A SOLR/Lucene based Multi Tier Annotation Search solution

Matthijs Brouwer
Meertens Institute, The Netherlands
Matthijs.brouwer@meertens.knaw.nl

Marc Kemps-Snijders
Meertens Institute, The Netherlands
marc.kemps.snijders@meertens.knaw.nl

Abstract

In recent years, multiple solutions have become available providing search on huge amounts of plain text and metadata. Scalable searchability on annotated text however still appears to be problematic. We add annotational layers and structure to the existing Lucene approach of creating and searching indexes, and furthermore present an implementation as Solr plugin providing both searchability and scalability. SOLR/Lucene is fast, scales well, and has a large basis of users as well as developers. The latter stands in sharp contrast to several existing corpus search and management systems, for which one or few developers have the task of maintenance and further development of the system. With SOLR/Lucene one almost gets this for free.

1 Introduction

Many solutions providing search on both plain text and metadata rely on the reverse index based Apache Lucene[1]. The existing and popular SOLR extension offers additional features such as distributed indexing, scalability, load balanced querying and more to the construction of a sustainable and scalable infrastructure for these types of search requirements. However, for annotated textual resources these solutions appear less suitable due to the additional complexity introduced by the various annotation layers and limited options available within Solr/Lucene. Also, results and derived statistics are mainly based on numbers of documents, while often individual hits are required. There seems to be an increasing demand for solutions to these problems.

Several approaches are also available, amongst other from the CLARIN community, eg. BlackLab, Korap, SketchEngine, Corpus Workbench, PaQu, GreTeL to name a few. Several considerations have led us to develop a new initiative in this area, most notably scalability, integrated metadata/annotation search and the ability to control the (future) development process.

2 Requirements for annotated resources

While the Nederlab project[2] [Brouwer2013] served as one of the primary use cases for the system design it's development is firmly situated in the CLARIN domain as part of the Dutch CLARIAH[3] project. One of the major requirements thus is the ability to include arbitrary (CMDI) metadata schemas in search processes. While considerable experience is at hand in making metadata available in metadata search processes, such as the CLARIN VLO, the search domain needs to be extended to include annotated texts, containing multiple, often interdependent, annotation layers. These consist of normalized and spell-corrected texts, translations, lemma, part of speech(including feature lists), named entities, entity links to external knowledge bases (e.g. DBpedia), chapters, paragraphs, sentences and other hierarchical annotations such as morphology or syntactic information. Given the myriad of annotation formats encountered in the domain the system should be configurable to cope with a fair amount of different annotation formats.

At the individual annotation level support must be provided for multivalued attributes, differentiation between multiple set values (e.g. to cater for simultaneous multiple tag sets in source

[1] https://lucene.apache.org

[2] https://www.nederlab.nl
[3] http://www.clariah.nl/

documents) and full Unicode support at the value level. The system should support CQL (Corpus Query Language), possibly extended with additional features for higher order structures.With respect to result delivery, both documents and keyword-in-context representations must be delivered, including statistical information regarding absolute and relative frequencies and hit distributions across result sets. Result set distributions must be calculated across mulitple metadata dimensions, including time intervals, while multiple metadata dimensions may be operative at the same time, e.g. distribution across time and genre. Result sets may also be grouped according to result characteristics, such as grouping of all adjectives preceding a noun, to assist in determining collocations. Also, the system should be able to produce frequency lists across any type of annotation; word forms, part of speech, named entities, etc. Finally, the system should be highly scalable, be able to work acros multi-billion word corpora, be easily manageable and be freely available for use to a wide user community under an open source license.

2.1 Current solutions

A choice between search engines is often a balancing act between ones requirements and depends upon ones scope, corpus size, available expertise or conditions of use. Several systems directed towards searching annotated text structures are currently available, each with its own strengths, weaknesses and track record, e.g. BlackLab[Reynaert2014], Korap[Banski2013], Corpus Workbench[Evert2011], Sketch Engine[Kilgarriff2004], PaQu[Odijk2015], GrETEL[Vandeghinste2014]. While many of these provide partial coverage of the listed requirements, none of these provide a balanced coverage of all requirements, as stated above, to be immediately applicable to the projects at hand. It thus became clear that if any of these existing solutions were to be used they would need to be modified to suit our needs. Other alternatives, such as using graph databases (e.g. Neo4J), showed that performance and scalability is still problematic for many types of expected queries.
In our attempt to extend the BlackLab functionality to take advantage of the advanced scalability, sharding and other options provided by SOLR, such as facetting, it became clear that the underlying architecture of the system prevented us from doing so without significantly altering the underlying code base. Rather than modifying the complete code base we choose to reimplement the system in such a manner that it was interoperable with SOLR from the start. SOLR/Lucene is fast, scales well, and has a large basis of users as well as developers. The latter stands in sharp contrast to several existing corpus search and management systems, for which one or few developers have the task of maintainance and further development if the system. With SOLR/Lucene one almost gets this for free thus becoming an interesting option as an implementation basis. Also, we had already gained considerable experience using SOLR/Lucene for metadata and plain text indexing, it ties in well with existing infrastructure components and it provides good options for scalability and large corpus maintenance through its sharding functionality.

3 Extending Lucene and SOLR

Basic Lucene search functionality is based on the idea that data is grouped into documents. Each document consists of several fields and each field can have multiple values. Using this approach for metadata purposes this allows for several values of *genre*, e.g. *fictie* and *proza* to be associated with each document. To the existing Lucene approach we add annotations and structure by using *prefixes* to distinguish between text and different annotations. This provides a direct solution to store and search for annotations on individual words within a text, and only an adjusted tokenizer is needed to offer the correct tokenstream to the indexer. Ranges of words, distinct sets of words(e.g. named entities) and hierarchical relations are stored as a *payload*. Several additional extensions are used implementing different query strategies, that mostly extend default Lucene methods.

We take direct advantage of this set up in one of our projects where three data sets were indexed with part of speech encodings from three different tag sets. Rather than using a runtime query expansion mechanism we decided to use one of the tag sets as a pivot, mapped all other tag sets onto the pivot and indexed both the original tag and pivot tag sets values in our index. Each word is thus annotated with multiple pos of speech tags and in some cases even multiple part of speech tags from the same set (e.g. V $\rightarrow$ V-fin or V-infin).

Lucene uses an **inverted index**, storing the mapping from content, such as a word, to its location in a document for quickly retrieving search results and location in a text. While the reverse index plays an important role in most search operations, for annotations it is also necessary to use **forward indexes**. These play an important role in result delivery processes such as keyword-in-context, lists and grouping functionality. We provide three main types of forward indexes for each available document, based on *position*, *parent id* and *object id*. To handle the complexity of the data structures represented in the tokenstream a special codec to extend the default postings format is used. By using this codec, the required files for the forward index are automatically constructed and managed, not only when Lucene commits indexed documents to a segment, but also on optimizing and merging cores.

From a maintenance perspective this provides the possiblity to index collections separately into separate cores and simply activate the new core using SOLR. Alternatively, separate cores can be merged into a single core as well. This is particularly useful when working with large data sets. One of our projects aims to make large Dutch annotated text corpora available to the scientific community. Using separate cores for the indexing process allows us to prepare these corpora in parallel and perform additonal checks on metadata and content before merging or adding the new core to the set of SOLR cores available for search and retrieval of the main environment.

The document indexing process itself is a complex process of mapping where the original text document is converted to a stream of tokens with possibly multiple tokens on the same position, addition of prefixes, interpretation of ranges and sets of positions as a token, assignment of unique subsequent id's to all tokens and finally the construction of indivual payloads containing all the right references. Depending upon the annotation structures and requested search capabilities, a series of choices has to be made to index available documents. We provide a configurable tokenizer that has been tested against FoLiA, ELAN and a WPL Sketch Engine like format. Testing on TEI is underway. Configuration of the tokenizer is specified in a separate file allowing search options to be adjusted and configured for specific needs.

The configurable tokenizer is particularly useful in situations where multiple annotation formats are to be expected and have to be stored in the same index. Apart from the mapping challenges of multiple tag sets the relevant information content that needs to extracted from the annotated documents mostly occurs in different locations in the document. The indexer provides the possibility to receive instructions on which configuration file to use when indexing a document. In one of our current projects this is used to index multiple annotation formats produced by different annotation services. Here, users transfer their textual documents to their workspace, request some processing service to work upon the document and the resulting annotated document is automatically indexed using this system. Since many of the annotation services produce different formats this method at least provides the possibility of searching and retrieving them from the same index. This method is also considered useful in combination with our archiving software allowing us to make the contents of the archive available not only at the metadata level, but also at the annotated content level while maintaining the flexibility to allow multiple annotation formats to be stored in our repository.

4 CQL support

Using the new approach of prefixes and adjusted payloads, the default query parsing mechanisms of SOLR and Lucene in most cases will not suffice. Therefore we support Corpus Query Language introduced by the Corpus Workbench. A CQL parser, based on JavaCC, has been developed mapping CQL queries onto the provided query parser. This makes it possible to search for CQL constructions such as :

```
[pos = "LID"] | word = "the"][pos= "ADJ"]?[pos="N"]
```
or
```
[lemma="er"][]{0,2}[lemma="over"][pos="WW" & feat.wvorm="vd" & feat.buiging="zonder" & feat.positie="vrij"]
```
or
```
< entity = "loc"/ > within (< s/ > containing [t_lc = "amsterdam"] )
```

Although many of the basic queries for annotated texts seem to be covered through this, especially more complex queries involving syntactic phenomena, such as dependencies, are expected to demand additonal query language features to be able to take full advantage of the capabilities of the index.

5 Result delivery

Many of the basic use cases involve delivery of keyword-in-context, supported by the system. More advanced information retrieval requires grouping, listings or gathering of statistical information, such as relative frequency distributions. In most of these cases we take the approach that the produced results are configurable, either by directly specifying a constant value (e.g. for specifying the context window for kwic representation) or by taking advantage of available CQL query comparisons. This makes is possible, for example, to calculate both the absolute and relative frequencies of specific words, annotations or patterns across the result set. Furthermore, additional metadata may also provide additional criteria to limit or classify result sets and can be used in the statistical analysis. It thus becomes possible to compute statistical information on arbitrary queries, e.g. [pos=”ADJ”][pos=”N”] / [pos=”N”], as shown in the figure below.

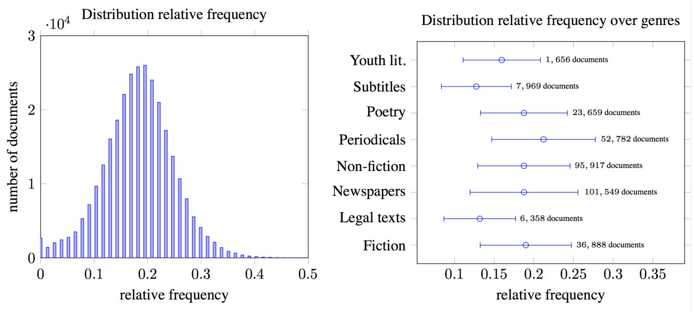

Figure 1: Statistics for the number of [pos="ADJ"][pos="N"] divided by the number of [pos="N"] on each document containing [pos="N"] and at least 500 words, computed with a single request in 47,021 ms on a corpus of 294874 documents

A commonly requested feature for information retrieval systems working on text corpora is the ability to extract term lists from retrieved result set. Our solution is capable of delivering term vectors on any of the annotation layers available in the index (words, lemmas, part of speech, named entities or otherwise) and, combined with the statistical features described above, deliver information on the distribution characteristics in the result set.

Many of the features described here have been integrated into the working environment of one of our main infrastructure projects where the translation from the statistical information to a user friendly projection for the end user is performed using pie charts, time line views and other visualization methods.

6 Performance

Performance measures strongly depend on the number of documents, document size, type of query performed and available hardware options. Using a collection of over 6.5 million documents containing 4.5 billion words and 19.9 billion annotations we provide a an overview of expected performance. Underlying hardware platform is Dell PowerEdge R730 - Xeon E5-2630L v3 (1.8GHz) - 8 x 16 GB - 2 x 2 TB HDD.

Table 1: simple CQL query response times on a corpus of 4.476.372.137 words (6.573.211 documents)

CQL query	#documents		#hits	
[t_lc="de" | t_lc="het" | t_lc="een"]	5, 704, 222	1.1 s	268, 297, 812	11.9 s
<s /> containing [t_lc="de"]	5, 441, 349	8.1 s	88, 760, 006	24.4 s
[pos="LID"][pos="ADJ"]{1,2}[pos="N"]	1, 507, 295	7.9 s	24, 260, 370	64.0 s

Table 2: Sample grouping feature on a corpus of 4.476.372.137 (6.573.211 documents)

CQL query	Group	Time	Results		
				documents	**hits**
[pos="ADJ"][t_lc="haat"]	t_lc	52.8 s	vol haat	390	436
			eeuwige haat	45	236
			algemeenen haat	212	235
			blinde haat	158	167
			felle haat	148	159
			diepe haat	119	129
			…	…	…
[pos="ADJ"][t_lc="liefde"]	t_lc	105.6 s	grote liefde	2, 405	2, 973
			zyne liefde	900	1, 968
			eene liefde	1, 204	1, 865
			vol liefde	1, 423	1, 766
			ware liefde	1, 305	1, 715
			myne liefde	617	1, 586
			…	…	…

7 Conclusion

We provide a scalable SOLR/Lucene based solution capable of performing CQL queries across a range of annotation formats. Query capabilities have been extended into the statistical domain allowing gathering of statistical information from the retrieved result sets. Our system supports retrieval of term vectors across search results documents. Result delivery features key word in context, listings and groupings.

While the solition provides a highly scalable solution the results should be further compared against other corpus search engines, such as Corpuscle(Meurer 2012), Corpus Workbench, Sketchengine and BlackLab.

References

[Banski2013]Banski, Piotr, et al. "KorAP: the new corpus analysis platform at IDS Mannheim." Proceedings of the 6th Language and Technology Conference. 2013.

[Brouwer2013]Brouwer, Mathijs, et al. "Nederlab, towards a Virtual Research Environment for textual data." Abstract submitted for The 9th edition of the Language Resources and Evaluation Conference (LREC 2014). 2013.

[Brugman2016]Brugman, Hennie, et al. "Nederlab: Towards a Single Portal and Research Environment for Diachronic Dutch Text Corpora." Proceedings of LREC. 2016.

[Evert2011] Evert, Stefan, and Andrew Hardie. "Twenty-first century corpus workbench: Updating a query architecture for the new millennium." (2011).

[Kilgarriff2004]Kilgarriff, Adam, et al. "Itri-04-08 the sketch engine." Information Technology 105 (2004): 116.

[Odijk2015]Odijk, J. E. J. M. "Linguistic research with PaQu." Computational Linguistics in The Netherlands Journal 5 (2015): 3-14.

[Vandeghinste2014]Vandeghinste, Vincent, and Liesbeth Augustinus. "Making a large treebank searchable online. The SoNaR case." Challenges in the Management of Large Corpora (CMLC-2) Workshop Programme. 2014.

[Reynaert2014]Reynaert, Martin, Matje van de Camp, and Menno van Zaanen. "OpenSoNaR: user-driven development of the SoNaR corpus interfaces." COLING (Demos). 2014.