Evaluating and Integrating Databases in the Area of NLP

Wahed Hemati, Alexander Mehler, Tolga Uslu, Daniel Baumartz, Giuseppe Abrami
Goethe University Frankfurt
Frankfurt, Germany
{hemati, mehler, uslu, baumartz, abrami}@em.uni-frankfurt.de

Abstract

Since computational power is rapidly increasing, analyzing big data is getting more popular. This is exemplified by word embeddings producing huge index files of interrelated items. Another example is given by digital editions of corpora representing data on nested levels of text structuring. A third example relates to annotations of multimodal communication comprising nested and networked data of various (e.g., gestural or linguistic) modes. While the first example relates to graph-based models, the second one requires document models in the tradition of TEI whereas the third one combines both models. A central question is how to store and process such *big* and *diverse* data to support NLP and related routines in an efficient manner. In this paper, we evaluate six *Database Management Systems* as candidates for answering this question. This is done by regarding database operations in the context of six NLP routines. We show that none of the DBMS consistently works best. Rather, a family of them manifesting different database paradigms is required to cope with the need of processing big and divergent data. To this end, the paper introduces a web-based *multi-database management system* (MDBMS) as an interface to varieties of such databases.

Keywords: Multi-Database Systems, Big Humanities Data, Natural Language Processing, Database Evaluation

1. Introduction

Digital humanities and related disciplines deal with a variety of data ranging from large index files (as in the case of word embeddings (Mikolov et al., 2013)) to inclusion hierarchies as mapped by document models in the tradition of TEI (Burnard and Bauman, 2007) and models of nested and networked data as exemplified by multimodal communication (Carletta et al., 2003). With the availability of web-based resources we observe both an increase concerning the size of such data – that is, in terms of big data – and the need to cope with this diversity simultaneously within the same project: lexica (represented as networks) are interrelated, for example, with large corpora of natural language texts (represented as tree-like structures) to observe language change (Kim et al., 2014; Michel et al., 2011; Eger and Mehler, 2016). Another example is given by digital editions (Kuczera,) in which annotations of textual data are interrelated with annotations of pictorial representations of the underlying sources (Leydier et al., 2014; Lavrentiev et al., 2015). We may also think of text mining in the area of learning analytics based on big learning data, in which textual manifestations of task descriptions are interrelated with student assessments and qualitative data (Robinson et al., 2016; Crossley et al., 2015). To cope with such diversities, databases are required that allow for efficiently storing, retrieving and manipulating data ranging from distributions to tree-like structures and graphs of symbolic and numerical data. As a matter of fact, such an omnipotent database is still out of reach. Thus, the question is raised *which existing database technology based on which paradigm performs best in serving these tasks*. Alternatively, the question is raised *which combination of these technologies best fits these tasks*.

In this paper, we evaluate six *Database Management Systems* (DBMS) as candidates for answering these two questions. This is done by regarding elementary database operations that underly typical scenarios in NLP or natural language annotation. We show that none of the DBMS under consideration consistently works best. Rather, families of such DBMS manifesting different database paradigms cope better with the rising needs of processing big as well as divergent data. To demonstrate this, the paper introduces a web-based *multi-database management system* (MDBMS) that allows for integrating families of databases. This is done to provide a single interface for querying different databases. To this end, the MDBMS encapsulates the specifics of the query languages of all databases involved. By using it, modelers are no longer forced to choose a single database, but can benefit from different databases that capture a variety of data modeling paradigms. This enables projects to simultaneously manage and query a variety of data by means of a multitude of databases – in a way that also allows for processing big data. Such a versatile MDBMS is especially interesting for NLP frameworks (Castilho and Gurevych, 2014; Hemati et al., 2016; Hinrichs et al., 2010; OpenNLP, 2010; Popel and Žabokrtský, 2010) that still do not store annotations of documents in a query-able format. In cases where such annotations are stored according to the *UIMA Common Analysis Structure* (UIMA-CAS), serialization is based on XML-files. This makes operations such as collecting statistical information from documents very time-consuming, since one first has to deserialize each file involved. The same problem occurs when trying to add annotation layers. For additionally annotating, for example, named entities, one has two options: either one runs the underlying UIMA pipeline together with the named-entity recognition (NER) from scratch or the serialized output of the pipeline is deserialized and then made input to NER. Obviously, both variants are very time-consuming. A better option would be to let NER work only on those annotations that are needed for annotating named entities. However, this presupposes that all annotations are

stored in a query-able format. By example of UIMA-CAS the present paper proposes a MDBMS that allows exactly this: to query annotations of documents according to varying data models even in cases where the amount of stored data ranges in terms of big data.

2. Related Work

For testing our framework, we include databases[1] that address different core tasks.

Until now, there has not been much research on the comparative evaluation of DBMS in the context of NLP. What is missing is a comparison of databases according to competing (relational, graph- or document-oriented) paradigms. Rather, databases are assessed in isolation. As an example for document-oriented databases, (Richardet et al., 2013) use MongoDB to provide an interface for UIMA-based applications. (Fette et al., 2013) present a relational framework for storing CAS objects in a MySQL database. Another example is given by (Hahn et al., 2008) who serialize UIMA-CAS objects by means of PostgreSQL. Since graph databases have become popular in digital humanities (Kuczera,), there are many examples according to this paradigm. (Lyon, 2016) use Neo4j to mine word associations. (Ganesan et al., 2010) introduce a graph-based summarization framework. (Mihalcea and Tarau, 2004) exemplify a graph-based ranking model for text processing. (Usbeck et al., 2014) use graph-based representations of texts and linked data for disambiguating named entities. (Rousseau et al., 2015) describe an approach to graph-based text classification in which texts are represented as graphs of words, while (Tixier et al., 2016) introduce a graph-based approach to keyword extraction. Finally, (Müller and Hagelstein, 2016) present an approach to analyzing and visualizing textual content that combines a graph database (Neo4j) with MongoDB. As enumerated in Section 3., we evaluate all of these and related databases by means of storing and retrieving annotations generated by six NLP operations.

3. Database Management Systems

This section describes the DBMS included to our evaluation. We use the UIMA type system of (Castilho and Gurevych, 2014) as a reference to refer to annotation layers to be managed by the databases. Further, the evaluation utilizes TextImager (Hemati et al., 2016) to get access to a large variety of NLP routines as test cases. The valuation scenario refers to the annotation of text corpora along six annotation layers that are output by six selected routines. Any annotation information is stored by means of CAS objects to map between the UIMA type system on the one hand and the different DBMS on the other. In this section our question is then how the selected DBMS store input corpora and annotations. Section 4. will then evaluate how these annotations are managed by the DBMS.

3.1. Baseline Scenario: File System

To get a baseline scenario, we generate compressed files out of serialized UIMA-CAS objects output by NLP and store them in the file system. Serialization uses XMI[2], a standard for representing objects in XML (Grose et al., 2002). Note that for each operation at document level, any compressed file has to be de-compressed and -serialized. Obviously, this results in a memory overhead.

3.2. MongoDB

As an example of a scalable document-orientated NoSQL database we evaluate `MongoDB`.[3] To this end, we serialize UIMA-CAS objects into binary-encoded JSON Objects. Due to its document-oriented character, these objects can be added directly to MongoDB next to the type system. This makes it extremely flexible especially in the case of frequently changed type systems.

Because of being schema-free, MongoDB supports two ways for representing relations among objects: in terms of *embedded documents* or *document references*. Embedded documents store objects relations in a single Binary JSON document. To this end, input documents are denormalized and stored contiguously on disk. This results in better reading performance when querying entire documents, since only one query is required (Copeland, 2013). However, document size can grow rapidly as each annotation layer is represented as a separate array of objects in the respective document. Since MongoDB has a size limit of 16 MB per document, storage of large documents is considerably limited in this scenario. Alternatively, relations can be stored by means of document references. In this case, one speaks of *normalized data* (Dayley, 2014). In our evaluation of MongoDB, every annotation of a CAS object is stored as a separate JSON document. This allows for representing and storing larger CAS objects than in the case of embedded documents. Database size is further reduced by avoiding redundancy. A disadvantage of normalized data is that multiple lookups are required when resolving dependencies and references (Dayley, 2014).

3.3. Cassandra

`Apache Cassandra`[4] is a column oriented NoSQL database that is horizontally scalable. To this end, Cassandra replicates data to multiple nodes of a cluster to ensure fault tolerance (Hewitt, 2010). For every annotation layer of the underlying type system we generate a separate table whose attributes correspond to those of the focal layer. Note that Cassandra does not allow for joining tables. From the point of view of the relational paradigm, this is problematic. Thus, dependency parses, semantic roles, anaphora and related relational data can hardly be modeled using Cassandra. However, an alternative is to distribute such data and synthesize it at runtime using query families.

3.4. MySQL

For evaluating `MySQL`[5], we use the NLP-related database schema of (Fette et al., 2013). It consists of six tables for storing annotations next to the type system.

[1] `http://db-engines.com/en/ranking`

[2] `http://omg.org/spec/XMI/`

[3] `https://mongodb.com/`

[4] `https://cassandra.apache.org/`

[5] `https://mysql.com/`

3.5. BaseX

BaseX[6] is a light-weight XML document-oriented database using an XPath/XQuery 3.1 processor. In our setup, UIMA-CAS objects are serialized into XMI documents and added to this XML database. XPath/XQuery provides functionality for querying XMI representations of CASes in accordance with the type system.

3.6. Neo4j

Neo4j[7] is a highly performant NoSQL graph database addressing networked data (Miller, 2013). In our experiment, we represent a text corpus as an adjacency graph in Neo4j. That is, each input text is represented by a node linking to all its token nodes whose syntagmatic order is mapped by token links. Further, any annotation is represented by nodes in Neo4j so that no pair of different nodes instantiates the same attribute-value pair. That is, the attribute-value pair (*POS*, *verb*), for example, is bijectively mapped to a node in Neo4j. This approach reduces the number of nodes, while raising the number of edges.

4. Experiments

We now evaluate the DBMS of Section 3. regarding the dual task of processing diverse as well as big data. To this end, we created a corpus of 70,000 German Wikipedia articles, each containing annotations of (1) *tokenization*, (2) *lemmatization*, (3) *POS tagging*, (4) *dependency parsing*, (5) *NER* and (6) *time recognition*. Table 1 lists the number of annotations generated for each of these layers. In the following sections, we evaluate the DBMS with respect to writing and reading the respective annotations. This evaluation is further differentiated by evaluating the performance of the DBMS with regard to querying attribute values and relational data.

Layer	Count
Document	70,000
Paragraph	598,041
Sentence	1,520,441
Token	31,647,060
Character	169,131,822
POS	31,647,060
Lemma	31,647,060
Dependency	31,647,492
Named Entity	2,078,212
Time	1,039,106

Table 1: Number of annotations per annotation layer.

4.1. Storage Performance

For each database of Section 3. we implemented a separate *UIMA Writer* to transform CAS objects output by the selected NLP routines to the CAS adaptations of the respective databases. Figure 1 shows the performance of each database in terms of single threaded write operations. Table 2 shows the time taken by the DBMS to store all documents. The time is measured in terms of serializing and storing CAS objects based on the respective database format. Obviously, while file system-based storage is fastest, MySQL is worst. Table 2 shows usage of disk space induced by all inserts. While the file system is still best, the worst case is now BaseX.

[6] http://basex.org/
[7] https://neo4j.com/

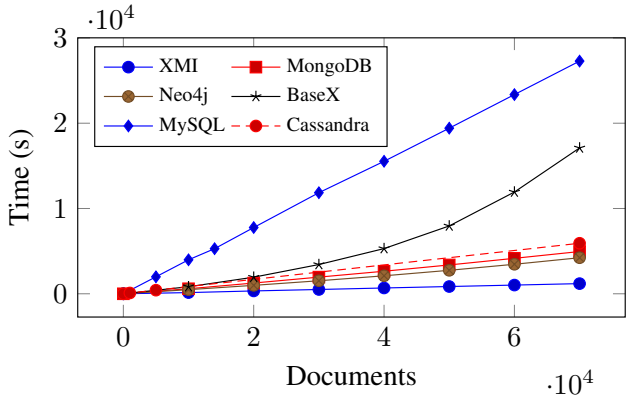

Figure 1: Time of writing using the corresponding DBMS.

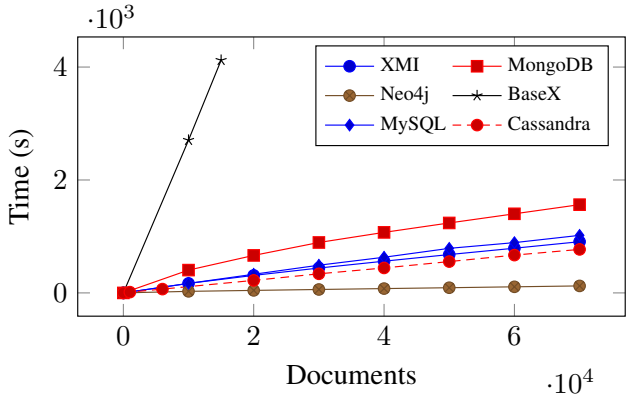

Figure 2: Time need to read documents.

	XMI	Mongo	Neo4j	BaseX	MySQL	Cas.
Runtime (s)	1,190	4,953	4,227	17,101	27,283	5,858
Disk (GB)	4.8	15.8	22.6	33.0	28.4	10.8

Table 2: Performance statistic for storing documents.

4.2. Reading Performance

For each database we implemented a *UIMA CollectionReader* for mapping between CAS adaptations of DBMS and NLP routines. This mode concerns situation in which subsequent NLP routines operate on pre-annotated data. Figure 2 as well as Table 3 show the performance of the DBMS when performing single threaded read operations. Now, Neo4j outperforms all its competitors, while BaseX is still worst.

4.3. Query Performance

Databases typically provide expressive query languages. The more expressive this language, the more informative the data that can be queried, for example, for text mining, distant reading or text visualization. Such a query mode is particularly interesting for NLP frameworks (see Section

XMI	Mongo	Neo4j	BaseX	MySQL	Cas.
906 s	1,563 s	123 s	20,217 s	1,009 s	770 s

Table 3: Time (in seconds) taken to read documents.

1.) that still do not rely on query languages. In our experiment, we distinguish between querying attributes and relations. This is done by example of the task of generating frequency distributions for annotation layers.

	XMI	Mongo	Neo4j	BaseX	MySQL	Cass.
POS	1,290.2	24.9	13.8	127.6	16.3	∅
Lemma	1,287.0	27.5	57.0	185.9	14.6	∅
Morph	1,299.1	26.3	48.0	132.3	18.6	∅
Dep	1,490.2	371.4	59.4	2,349.7	87.7	∅

Table 4: Performance statistics of querying in seconds.

4.3.1. Querying Attribute Values

We query the databases for counting attributes like POS or lemma and use this data to calculate type-token ratios etc. Table 4 lists the time required by the DBMS to determine the corresponding frequency distributions. Note that Cassandra does not support count operations. The best performer is once more Neo4j, while the file system performs worst. Obviously, querying requires a DBMS.

4.3.2. Querying Relational Data

For running a performance test regarding relations (e.g., dependency relations), we reconstructed the experiment of (Fette et al., 2013). That is, all dependency relations of all words are queried governing the verb "*gehen*"/"*to walk*". Tokens of this lemma are retrieved together with all dominating or dependent tokens. Table 4 row *Dep* shows the test results: the best performer Neo4j, while BaseX is worst. Since Cassandra does not support join operations, we did not model dependency relations using this DBMS.

4.4. Combining databases

In this section we briefly describe how we combined databases to generate an MDBMS. The underlying optimization process focuses on two dimensions: runtime and memory consumption. Since the tasks described in Section 4. are independent of each other, the MDBMS is generated in such a way that it consists of databases that require a minimum runtime for the respective task. Thus, the best performing combination of databases is Neo4j (*read*,*dependency*), MySQL (*pos*,*lemma*,*morph*).

5. System Demonstration of MDBMS

In this section, we demonstrate the MDBMS as a framework for combining multiple DBMS. Each database to be integrated into the MDBMS has to be implemented as a Java interface. This interface provides functions for reading, writing and querying the respective database. To this end, we have written interfaces for all databases described in Section 3.. In order to allow for using the MDBMS in a user-friendly manner, we utilize the web interface of TextImager (Hemati et al., 2016) (see Figure 5.). TextImager

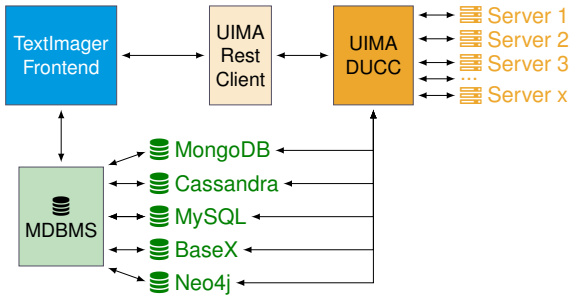

Figure 3: The MDBMS as being interfaced by TextImager.

is a UIMA-based framework that includes a wide range of NLP tools and visualization techniques for numerous languages. TextImager can upload and process multiple documents, compressed corpora and downloadable URLs according to its language-specific pipelines. All data preprocessed or output by TextImager is then stored in the MDBMS. Our system demonstration shows how TextImager uses the MDBMS to perform the tasks described in Section 4. and to visualize the corresponding results. The demonstration can be found at `http://textimager.hucompute.org/lab/database/index.html`.

6. Conclusion and Future Work

We evaluated six DBMS in the context of six database operations underlying typical working scenarios in NLP. We showed that none of the DBMS under evaluation consistently works best, but rather a combination of databases addressing different paradigms of data modeling. That is, facing the rising needs of processing big as well as divergent data, families of such divergent DBMS are the better choice. We addressed this need by means of a so-called *Multi-DataBase Management system* (MDBMS) as an interface to families of such databases. Note that our evaluation of the DBMS in question considered independent tasks. In the case of non-circularly dependent ones, the optimization can be performed by means of a multistage graph G whose nodes denote combinations of tasks and databases. In this scenario, a node x is assigned to stage Y of smallest order, so that all tasks that are required to be completed before x belong to stages preceding Y. Edges connecting nodes of different stages represent dependencies of task completion whose weight equals the runtime required by completing the task denoted by the source node by means of the respective database. This allows for solving the underlying optimization problem by means of dynamic programming for minimizing cost of possible paths within G (Sniedovich, 2010). This will be our task in future work.

7. Bibliographical References

Lou Burnard et al., editors, (2007). *TEI P5: Guidelines for Electronic Text Encoding and Interchange*, chapter A Gentle Introduction to XML. Text Encoding Initiative Consortium.

Carletta, J., Kilgour, J., O'Donnell, T., Evert, S., and Voormann, H. (2003). The NITE object model library for handling structured linguistic annotation on multimodal

data sets. In *Proc. of the EACL 2003 Workshop on Language Technology and the Semantic Web*.

Castilho, R. E. d. and Gurevych, I. (2014). A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proc. of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11.

Copeland, R. (2013). *MongoDB Applied Design Patterns: Practical Use Cases with the Leading NoSQL Database*. O'Reilly Media.

Crossley, S., McNamara, D., Baker, R., Wang, Y., Paquette, L., Barnes, T., and Bergner, Y., (2015). *Language to Completion: Success in an Educational Data Mining Massive Open Online Class*.

Dayley, B. (2014). *NoSQL with MongoDB in 24 Hours, Sams Teach Yourself*. Sams Publishing.

Eger, S. and Mehler, A. (2016). On the linearity of semantic change: Investigating meaning variation via dynamic graph models. In *Proc. of ACL 2016*.

Fette, G., Toepfer, M., and Puppe, F. (2013). Storing UIMA CASes in a relational database. In *Proc. of UIMA@GSCL*, pages 10–13.

Ganesan, K., Zhai, C., and Han, J. (2010). Opinosis: A graph-based approach to abstractive summarization of highly redundant opinions. In *Proc. of COLING '10*, pages 340–348.

Grose, T., Doney, G., and Brodsky, S. (2002). *Mastering XMI: Java Programming with XMI, XML and UML*, volume 21. John Wiley & Sons.

Hahn, U., Buyko, E., Landefeld, R., Mühlhausen, M., Poprat, M., Tomanek, K., and Wermter, J. (2008). An overview of JCoRe, the JULIE lab UIMA component repository. In *Proc. of the LREC*, volume 8, pages 1–7.

Hemati, W., Uslu, T., and Mehler, A. (2016). TextImager: a distributed UIMA-based system for NLP. In *Proc. of the COLING 2016*.

Hewitt, E. (2010). *Cassandra: The Definitive Guide*. O'Reilly Media, Inc., 1st edition.

Hinrichs, E., Hinrichs, M., and Zastrow, T. (2010). WebLicht: Web-based LRT services for German. In *Proc. of the ACL 2010*, pages 25–29.

Kim, Y., Chiu, Y., Hanaki, K., Hegde, D., and Petrov, S. (2014). Temporal analysis of language through neural language models. *CoRR*, abs/1405.3515.

Kuczera, A.). Digital editions beyond XML - graph-based digital editions. In *Proc. of the HistoInformatics 2016 co-located (DH*.

Lavrentiev, A., Stutzmann, D., and Leydier, Y. (2015). Specifying a TEI-XML based format for aligning text to image at character level. In *Symposium on Cultural Heritage Markup.*, volume 16, pages BalisageVol16–Lavrentiev01.

Leydier, Y., Églin, V., Brès, S., and Stutzmann, D. (2014). Learning-free text-image alignment for medieval manuscripts. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 363–368. IEEE.

Lyon, W. (2016). Natural language processing with graph databases and Neo4j.

Michel, J.-B., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M. K., , Pickett, J. P., Hoiberg, D., Clancy, D., Norvig, P., Orwant, J., Pinker, S., Nowak, M. A., and Aiden, E. L. (2011). Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014):176–182.

Mihalcea, R. and Tarau, P. (2004). TextRank: Bringing order into texts. In *Proc. of EMNLP-04*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Miller, J. J. (2013). Graph database applications and concepts with Neo4j. In *Proc. SAIS 2013*, page 36.

Müller, B. and Hagelstein, A. (2016). Beyond metadata–enriching life science publications in LIVIVO with semantic entities from the linked data cloud.

OpenNLP. (2010). Apache OpenNLP.

Popel, M. and Žabokrtský, Z. (2010). TectoMT: Modular NLP framework. In *Proc. of IceTAL'10*, pages 293–304.

Richardet, R., Chappelier, J., and Telefont, M. (2013). Bluima: a UIMA-based NLP toolkit for neuroscience. In *UIMA@ GSCL*, pages 34–41.

Robinson, C., Yeomans, M., Reich, J., Hulleman, C., and Gehlbach, H. (2016). Forecasting student achievement in MOOCs with natural language processing. In *Proc. of the LAK '16*, LAK '16, pages 383–387, New York, NY, USA. ACM.

Rousseau, F., Kiagias, E., and Vazirgiannis, M. (2015). Text categorization as a graph classification problem. In *Proc. ACL 2015*, pages 1702–1712.

Sniedovich, M. (2010). *Dynamic programming: foundations and principles*. CRC press.

Tixier, A., Malliaros, F., and Vazirgiannis, M. (2016). A graph degeneracy-based approach to keyword extraction. In *Proc. of EMNLP*.

Usbeck, R., Ngomo, A. N., Röder, M., Gerber, D., Coelho, S., Auer, S., and Both, A. (2014). AGDISTIS-graph-based disambiguation of named entities using linked data. In *ISWC 2014*, pages 457–471.