MTAS: A Solr/Lucene based Multi Tier Annotation Search solution

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

Hennie Brugman
Meertens Institute
The Netherlands
`hennie.brugman@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. With Mtas, an acronym for Multi-Tier Annotation Search, we add annotation 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. We present a configurable indexation process, supporting multiple document formats, and providing extended search options on both metadata and annotated text, such as advanced statistics, faceting, grouping and keyword-in-context. Mtas is currently used in production environments, with up to 15 million documents and 9.5 billion words. Mtas is available from GitHub[1].

1 Introduction

Many solutions providing search on both plain text and metadata rely on the inverted index based Apache Lucene[2]. The existing and popular extension Solr offers additional features such as distributed indexes, scalability, and load balanced querying 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 and 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 already available, amongst others from the CLARIN community, e.g. BlackLab, KorAP, SketchEngine, Corpus Workbench, PaQu, GrETEL, Corpuscle to name a few. Various considerations have led us to develop a new initiative in this area, most notably scalability and integrated metadata/annotation search. We provide an overview of functional requirements from our infrastructure projects and scientific users, and elaborate on development decisions taken in relation to existing solutions. After a short introduction of the implemented CQL support, we discuss performance and capabilities of statistics, faceting, grouping and termvectors. We conclude with performance, consistency checks and some suggestions on future work.

2 Requirements

In this section we describe the main high-level requirements that determine the general scope and direction of Mtas development. While it is certainly true that, depending on the envisaged use case or in comparison with other systems, additional requirements could be formulated that equally deserve attention, a number of specific strategic, functional and operational requirements provides the main

[1] https://github.com/meertensinstituut/mtas
[2] https://lucene.apache.org/

foundation for our development ambitions. From a strategic perspective, multi tier annotation search represents one of the key components that supports the data management life cycle in our domain. Annotated text is essential to unlock (textual) data contents beyond the metadata level. Therefore we consider it imperative that we build up internal knowledge and experience in this area. Moreover, in order to achieve sufficient control and room for experimentation we strive for close collaboration with system providers or, if close collaboration proves impossible, investment in independent system creation. Close collaboration with researchers ensures that we invest in those functional areas that are immediately of interest to the research community we work with. Finally, our operational requirements are related to ease of deployment, testing and maintenance of the system. For example, ease of integration of new collections and the ability to quickly deploy various instances of the system for testing or production purposes are very important.

Besides the Nederlab project[3] (Brouwer, et al., 2014) serving as one of the primary use cases and application platforms for the system design, the development of Mtas is firmly situated in the CLARIN domain as part of the Dutch CLARIAH project. One of the major requirements therefore is the ability to include arbitrary (CMDI) metadata schemas in search processes. Considerable experience is at hand in making metadata available in metadata search processes, e.g. in the CLARIN VLO. This search domain however needs to be extended to include annotated text, containing multiple, often interdependent, annotation layers. These layers consist of normalized and spell-corrected texts, translations, lemma, part of speech (including feature lists), named entities, entity links to external knowledge bases such as 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 level of individual annotation layers, support must be provided for multivalued attributes, differentiation between multiple set values (e.g. to cater for multiple tag sets simultaneously occurring in source 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, such as for example hierarchies. The choice for Corpus Query Language is motivated by the fact that this is commonly used by various systems in the community, albeit with local differences in interpretation. Also, with the advances of the Federated Content Search program in CLARIN it is anticipated that Corpus Query Language will be adopted to extend the current SRU/CQL (Search/Retrieval via URL and Contextual Query Language) capabilities allowing for a more easy alignment with FCS activities.

With respect to result delivery, both (annotated) documents and keyword-in-context representations must be delivered, as well as statistical information regarding absolute and relative frequencies and hit distributions across result sets. Result set distributions must be calculated across each available metadata dimension, including time intervals, and not only over single but also multiple metadata dimensions, e.g. a distribution across both 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 result set and type of annotation; word forms, part of speech, named entities, etc. Finally, the system should be highly scalable, be able to work across multi-billion word corpora, be easily manageable and be freely available for use to a wide user community under an open source license[4].

2.1 Current solutions

A choice between search engines is often a balancing act between one's requirements and depends upon one's functional scope, corpus size, available expertise or conditions of use. Several systems designed for searching annotated text structures are currently available, each with its own strengths, weaknesses and track record, e.g. BlackLab (Reynaert, et al., 2014), KorAP (Banski, et al., 2013), Corpus Workbench (Evert & Hardie, 2011), Sketch Engine (Kilgarriff, et al., 2004), PaQu (Odijk, 2015), GrETEL (Vandeghinste, et al., 2014) and Corpuscle (Meurer, 2012).

Although many of these provide partial coverage of the listed requirements, as can be seen from our findings listed in Table 1, none of them provides a balanced coverage to be immediately applicable to

[3] https://www.nederlab.nl/
[4] https://github.com/meertensinstituut/mtas

our 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. It should also be noted that the presented list is not considered to be exhaustive, but indicates the functional scope of the project. Other, non-functional, factors in the system choice were a preference towards a widely adopted and supported open source framework with clear design principles an active community maintaining the framework. This allows us to benefit from new insights gained and new progress made by the wider community and minimizes the risk of getting stuck in a dead-end program. Should such a framework reach its end-of-life then most likely it will be possible to secure a graceful migration path towards other systems.

As a development approach, new features were to be added using an iterative approach with short development cycles. This helps to identify risks in early stages of development and prevents over-engineering. 'Gold plating' is to be avoided, focusing on only delivering those features that are relevant to the use cases and research questions at hand.

Looking at the requirements it becomes clear that the *open source* and *scalability* requirements narrow the choice to only a limited number of systems. The Corpus Workbench is considered to be nearing its end-of-life judging from the new developments at IMS. Initial tests with Neo4J indicated that, even with adjustments, for graph databases such as Neo4j, performance and scalability is expected to remain problematic: the more general graph structure prevented us to take full advantage of the sequential nature of annotated text with reasonable response times. The Corpuscle system, besides its small user community and our inability to locate the source code, is considered to a rather exotic implementation being written in Common Lisp.

The BlackLab solution, being based also on Lucene, may seem to have some resemblances with our approach, although in Mtas we choose to take a completely different approach to represent distinctive annotation layers and hierarchical structure in Lucene. However, in our initial attempts to extend the BlackLab functionality, starting with taking advantage of the advanced scalability, sharding and other options provided by Solr, such as faceting, 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 re-implement 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 maintenance and further development if the system. With Solr/Lucene one gets speed and scalability almost for free which makes it 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. Sharding refers to the possibility to create horizontal partitions of the data. Horizontal partition is a term that originates from the database community and refers to splitting one or more tables by row. In Solr, shards have one or more replicas and each replica is a core. A core refers to a single index and associated transaction log and configuration files. In our use cases, individual collections or sub collections can be indexed into separate cores and, using the sharding features, be addressed separately or collectively.

	Corpus Workbench	Sketch Engine	PaQu	GrETEL	BlackLab	Corpuscle	Neo4J	Solr	Solr + Mtas
Open source	✓	✗	✓	?	✓	?	✓	✓	✓
Highly scalable	✓	✓	✗	✗	✓	?	✓	✓	✓
Distributed search	?	?	✗	✗	✗	?	✓	✓	✓
Arbitrary (CMDI) metadata schemas	✗	✗	✗	✗	✗	?	✓	✓	✓
Annotated text	✓	✓	✓	✓	✓	✓	✗	✗	✓
- full support annotations	✗	✗	✓	✓	✗	✗	✗	✗	✓
- hierarchical structure	✗	✗	✓	✓	✗	✗	✗	✗	✓
- configurable mapping of input format on index	✗	✗	✗	✗	✗	✗	✗	✗	✓
- support FoLiA annotation format	✗	✗	✓	✗	✓	✗	✗	✗	✓
- corpus query language (CQL)	✓	✓	✗	✗	✓	✓	✗	✗	✓
- full statistics	✗	?	✗	✗	✗	✗	✗	✗	✓
- term vectors over any result document set	✓	✓	✗	✗	✗	✗	✗	✗	✓
- grouping	✓	✓	✗	✗	✓	✓	✗	✗	✓
- faceting	✗	✗	✗	✗	✗	✗	✗	✗	✓
- keyword in context (kwic)	✓	✓	✓	✓	✓	✓	✗	✗	✓

Table 1: Overview findings native coverage of our specific requirements for several existing solutions directed towards searching annotated text and/or structures.

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, several values of *genre*, e.g. *fictie* and *proza*, can be associated with each document. For fields containing textual data, position information is available for each value. To this existing Lucene approach, we add annotations and structure by using prefixes to distinguish between text and different annotations. Annotations and structure are stored together with text in a separate designated type of Lucene field, thus providing simultaneous access to traditional Lucene fields for storing metadata features and Mtas enabled content. 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 token stream to the indexer. Ranges of words, distinct sets of words (e.g. named entities) and hierarchical relations are stored as a *payload* A payload, in Lucene terms, refers to an arbitrary array of bytes associated with a Lucene token at a certain position. Several additional extensions are used implementing different query strategies, most of them extend default Lucene methods. Our extension assumes a basic tokenization enriched with annotations on both single and multi-token levels. In most cases word level tokenization is used. It also possible to define other tokenizations, for example at the morpheme level, and in this case words will usually span multiple tokens. Table 2 provides a representation of various layers expressed in our index. Single and multi-token elements can be distinguished in the *position* column and parent hierarchy may be derived from the *parent* column. Prefixes displayed in this table are described through the configurable mapping (see 3.1).

Id	Offset		Position		Parent	Payload	Prefix	Postfix
72	1443	8100	0	5			s	s
0	1515	1536	0		72		t	Amsterdam
1	1515	1536	0		72		t_lc	amsterdam
2	1605	1616	0		3	1.0	feat.spectype	deeleigen
3	1540	1668	0		72	1.0	pos	SPEC
4	1674	1683	0		72		lemma	Amsterdam
53	5458	5617	0		72	0.885417	chunk	NP
56	6122	6324	0		72		entity	loc
5	1808	1822	1		72		t	is
6	1808	1822	1		72		t_lc	is
7	1894	1905	1		10	0.999891	feat.wvorm	pv
8	1938	1949	1		10	0.999891	feat.pvtijd	tgw
9	1984	1995	1		10	0.999891	feat.pvagr	ev
10	1826	2037	1		72	0.999891	pos	WW
11	2043	2052	1		72		lemma	zijn
54	5625	5777	1		72	0.993895	chunk	VP
…								
48	5019	5105	0		49		dependency.dep	
49	4880	5120	0	1	72		dependency	su
47	4936	5014	1		49		dependency.hd	
…								
66	7550	7628	1		68		dependency.hd	
67	7633	7714	4		68		dependency.dep	
68	7410	7729	1, 4		72		dependency	predc
…								

Table 2 Sample representation of Mtas index showing offsets, positions and postfix information for various prefixes.

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 inverted index plays an important role in most search operations, especially for dealing with multiple tiers in 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 currently provide three main types of forward indexes for each available document, based on position, parent id and object id. These indexes are created and updated automatically when documents are added or deleted, or when cores are merged or optimized[5].

From a maintenance perspective, this approach provides the possibility to index collections separately into separate cores and simply activate new cores 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 additional checks on metadata and content before merging or adding the new core to the set of Solr cores available for search and retrieval in the production environment.

3.1 Indexing and configurable mapping

The document indexing process itself is a complex process 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 ids

[5] A special codec extending the default postings format is used. By using this codec, the required files for the forward index are automatically constructed and managed.

to all tokens and finally the construction of individual payloads containing all the right references. Depending upon the processed 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, a WPL Sketch Engine like format and TEI among others. Configuration of this tokenizer can be specified in a separate file allowing search options to be adjusted and configured for specific needs. The indexer can thus be instructed to use a different indexing strategy for each individual file to be indexed. Also, the process may be instructed to differentiate between locally available files and remote ones.

The configurable tokenizer is particularly useful in situations where documents using multiple annotation formats are imported into the same index. Apart from the mapping challenges of multiple tag sets, the relevant information content that needs to be extracted from the annotated documents often occurs in different locations in the document. In our projects, by using configuration files put together to match both specific document structure and user requirements, we are also able to collect all information for more complex structures like e.g. entities, paragraphs and chunks from the documents, and include this in the token stream.

The indexer can be instructed about which configuration file to use when indexing a specific document type. 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 a personal 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 such annotations from one uniform 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.

We also took direct advantage of this setup 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 part 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).

4 CQL support

Using the new approach based on prefixes and adjusted payloads, the default query parsing mechanisms of Solr and Lucene in most cases will not suffice. Our choice of query language support is furthermore largely motivated by the idea that this should match closely with current practices in the field. This reduces the learning curve for our potential end user community. This also has the practical advantage that front-end development may reuse some of the visual query construction mechanisms already available in the domain targeted at various proficiency levels (beginner, advanced, expert) of end users. 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 Mtas query methods. Not only does this language seem to be easily apprehensible by users with more specific search requirements, the syntax of this query language also directly matches the prefix/postfix structure we incorporated into the Mtas index structure. A query for a *word* with *part-of-speech* annotation *noun*, represented in the index as a single position token with prefix *pos* and postfix value *N*, is expressed in CQL as

```
[pos="N"]
```

while the search for a paragraph, represented in the index as a multiple position token with prefix *p*, can be performed using angular brackets

```
<p/>
```

The use of the and-operator & and the or-operator |, together with the use of parentheses, provides advanced options for more complex conditions on single words. Multiple conditions may be lined up into sequences, a question mark can be used to mark a part as optional; multiple occurrences of the same part may be indicated with a single number or a minimum and maximum between curly brackets, e.g.

```
[pos="LID" | lemma="the"][pos="ADJ"]{0,2}[pos="N"]
[pos="ADJ"]([word="," | word="and"][pos="ADJ"])?[pos="N"]
```

This can be even further extended by combining these constructed conditions on words and sequences to a new condition by using *containing* or *within* operators, e.g.

```
<entity="loc"/> within (<s/> containing [lemma="amsterdam"])
```

The conventions to search for words at the beginning or end of multiple token annotations, e.g. an adjective at the start of a sentence, or a noun within three positions before the end of a paragraph, closely follow the syntax as known from other formats using this angle bracket notation.

```
<s>[pos="ADJ"]
[pos="N"][]{0,2}</p>
```

By using a dash, the position of a word in the original document can be referred to, e.g. to get the first word of a document, or to query for an adjective within the first ten words

```
[#0]
[#0-9 & pos="ADJ"]
```

In addition to the standard CQL constructs shown above some additional extensions were made to the allow operations that were encountered in specific use cases under consideration while developing Mtas. One feature that was introduced is the ability to request the full prefix list from the system. Although not directly expressed in CQL, it is highly useful to be able to automatically extract this list given that the underlying index may contain arbitrary prefixes depending on the configuration settings while indexing. This list also distinguishes between single and multiple positions allowing to adjust any CQL query accordingly.

The not operator is supported by specifying an exclamation mark in front of the prefix

```
[!pos="ADJ"]
```

Many CQL implementations allow the user to put a word between double quotes as a short hand notation for querying for a single word using the bracket notation. However, since Mtas offers the user full freedom in choosing prefixes to distinguish the different annotation layers, such a notation would be ambiguous without defining the default prefix to apply for such requests. Therefore, requests like the following only can be formed when such a default prefix is provided

```
"the" [pos="ADJ"]?[pos="N"]
```

Furthermore, the multitude of annotation layers may result in queries not matching some results because of annotations unknown to the user. For example, some texts may contain anchors, indicators of some event occurring after the first and before the second word, that would for the following two examples cause the second query to have matches that a user unfamiliar with these anchors would have expected also to match the first query

```
[pos="ADJ"] [pos="N"]
[pos="ADJ"]<anchor/>?[pos="N"]
```

To overcome this problem, an optional *ignore query* can be provided together with each CQL expression, to define everything that should be ignored when searching for sequences and recurrences. By describing the anchor in such an ignore query, the first of the two expressions will now match exactly the same expressions as the latter.

In some of our use cases, the use of a lexicon service to expand queries was required. Instead of defining an explicit value between double quotes, we therefore allow the use of a variable as postfix within the condition of a single position token, where a list of possible values for this variable should be always provided, e.g.

```
[pos="ADJ"][lemma=$1]
```

where $1 will be replaced with items from a list, e.g.: `{"horse", "cow"}`

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

5 Result delivery

One of the primary use cases for the system, the Nederlab project, currently provides access, both in terms of metadata and annotated text, to over 15 million items for search and analysis as specified in Table 3. Collections are added and updated regularly by adding new cores, replacing cores and/or merging new cores with existing ones. Currently, the data is divided over 23 separate cores. The Nederlab underlying hardware platform is a Dell PowerEdge R730 - Xeon E5 - 2630L v3 (1.8GHz) - 8 x 16 GB - 2 x 2 TB HDD with 67 GB of the available 128 GB memory assigned to Solr.

	Total	**Mean**	**Min**	**Max**
Solr index size	1,146 G	49.8 G	268 k	163 G
Solr documents	15,859,099	689,526	201	3,616,544

Table 3: Size and content of the Solr index consisting of 23 separate cores within the Nederlab project (January 2017).

	Total	**Mean**	**Min**	**Max**
Words	9,584,448,067	654	1	3,537,883
Annotations	36,486,292,912	2,488	4	23,589,831

Table 4: Number of available words and annotations for the 14,663,457 documents containing annotated text (January 2017).

For 14,663,457 of these documents, as described in Table 4, annotated text varying in size from 1 to over 3.5 million words is included. The remaining part of the 15,859,099 documents mentioned in Table 3 concerns descriptions of persons, volumes and other items for which only metadata is available.

On querying the index, Solr allows to filter documents with conditions on metadata in regular fields. By providing a parser plugin, this filtering is extended with the possibility to use CQL conditions on annotated text within reasonable time. Searching for all 1,944,167 documents containing an adjective followed by a noun[6] takes less than 4 seconds, searching for the 161.734 documents with a sentence containing both the word *amsterdam* and the word *rotterdam* takes 6 seconds. Additional restrictions on metadata only reduces the number of potential hits, and therefore result in faster search results.

Many of the features described below have been integrated into the working environment of one of our main infrastructure projects where the translation of statistical information to a user-friendly representation for the end user is performed using pie charts, time line views and other visualization methods. Notice that, although we tried to use both illustrative and realistic examples, the quality of the provided annotations in part of the resources is not quite up to standard, which may sometimes lead to unexpected results.

[6] Only 2.217.779 documents contain text with part of speech annotation.

5.1 Statistics

Whereas Solr only produces statistics on the number of documents, additional methods had to be implemented to produce, within the (filtered) set of documents, statistics on the number of words and the number of hits for specific CQL queries. Furthermore, computing statistics on the composition of these numbers within documents should be possible, e.g. statistics on the number of hits for a CQL query divided by the total number of words within each document.

Number of documents	138,152	Geometric mean	0.16290333781014
Sum	23894.977875106	Variance	0.002695471072453
Mean	0.17296150526309	Population variance	0.0026954515615442
Sum of squares	4505.2933656369	Standard deviation	0.051917926311179
Sum of logs	-250690.38070139	Median	0.17269981462327
Maximum	0.45167923235093	Skewness	0.0043594322359785
Minimum	0.00070521861777151	Kurtosis	0.59294319460155
Quadratic mean	0.18058553060646		

Table 5: Statistics for the number of adjectives followed by a noun divided by the number of nouns within all documents containing an adjective followed by a noun, and with at least 2000 words, computed in 53 seconds.

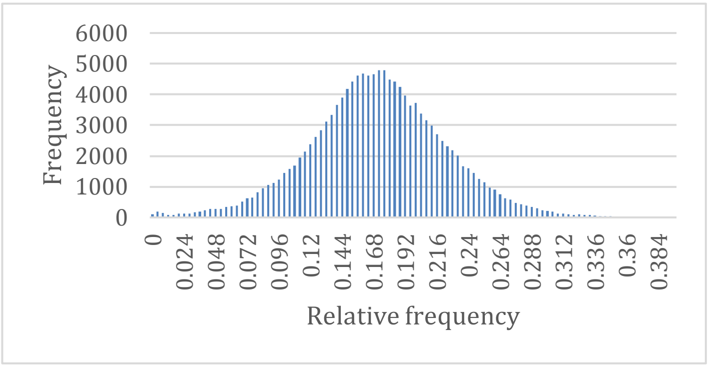

Figure 1: Frequency distribution for the number of adjectives followed by a noun divided by the number of nouns within all documents containing an adjective followed by a noun, and with at least 2000 words, computed in 56 seconds.

As illustrated in Table 5, several statistical properties can be computed, where more advanced items like *median*, *skewness*, and *kurtosis* tend to require more time, since not only a few aggregations but all individual values on document level have to be collected from the participating cores.

Besides these properties, also frequency distributions can be retrieved that can be used to create a graphical representation of the distribution of the studied value. Figure 1 demonstrates such a distribution for the example described in Table 5.

5.2 Faceting

Taking advantage of the available metadata, statistics can be computed for each occurring value of one or multiple metadata fields. This basically extends the available Solr options for faceting with the previously described statistical extensions.

Again, the mean number of adjectives followed by a noun divided by the number of nouns is computed, but now for all documents within a decade. In Figure 2 this mean value is plotted for each decade between 1270 and 2010 for all documents containing at least one noun.

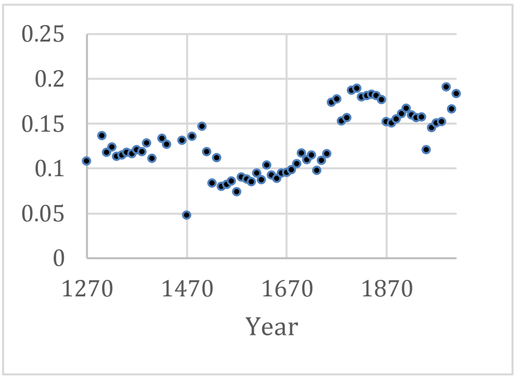

Figure 2: The distribution of the number of adjectives followed by a noun divided by the number of nouns: the mean value computed over all documents with publication year within the same decade is plotted against all 74 adjoining decades between 1270 and 2010 for all documents containing at least one noun.

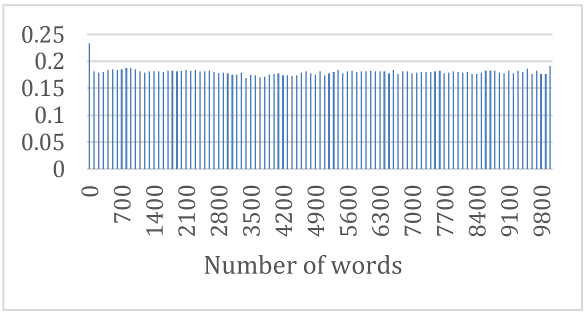

Figure 3: The distribution of the number of adjectives followed by a noun divided by the number of nouns. The mean value computed over all documents with size within the same range of size 100 is plotted against all 100 adjoining ranges of document sizes between 0 and 10,000 for all documents containing at least one adjective followed by a noun.

Besides based on classic metadata fields like year of publication, faceting can also be based on the number of words per document, which is directly derived from the annotated text during indexing. This is illustrated in Figure 3, where instead of using decades, the values found are grouped by and plotted against number of words.

Also, more advanced statistics are available, e.g. the standard deviation as a measure of spread around the mean value as has been illustrated in Figure 4.

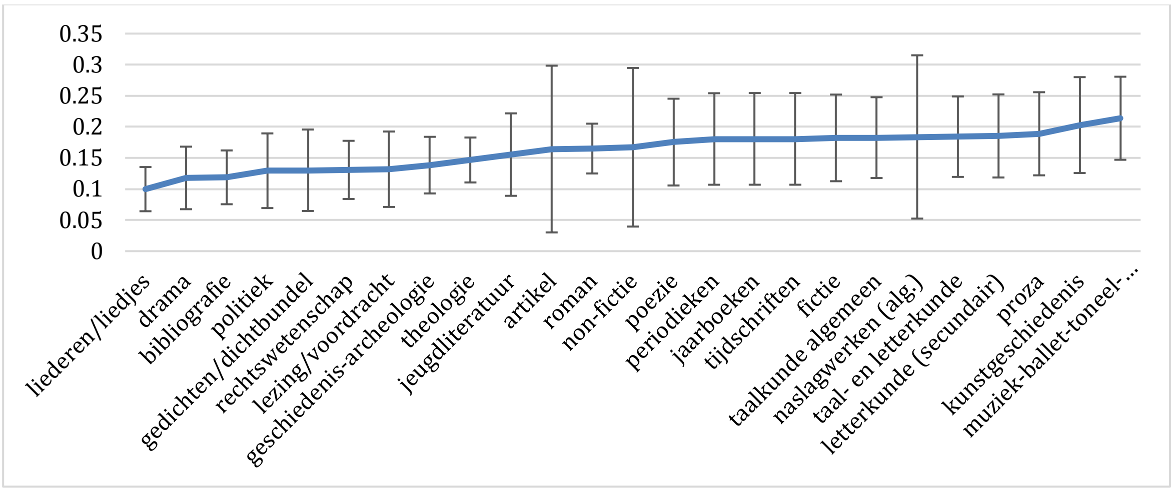

Figure 4: The distribution of the number of adjectives followed by a noun divided by the number of nouns. The mean value and standard deviation computed over all documents within the same genre is plotted for all genres, sorted ascending by mean value, for all documents containing at least one noun.

Finally, in Figure 5 the evolution of the distribution for the mean sentence length of documents within a decade is plotted, illustrating the possibility to study statistics over multiple dimensions.

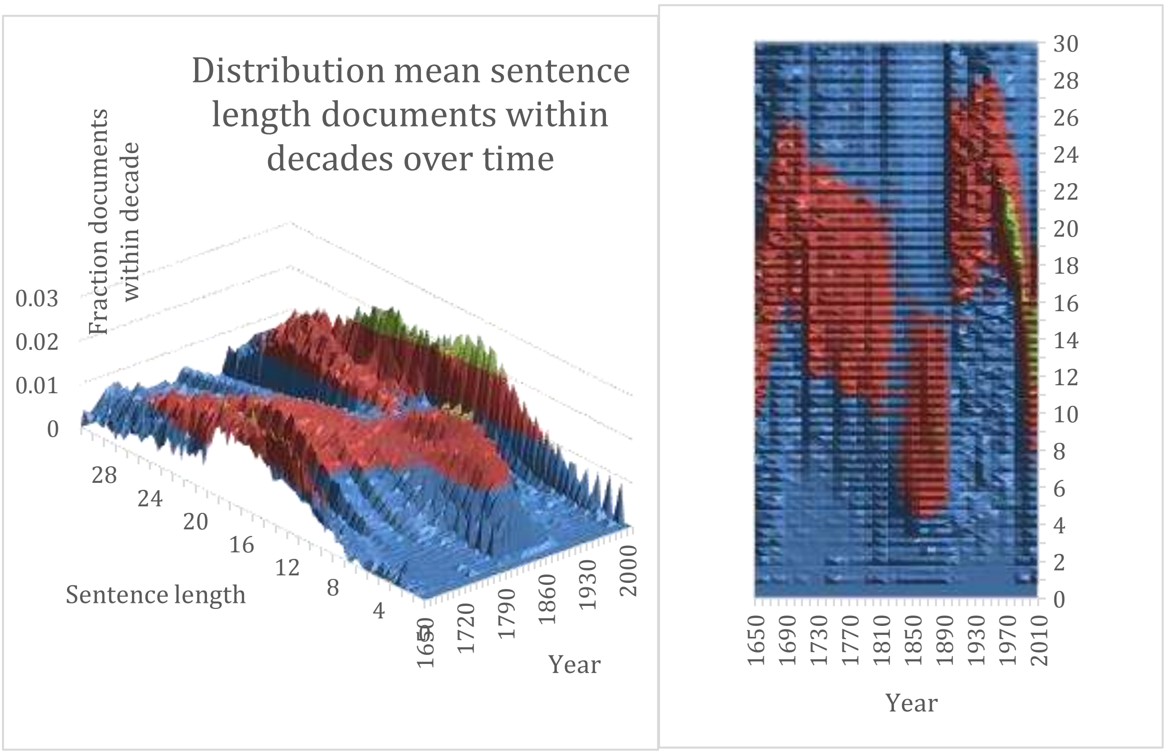

Figure 5: Evolution over time of the distribution for the mean sentence length of documents within a decade; Distribution is plotted for all documents containing at least one sentence and published in or after 1650, necessary data computed in 198 seconds.

5.3 Grouping

The previously presented possibilities of faceting can be seen as statistically grouping results based on metadata. Grouping of query results based on one or multiple annotation layers on the other hand produces lists of occurring values with number of occurrences and documents, sorted by frequency in descending order. These type of queries can be computationally quite expensive, especially for queries with large numbers of hits and also large numbers of distinct associated values for the annotation layer(s).

When grouping the occurring part-of-speech annotations associated with a query for the words *de* or *het*, the number of hits is large: 459,302,283 in 15,859,099 documents. However, there are only ten distinct associated values for the part-of-speech annotation layer, therefore performing this grouping is still possible within reasonable time. The result is illustrated in Figure 6 with, due to the large range, frequencies plotted on a logarithmic scale for each of the occurring values.

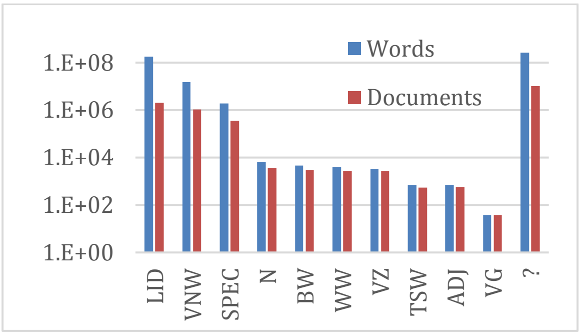

Figure 6: Number of occurrences and documents by grouping a query on "de" or "het" on the associated part-of-speech over all documents, computed in 209 seconds.

On grouping the occurring terms for adjectives followed by lemma *liefde* and for adjectives followed by lemma *haat*, the number of distinct associated values for the occurring terms, 7,400 and 2,605, is much higher. But, since the numbers of hits, 86,973 and 10,381, are substantially lower, performing such a grouping can still be performed within reasonable time. The result, the most frequent adjectives for lemma's *liefde* and *haat*, are listed in Table 6 and Table 7

ADJ + "liefde"	documents	hits
grote	2,539	3,115
zyne	901	1,968
eene	1,213	1,867
vol	1,454	1,798
ware	1,337	1,749
myne	617	1,586
groote	1,182	1,504
christelijke	991	1,385
eeuwige	899	1,375
oude	1,038	1,166

Table 6: Grouping of the occurring terms for the 86,973 adjectives followed by lemma "liefde". Computing the 7,400 unique values took 295 seconds, the 10 most frequent values are listed together with number of hits and documents.

ADJ + "haat"	documents	hits
vol	333	363
eeuwige	46	237
algemeenen	211	234
blinde	166	175
felle	149	160
diepe	122	133
fellen	117	128
doodelijken	115	121
onderlinge	105	115
ouden	100	113

Table 7: Grouping of the occurring terms for the 10,381 adjectives followed by lemma "haat". Computing the 2,605 unique values took 181 seconds, the 10 most frequent values are listed together with number of hits and documents.

Taking advantage of the full possibilities offered by CQL, grouping can be used to retrieve more complex results, e.g. all person or location entities occurring within sentences containing the word *rembrandt*, as listed in Table 8 and Table 9. Notice the multiple token results in the list of person entities.

	documents	hits
rembrandt	3,475	15,009
rubens	354	506
van den	178	274
van gogh	173	206
vermeer	147	192
jan steen	130	156
van der helst	86	152
saskia	69	148
hals	83	144
shakespeare	100	138

Table 8: The 10 most frequent person entities within sentences containing Rembrandt, computed in 65 seconds.

	documents	hits
amsterdam	407	696
nederlandse	153	216
nederland	143	178
nachtwacht	122	170
holland	96	132
leiden	82	125
un	61	122
land	96	118
rijn	92	118
hollandse	69	92

Table 9: The 10 most frequent location entities within sentences containing Rembrandt, computed in 77 seconds.

Finally, grouping does not need to be restricted to a single layer of annotation, as can be seen when grouping on term, part-of-speech, and form for all occurrences of the lemma *zijn*. The 5 most frequent combinations occurring in documents from 1800 are listed in Table 10.

term	pos	tense	form	documents	hits
is	WW	present	pv	913	77,542
was	WW	past	pv	665	44,558
zijn	WW	present	pv	251	24,159
zijne	VNW	-	-	198	20,546
waren	WW	past	pv	456	14,794

Table 10: The 5 most frequent combinations of term, part-of-speech and form when grouping for occurrences of the lemma zijn in documents from 1800, computed in 29 seconds.

5.4 Termvector

A commonly requested feature for information retrieval systems working on text corpora is the ability to extract term lists from retrieved result sets. Our solution is capable of delivering termvectors 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. The list of terms can be sorted on term or frequency, where the latter is more computationally expensive and complex when retrieving results over multiple cores, and can be restricted by a regular expression and/or a user defined set of words.

	Documents	Total	Mean	Median	Max	Mean	Median	Std. deviation	Kurtosis	Skewness
Term	**Frequency**					**Relative frequency**				
welke	4,170,151	12,242,628	2.94	1	3,996	0.0032	0.0021	0.0046	223.4	10.9
zijne	2,628,115	7,541,324	2.87	1	4,523	0.0030	0.0020	0.0036	56.1	5.29
hunne	2,291,385	5,237,717	2.29	1	3,994	0.0025	0.0016	0.0031	63.8	5.53
goede	2,268,787	4,081,615	1.80	1	1,318	0.0027	0.0015	0.0043	709.2	14.2
einde	1,954,052	3,351,655	1.72	1	893	0.0021	0.0012	0.0031	4932	27.3

Table 11: List of the 5 most frequent terms containing 5 letters and ending with e for all 14,663,457 documents. Besides the number of documents, and further statistics on the frequency, also statistics on the relative frequency within the documents are computed in 172 seconds.

Computing data for the list in Table 11, describing the most frequent terms containing 5 letters and ending with -*e* for all documents in all participating cores, took less than 3 minutes. Besides the number of documents, statistics on both frequency and relative frequency are included. The total length of the termvector, describing the total number of matching terms, is not computed by default, since this potentially is a very heavy operation.

5.5 Document

Although computing the full termvector over a set of documents can be quite expensive, as noted in the previous section, this type of computation is less excessive when only a single document is involved. In Table 12, the frequency distribution for a Dutch translation of the bible is illustrated, together with the ten most frequent terms for this document. Computing these results took approximately 6 seconds.

words	2,409,382
unique	47,421

term	frequency
ende	126,688
de	97,311
van	50,364
het	38,744
in	34,849
dat	33,020
die	30,010
den	29,271
te	27,758
en	26,516

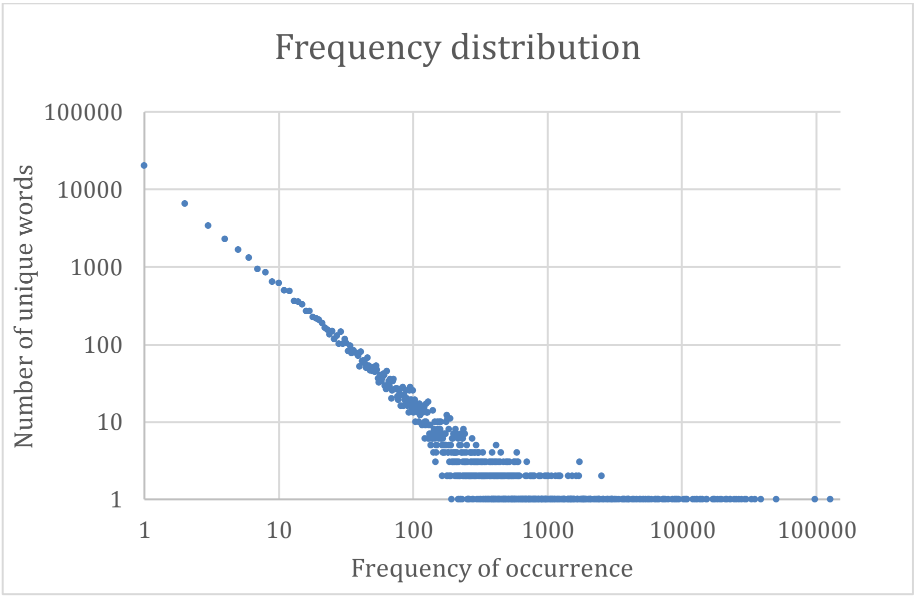

Table 12: Frequency distribution for a single document: the frequency distribution for a Dutch bible translation is computed within 6 seconds; the total number of words and total number of unique words together with the 10 most frequent terms are listed and the distribution of the frequency of occurrence is plotted, demonstrating behaviour as predicted by Zipf's law.

5.6 Keyword-in-context (kwic)

Using Mtas as a plugin, the default presentation of results from Solr, providing (part of) the list of matching documents and listing stored values for all or limited set of fields is extended by providing the option to list a set of matches to one or multiple CQL queries. This keyword-in-context like functionality provides the user with the option to investigate specified annotations on or around the location of hits within the annotated text. This includes multiple-position tokens, positions and hierarchical structure, as can be seen from the example in Figure 7 where such a kwic result from a query for

```
[pos="LID"][pos="ADJ"]
"Amsterdam"
```

is visualized. The application of a forward index, as described previously, makes the additional time needed to generate these representations almost always negligible compared to the time needed for the execution of the query involved.

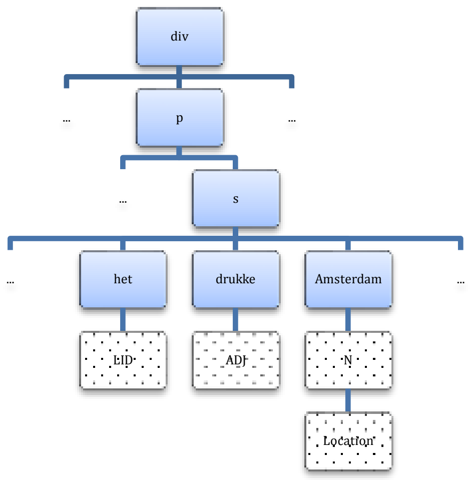

Figure 7:Keyword-in-Context result for a query to an article and and adjective followed by "Amsterdam"

5.7 Consistency checks

When developing Mtas, no suitable reference sets of resources, queries and results were available to test the provided functionality. Direct checks on consistency of the indexation process were therefore limited to manual tests on relatively small documents. However, much of the tokenization process can be tested indirectly with queries, using general knowledge on the structure of the resources. For example

- For most documents, the number of words satisfying the condition of being contained within a sentence, must equal the total number of words.
- For most part-of-speech annotated documents, the sum of the total number of words within each occurring part-of-speech value, must also equal the total number of words.

Furthermore, many aspects of the implemented Mtas functionality could also be tested by comparing results for specific queries. Consistency in these results from different methods, some of them even being native Solr functionality, does indirectly provide a test on those methods themselves. For example

- The number of documents for each term in the native Solr termvector should equal the number of documents in the Mtas termvector result, and also the number of hits for each separate term from the Mtas termvector should equal the number of hits in the Mtas statistics for a query to this specific term.
- The number of documents in native Solr facets should equal the number of documents in the corresponding Mtas facet, and also the number of hits within the Mtas facet response should be reproducible by requesting Mtas statistics with corresponding conditions on the metadata.

Finally, for the implemented Mtas functionality, consistency checks were done in comparing query results over separate cores with results where sharding was applied.

6 Performance

Performance measurements strongly depend on the number of documents, document size, type of query performed and available hardware options [7] Together with differences in implemented functionality, this makes it difficult to really compare Mtas performance with e.g. the solutions listed in Table 1. We tried to provide some indication of the performance by including the required search time in most of the previously introduced examples.

The advantages of a distributed setup in the process of adding and updating data have already been mentioned. The influence of distribution on performance of our implemented system can be illustrated more explicitly. For the graph in Figure 8, basic statistics for the number of sentences were computed multiple times for a setup with a single core, and for setups with multiple cores.

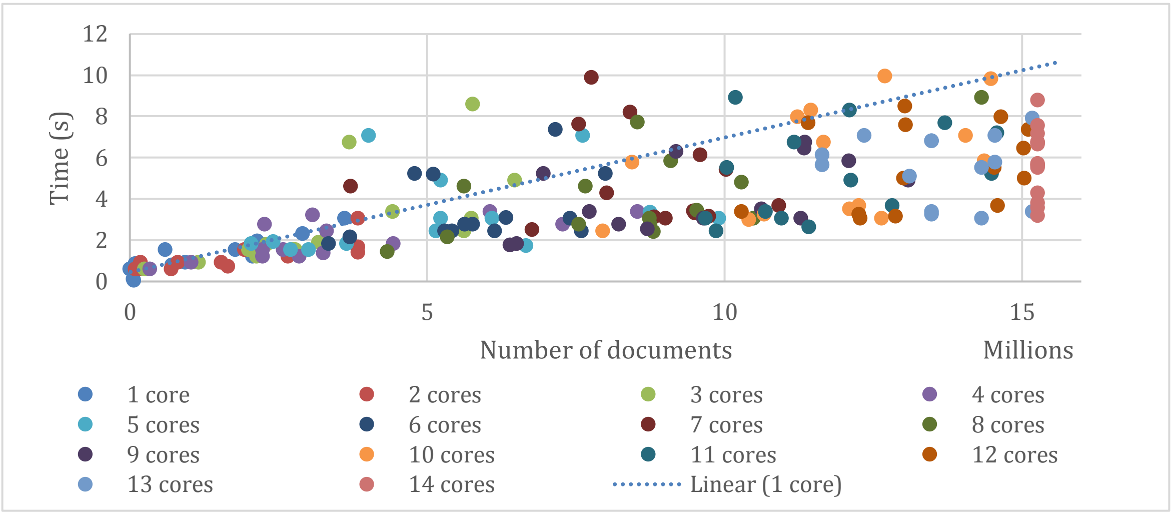

Figure 8: Measurements of query time for basic statistics on the number of sentences against number of matching documents for a single core setup, and for setups with multiple cores varying from two to fourteen.

As can be seen from the graph, most measured times for setups with multiple cores lie below the linear trendline from the single core setup. Total query time is likely strongly to be determined by disk access speed, where the spread in time possibly is caused by the availability time being influenced by disk access in the same location shortly before. The upper and lower limit in this band do not seem to be heavily influenced by the number of cores and/or documents.

[7] Underlying hardware platform is Dell PowerEdge R730 - Xeon E5 - 2630L v3 (1.8GHz) - 8 x 16 GB - 2 x 2 TB HDD; currently, with 67 GB of the available 128 GB memory assigned to Solr.

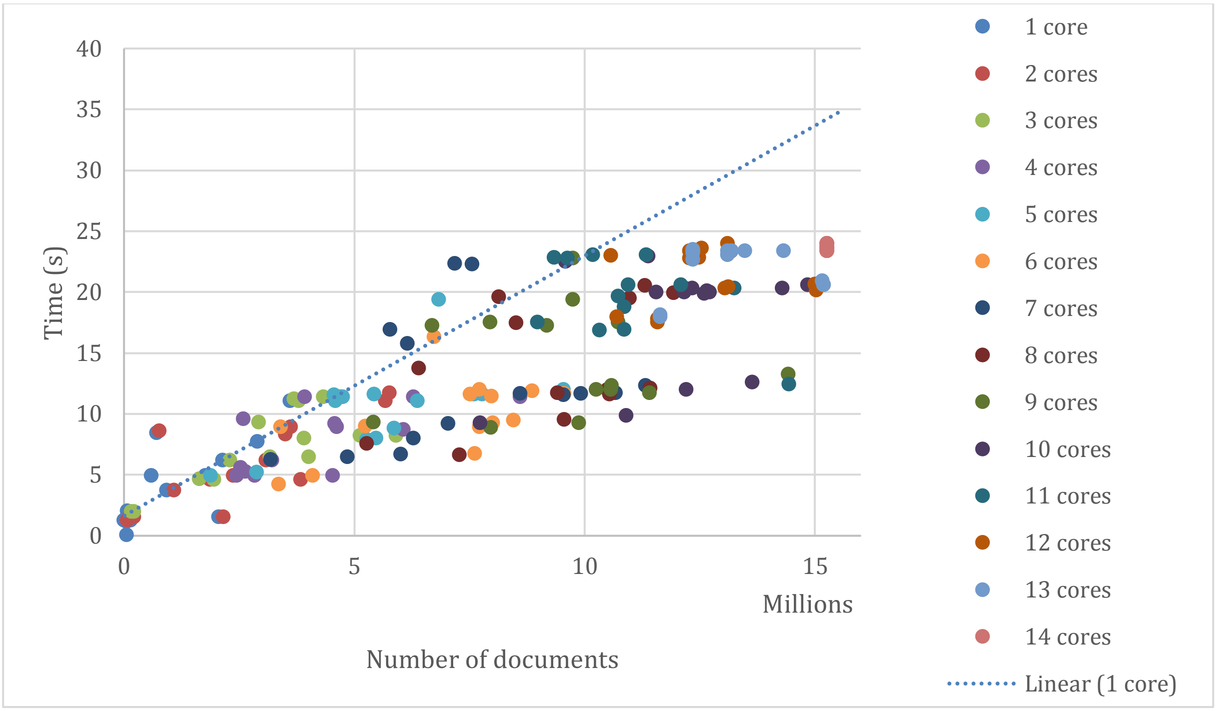

Figure 9: Measurements of query time for computing a termvector against number of matching documents for a single core setup, and for setups with multiple cores varying from two to fourteen.

Another illustration of performance for distributed queries is given in Figure 9, where the required time to compute a termvector sorted by frequency is measured, again in a single and multiple core situation. Again, most measured times for setups with multiple cores lie below the linear trendline from the single core setup. Furthermore, we seem to be able to distinguish two levels in this plot that can be explained by the algorithm used to efficiently compute a termvector over multiple cores. In this approach, sometimes a second termvector has to be computed by individual participating cores to be able to compute the required merged result, especially when the number of documents and/or participating cores increases.

7 Conclusion and future work

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 termvectors across search results documents. Result delivery features key word in context, listings and groupings.

Although most of the requirements for e.g. the Nederlab project are probably sufficiently covered by the current implementation, multiple additional features seem to be desirable. By using technical or performance related considerations and by watching the search and analysis techniques applied in the research fields involved, several suggestions can be made:

- Exploring the hierarchical structure, already fully integrated in the index structure, is not covered very well in the CQL query language. Further development, e.g. integrating an additional query language to exploit this structure and possibly adjusting the index structure to new types of queries, should preferably be done in collaboration with specialized researchers and based on specific use cases.
- Queries containing CQL conditions seem to perform reasonably well, but little or no attention is paid to including e.g. the number of hits in determining the score value for each document.

Within the use cases at hand, currently no clear thoughts seem to be available with respect to the desired manners of weighing documents. Further adjustments to the scoring mechanism should be accompanied by theory and/or explanations to guarantee acceptance and understanding.

- Within the research areas of use cases involved, new techniques concerning clustering and analysis seem to gain popularity, especially when huge amounts of data are involved. Although some experiments related to these techniques are planned, these projects all seem to rely on very basic and often inefficient use of the possibilities offered by Mtas. Often, users plan to export potentially very large result sets and analyze them with the external tools they are used to. Integrating the computation of e.g. covariance matrices, and furthermore offering options to reuse the found factorizations in further search requests, although probably still keeping the cluster and factorization computations outside Mtas, seems a far more efficient approach, probably also directly applicable by other research projects.
- Whereas currently in Mtas annotated text is assumed to contain a basic granularity on word level, enriched with annotations on both single and multiple word level, some textual data does not completely fit into this scheme. Fully including annotated text containing a translation for example will be problematic, since translations will align probably on sentence or paragraph level, but not (always) on the level of words. Including a decomposition of words into syllables and morphemes also does not seem to fit the current structure.
- As illustrated in examples above, many statistical properties on the number of hits already can be determined. Less attention is paid to e.g. the distribution of these properties within single documents, bootstrapping methods and applying more advanced techniques in comparing documents, although these techniques do seem to applied regularly in the research areas involved.
- Producing termvectors over multiple cores is reasonably fast, but only regular expressions or explicit lists can be used to restrict the outcome. There seems to be a need to reduce these termvectors even further by using conditions on additional layers, e.g. only nouns. To achieve this, without falling back on far less performing grouping methods, adjustments to the indexation process have to be made. This may also improve the speed of other queries involving conditions on multiple annotation levels on the same position or word.
- To get the most relevant terms, TF-IDF for termvectors should be made available as statistic and sort condition, both on document level and for multiple documents within some configurable reference set.

Important for all future development seems to be to focus on a combination of performance and more advanced analysis techniques, preferably driven by use cases from active projects and in collaboration with researchers with experience and basic knowledge of the algorithm involved.

References

Banski, P. et al., 2013. KorAP: the new corpus analysis platform at IDS Mannheim.. s.l., s.n.

Brouwer, M. et al., 2014. Nederlab, towards a Virtual Research Environment for textual data.. s.l., s.n.

Brugman, H. et al., 2016. Nederlab: Towards a Single Portal and Research Environment for Diachronic Dutch Text Corpora.. s.l., ELRA, pp. 1277-1281.

Evert, S. & Hardie, A., 2011. Twenty-first century Corpus Workbench: Updating a query architecture for the new millennium. Birmingham, s.n.

Kilgarriff, A., Rychly, P., Smrz, P. & Tugwell, D., 2004. Itri-04-08 the sketch engine. Lorient, s.n.

Meurer, P., 2012. Corpuscle – a new corpus management platform for annotated corpora. In: G. Andersen, ed. Exploring Newspaper Language: Using the web to create and investigate a large corpus of modern Norwegian. s.l.:John Benjamins.

Odijk, J., 2015. Linguistic research with PaQu.. Computational Linguistics in The Netherlands, Volume 5, pp. 3-14.

Reynaert, M., Camp, M. v. d. & Zaanen, M. v., 2014. OpenSoNaR: user-driven development of the SoNaR corpus interfaces.. s.l., s.n., pp. 124-128.

Vandeghinste, Vincent & Augustinus, L., 2014. Making a large treebank searchable online. The SoNaR case.. Reykjavik, s.n., pp. 15-20.