard way to *configure* experiments. Many participants have exploited the optional arguments for indexing and searching, however the problem now is that each participant uses different syntax and naming conventions for standard parameters (e.g., which retrieval model to use, whether to perform stemming or not, etc.). We suggest that if this challenge were to be run again, another hook be added which defines the *capabilities* of a Docker image, which can include official, standard capabilities, such as the name and parameters of the retrieval model, what document pre-processing steps to take, or what fields to index (which generally apply to all search systems), as well as self-defined capabilities such as the compression algorithm to use (which may only apply to a subset of systems). Currently, while it is limiting to directly compare the evaluation results of our image to the images of other participants due to the reasons listed above, it becomes possible for others working with Elasticsearch to easily compare their system to this baseline. The current Docker images developed as part of this initiative only allow runs to be *replicable* and not *comparable*. By listing the capabilities of systems, it would allow one to fairly compare between two or more systems that share capabilities. Although system comparison was not the goal of this challenge, it could easily be facilitated by adopting solutions similar to that described above.

The OSIRRC 2019 initiative was valuable from both the perspective of implementing the jig hooks, and from comparing how others chose to implement them in their hooks. Despite the fact that Elasticsearch is primarily a commercial/production product, and not a research oriented tool, there are several published academic research papers that use it, e.g., those from our ielab research team[6] [4–6], and other groups [1, 3, 7]. This may be in part due to the ease with which one can index and search documents in Elasticsearch (very little to no configuration is necessary, provided an adequate parser is available). We hope that our contribution to and participation in this challenge leads to further replicability and reproducibility of Elasticsearch and other search systems, and that others can more easily compare their own Elasticsearch experiments to this common baseline.

REFERENCES

[1] Giuseppe Amato, Paolo Bolettieri, Fabio Carrara, Fabrizio Falchi, and Claudio Gennaro. 2018. Large-Scale Image Retrieval with Elasticsearch. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. ACM, New York, NY, USA, 925–928.

[2] Leif Azzopardi, Matt Crane, Hui Fang, Grant Ingersoll, Jimmy Lin, Yashar Moshfeghi, Harrisen Scells, Peilin Yang, and Guido Zuccon. 2017. The Lucene for information access and retrieval research (LIARR) workshop at SIGIR 2017. (2017).

[3] Rafael Glater, Rodrygo L.T. Santos, and Nivio Ziviani. 2017. Intent-Aware Semantic Query Annotation. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '17)*. ACM, New York, NY, USA, 485–494.

[4] Jimmy, Guido Zuccon, and Bevan Koopman. 2016. Boosting titles does not generally improve retrieval effectiveness. In *Proceedings of the 21st Australasian document computing symposium*. 25–32.

[6] http://ielab.io/

[5] Daniel Locke, Guido Zuccon, and Harrisen Scells. 2017. Automatic Query Generation from Legal Texts for Case Law Retrieval. In *Asia Information Retrieval Symposium*. Springer, 181–193.
[6] Harrisen Scells, Guido Zuccon, Bevan Koopman, Anthony Deacon, Leif Azzopardi, and Shlomo Geva. 2017. Integrating the framing of clinical questions via PICO| into the retrieval of medical literature for systematic reviews. In *Proceedings of the 26th ACM International Conference on Information and Knowledge Management*. 2291–2294.
[7] Luca Soldaini, Arman Cohan, Andrew Yates, Nazli Goharian, and Ophir Frieder. 2015. Retrieving medical literature for clinical decision support. In *European conference on information retrieval*. Springer, 538–549.