A Demonstration of TextDB: Declarative and Scalable Text Analytics on Large Data Sets

Zuozhi Wang, Flavio Bayer, Seungjin Lee, Kishore Narendran, Xuxi Pan, Qing Tang, Jimmy Wang, Chen Li
UC Irvine, CA 92697, USA
{zuozhiw, fbayer, seungl13, narendrk, xuxip, qingt, jianwenw, chenli}@uci.edu

***Abstract*—We are developing TextDB, an open-source data-management system that supports text-centric operations in a declarative and efficient way using an algebraic approach as in relational DBMS. In this demonstration, we show scenarios where we can use TextDB to perform powerful information extraction easily and efficiently on text documents.**

Video: https://github.com/TextDB/textdb/wiki/Video

I. Introduction

We are living in an era where a large amount of information is recorded as text. Text data needs to be stored, indexed, and managed to support advanced analytics and knowledge discovery. Some of the tasks are performed by search engines, as they support efficient keyword-based search on text, and some tasks are managed by relational databases, as they provide a query-processing engine and a declarative language. Many text-centric applications rely on different components by combining them with ad-hoc scripts and programs, which require a significant amount of programming efforts and are also inflexible. To overcome these limitations, we are developing TextDB[1], an open-source data-management system to store, index, and query large amounts of text data in a declarative and efficient way using an algebraic approach as in relational DBMS. A family of text-related operations, such as keyword search, regular expression match, and named entity extraction, are supported natively by the system so that end users need not write the same kind of operations repeatedly. The system also provides a user-friendly GUI to let users easily formulate a query.

Related Work: Relational DBMS systems are mainly optimized for storing and querying data types such as integer, float, and string, but they have limit support for the text type, which is not treated as a first-class citizen. Open-source search engines, such as Lucene, have a main focus of keyword search. A fundamental operator, namely join, is missing in these engines. There are many open-source systems specifically developed for text analytics. For example, Stanford NLP supports statistical NLP, deep learning NLP, and rule-based NLP tools for major computational linguistic problems. TextDB is different from the systems aforementioned by: 1) providing storage and indexing capabilities as in a database, 2) supporting built-in text-specific operators such as keyword matcher, or operators provided by incorporating other packages, and 3) the ability to connect operators to make a more expressive plan, and 4) a user-friendly interface to formulate a query easily. The IBM SystemT [1] is a rule-based information extraction system that takes a database-like algebraic approach. It provides a declarative language called *AQL* to formulate queries. TextDB differs from SystemT in TextDB's storage and indexing capability, as well as extensibility of incorporating other packages. SystemT is a commercial product, while TextDB is open-source.

[1]https://github.com/TextDB/textdb

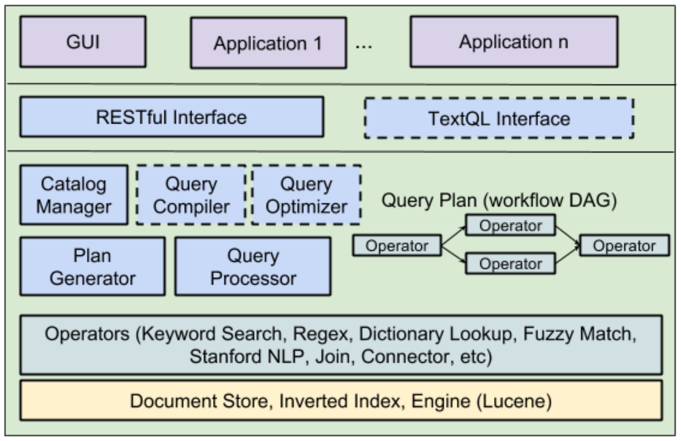
GUI
Application 1
...
Application n
RESTful Interface
TextQL Interface
Catalog Manager
Query Compiler
Query Optimizer
Query Plan (workflow DAG)
Operator
Operator
Operator
Operator
Plan Generator
Query Processor
Operators (Keyword Search, Regex, Dictionary Lookup, Fuzzy Match, Stanford NLP, Join, Connector, etc)
Document Store, Inverted Index, Engine (Lucene)

Fig. 1. TextDB system architecture. Modules with dotted lines are under development.

II. System Overview

Figure 1 shows the architecture of TextDB. Similar to a relational DBMS, it includes a persistent storage layer for storing and indexing text data, a rich set of text-specific operators, and a plan generator to construct query plans, which can be executed by a run-time engine. Data is modeled as a collection of records, with each record containing a few fixed-type attributes. Each attribute can be text, as well as other data types such as integer, float, and date. Operators are pull-based iterators, providing a `getNextTuple()` function and a `getOutputSchema()` function, so that they can be connected to each other to form a directed acyclic graph (DAG) as a query plan.

TextDB includes relational operators such as `select` and `project`. It also provides several text-specific operators specifically for information extraction [2], such as keyword matching, regular expressions, and named entities. These operators return matching results as a list of spans. Each span is a pair `<begin, end>` that represents a text region from the beginning position to the ending position. The system has a join operator that can match tuples based on conditions over spans, such as "two spans are within a certain character or token distance."

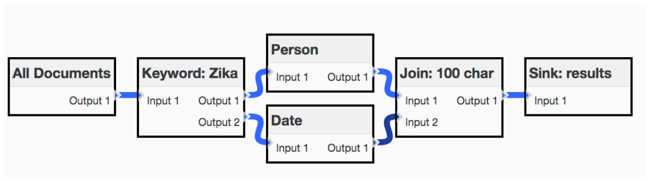

Fig. 2. Demo 1: Extracting Zika cases.

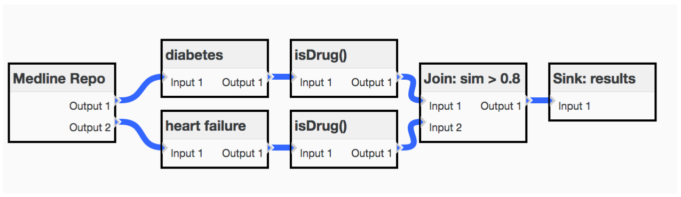

Fig. 3. Demo 2: Extracting drug information

As a data-management system, it is critical to provide a storage layer to keep information reliably. TextDB uses Lucene, an open-source package for keyword search, as its storage layer. Lucene's 18 years of active development and its large community have made it very stable. We decided not to implement the storage layer from scratch, considering that it would require a significant amount of engineering effort. We also did not use a storage module of an existing relational DBMS, due to the fact that their storage is not optimized for text and is hard to be detached. This architectural decision allows us to focus on new open-ended research topics. At the same time, we also need to deal with related challenges due to the limited interface of Lucene.

TextDB includes a declarative language (under development), which is an extension of the commonly used SQL language for easy use by developers. A compiler and an optimizer convert a query to a plan to be executed by the engine. Using a declarative method, users can fully focus on query formulation and leave the physical implementation behind. TextDB can also run as a web service with a RESTful API. In addition, since it is not easy for users (especially non-experts) to formulate a query to express their needs, TextDB provides a user-friendly Web-based GUI to let users easily formulate a query by dragging and dropping predefined elements such as a regular expression for emails and a dictionary of cities.

III. Demonstration Scenarios

We will use two scenarios to demonstrate the system.

A. *Case Study 1: Extracting Zika Case Reports*

Consider a case where a public health researcher wants to investigate the recent Zika disease outbreaks by finding case reports from various websites, such as CDC reports and news articles. Specific case reports can help the researcher determine the severity of the outbreak. She can also use these cases to warn the public when traveling to dangerous places. In order to find case reports, she could search "zika" on a search engine such as Google or Bing, and manually find cases from the search results. This process can be time consuming and error prone. With the help of TextDB, she can find more results with less effort by constructing a query plan using the GUI interface, as shown in Figure 2.

The query uses various operators to accomplish the extraction task. The query relies on an assumption that, within articles containing the keyword "zika", if a person and a date appear closely together, then there is a high chance that the article has a Zika case report. The extraction plan first filters the documents containing the "zika" keyword using a Keyword Matcher. Then it extracts person-related information by using a pre-defined extractor, which can be implemented by a Name-Entity operator. Date-related information is extracted by a pre-defined extractor too, which can be implemented by a regular expression. Finally, the results from "Person" and "Date" are joined based on the condition that their distance is smaller than 100 characters.

B. *Case Study 2: Discovering Drug Information*

A biomedical researcher wants to discover a pair of articles (r, s) in a large medical literature repository such as MEDLINE. In one article r, a drug is verified to be effective to diabetes, while in another article s, the same (or similar) kind of drugs were discovered to cause heart failure. This piece of information is very important for a doctor when deciding the medication for patients who have both diseases. Using existing search engines, one has to retrieve publications containing keyword diabetes and those containing keywords heart failure, and then identify similar drugs mentioned in these publications. Again, this approach is doable, but can be very costly and challenging, especially for non-experts. Using TextDB, she can formulate a declarative query on a GUI to find a set of potential answers, as shown in Figure 3. In this query, documents containing "diabetes" and "hearth failure" are selected respectively from the repository. The extractor `isDrug()` finds drugs in a document, and can be implemented using regular expressions. The join operator with a predicate `sim > 0.8` specifies a similarity threshold that allows the drug names extracted from the previous step to be slightly different.

In the two scenarios, both queries use relational algebra to select and join multiple pieces of text to improve search efficiency and quality. In each of them, we will show how to use the GUI to formulate a query, submit the query to the backend engine, and display the results on the frontend interface. We will explain the design and implementation of each operator. We will show the benefits of the system by allowing the user to dynamically modify the operators (such as adding or deleting entities in a dictionary or a regular expression) and re-run the query. We will show its NLP capabilities using the incorporated Stanford NLP package. We will also show the high efficiency of the system on large data sets due to its internal storage and indexing.

References

[1] Y. Li, F. Reiss, and L. Chiticariu, "SystemT: A declarative information extraction system," in *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA - System Demonstrations*, 2011, pp. 109–114.

[2] R. Grishman, "Information extraction: Capabilities and challenges," Notes prepared for the 2012 International Winter School in Language and Speech Technologies, Jan 2012.