Berri Corpus Manager: A Corpus Analysis Tool Using MongoDB Technology

Hugo SANJURJO-GONZÁLEZ

Department of Information Technology, Electronics & Communications, University of Deusto, Spain

Abstract. Nowadays, there are many options for corpus linguistic analysis that make use of different approaches for corpus storage. There are tools based on SQL databases, dedicated implementations such as CQP/CWB and others that employ plain-text corpora. NoSQL databases have been widely used for big data, data mining and even sentiment analysis. However, as far as we can see, there is a lack of a widespread concordancer or consolidated framework that makes use of MongoDB architecture for the purposes of corpus linguistics. This paper aims to describe the architecture of a software that allows users to analyse monolingual and bilingual parallel corpora with grammatical annotation using MongoDB technology. Our premises are that MongoDB is ideal for non-structured data and provides high flexibility and scalability, so it may be also useful for corpus linguistic research. We analyse functionalities of MongoDB such as text search indexes and query format in order to examine its suitability.

Keywords. Corpus analysis tool, concordancer, NoSQL database, MongoDB, corpus linguistics

1. Introduction

There are many approaches for corpus storage regarding corpus analysis software. Sanjurjo-González [1] offers a comprehensive survey of the available corpus analysis software characterised by linguistic and technological features. In this survey, we can find tools based on SQL databases, mixed approaches that employ SQL, software that makes use of dedicated implementations for corpus indexing and querying [2] and last, software that employs plain-text corpora.

The most popular approaches for concordancers are those that are SQL-based, or dedicated implementations that use CQP/CWB [3]. For instance, SQL is used in corpus.byu.edu² and PELCRA³, CQP/CWB is used in CQPWeb [4] and ACM⁴ [5], among others. Although these approaches offer different performances using extreme scale cor-
poras, there is no significant difference employing a smaller corpus for a linguistic oriented user carrying out the most common corpus linguistic operations such as KWIC (Key Word In Context) concordances, frequency lists, collocations, etc.

MongoDB was officially released for production purposes in 2011 and is one of the most used NoSQL databases. It is a document-oriented database with high scalability and flexibility that stores data in BSON (binary JSON) documents. It is widely used to store data for sentiment analysis [6,7,8], data mining [9,10,11] or even big data [12,13,14]. In [15], Coole, Rayson and Mariani carried out different experiments using NoSQL databases, SQL databases and CWB/CQP with extreme scale corpus, showing that NoSQL databases like MongoDB or Cassandra are viable solutions for performing KWIC searches employing several servers. For this reason, their performance may be good enough using a large corpus even without clustering.

Consequently, this paper aims to describe the architecture of the software that allows users to analyse monolingual and bilingual corpora with grammatical annotation using MongoDB technology. Our premises are that MongoDB is ideal for non-structured information and corpus texts might be considered in this way, providing high flexibility and scalability.

The remainder of this article is organized as follows. In Section 2, we survey relevant research to the scope of the paper. We then describe the architecture and technology of the proposed software in Section 3. We introduce experiment results in Section 4. Finally, we describe conclusions in Section 5.

2. Related Work

NoSQL databases have gained popularity with emerging demands of scalable databases, mainly related to big data research [16]. This emerging trend of NoSQL paradigm for handling big data information systems should be at least considered for corpus linguistics research. Our premises are that MongoDB is ideal for non-structured information such as corpus texts thanks to its high flexibility, vertical scalability and schemaless architecture. However, as far as we can see, there is no widespread concordancer or consolidated framework that makes use of MongoDB architecture for the purposes of corpus linguistics. In fact, some approaches make use of MongoDB for ad hoc solutions. We can mention Perkins [17] that presents a proof of concept that combines NLTK [18] with MongoDB database; Frey, Glaznieks and Stemle [19] download and merge data as a corpus zero using MongoDB; Gutierrez-Vasques, Sierra and Pomp [20] combine MongoDB with Lucene/Solr to build and query their corpus; and last, Dorantes et al [21], develop a search interface using custom Python scripts and MongoDB database without describing any additional details about the implementation.

As previously mentioned, the most relevant work using MongoDB technologies in corpus analysis may be Coole, Rayson and Mariani [15]. In this case, they made their experiments in a clustering environment as a consequence of the extreme size of the corpus. In the present work, we do not use cluster components as the size of a typical corpus does not require as much power. In addition, we want to test MongoDB technology in the

---

5https://www.mongodb.com/
6https://lucene.apache.org/
most common software development environment, composed of only one server, which does not require complex configurations or high hardware specifications.

3. Architecture and Technology

3.1. Data Model

Berri Corpus Manager employs MongoDB for corpus storage. For each corpus, we store language, size, whether it includes any type of annotations or if it is part of a parallel corpus (Figure 1). As a consequence of the flexibility and schema-less architecture of the database, the design is very simple and can be easily modified if required. Grammatical annotations are included using prefix notation. Last, we employ sentence alignment for parallel corpora.

3.2. User Interface

User interface provides a way to query the corpus. This interface is based on Sanjurjo-González and Izquierdo [22] and offers a user-friendly layout that enables effective corpus analysis and simplifies the pattern (Figure 2) and parallel queries (Figure 3) that may or may not include annotations.

3.3. Implementation

To implement the software, we employ Python technologies, more concretely Flask\(^7\) and PyMongo\(^8\).

Flask is a lightweight micro-framework based on Werkzeug web server and Jinja2 templates. It allows us to employ Python language to develop web applications. Flask maps uri to different Python functions. Thus we can use Python in the server for natural

\(^7\)https://flask.palletsprojects.com/en/1.1.x/
\(^8\)https://docs.mongodb.com/drivers/pymongo
language processing tasks such as grammatical tagging, tokenisation and alignment, as well as for scripting tasks such as formatting and document corpus selection.

PyMongo is a Python distribution containing tools to work with MongoDB. By PyMongo we can easily interact with the MongoDB database using Python. As a consequence, we populate and query corpora employing available methods. It is also necessary to parse queries from the user interface to the MongoDB query language. To do that, we make use of the text search feature of MongoDB query language that includes a text index. In Figure 4, text variable refers to the query including or not including annotations.
As it would be expected, text search increases performance, but it also has several issues regarding the most common queries in corpus linguistics:

1. It provides language specification, however, it uses stemming and removes stop words, so it might be useless if the user queries for a particular phrase of word. For this reason, it is better not to employ language specification. Size of the database increases significantly for this selection.

2. Text indexes do not support partial word searches, so they cannot be used to search patterns such as prefixes, suffixes, or others. To overcome this issue, we must employ $regex operator, which affects negatively the performance of the query (Figure 5).

3. As a consequence of the designed data model, grammatical annotations are included using prefix notation and some queries may be affected, for instance, if a user wants to search for all the words that are proper or common names.

4. As is the case with all the databases that are not linguistically oriented, the count function returns a count of documents, in our case sentences, that would match a query. However, one sentence may include more than one instance, so we need an additional processing. To do that, we make use of the MongoDB aggregation pipeline functionality. This functionality allows us to create a framework for data aggregation modelled on the concept of data processing pipelines. It works relatively well but it takes one minute or more if the number of results is high.

Parallel queries functionality of Berri Corpus Manager supports queries in both subcorpora at the same time in order to search the correspondences. For instance, we may be interested in searching what type of expressions translate the English word "table" into Spanish (tabla or mesa). As MongoDB is not designed for this type of parallel queries, we search in both subcorpora at the same time, and obtain sentences for which identifiers are common in both results (Figure 6).

3.4. System Architecture

Berri Corpus manager operates across a simple Model-View-Controller pattern (Figure 7): Flask (controller), MongoDB (model), Jinja2 and Bootstrap v.4 (view). Berri Corpus Manager interacts with the corpus by means of PyMongo utilities. Python scripts are
used to parse user queries into understandable commands using MongoDB format as well as natural language processing tasks.

4. Experiments

Berri Corpus Manager is able to handle monolingual and bilingual corpora with grammatical annotation. For our experiment, we selected the well-known corpus Europarl [23]. More concretely, we employed English and Spanish subcorpora that were already aligned and have a size of 49,429,773 and 51,603,160 words, respectively. Grammatical annotation has been carried out using Spacy.9

MongoDB presents a good performance if the search makes use of the text search index. It takes more time if it makes use of regex patterns. For instance, in our experiments, searching for all the occurrences of the word "a" takes 20 seconds using the text index and 30 seconds if it is not used.

Berri Corpus Manager employs server pagination, so there are no issues returning results. However, as it was previously mentioned, returning the total number of matches might take some time if there is a high number of results.

Last, it should be mentioned that query plan cache methods save a lot of time for recurrent operations. Therefore, high performance queries can takes less time than expected if they have been previously executed.

9https://spacy.io/
5. Conclusions

This paper presented one of the first implementations of a concordancer using MongoDB technology. Employing lightweight technologies such as Flask makes development extremely quick. MongoDB has several advantages: it is schema-less, flexible and scalable. However, as it has not been designed for specific purposes of corpus linguistics, it also presents some issues, for instance, the unavailability of the partial search using text indexes, which has a serious effect on the queries’ performance, or in returning the number of occurrences of a concordance, as it is based on the number of documents that match the query instead of the number of occurrences of the query. If these issues are solved, MongoDB can become a prominent technology for corpus linguistic software development as a consequence of its flexibility and fast deployment.

References


