ALASKA for Ontology Based Data Access

Jean-François Baget, Madalina Croitoru, Bruno Paiva Lima da Silva

LIRMM (University of Montpellier II & CNRS), INRIA Sophia-Antipolis, France

Abstract. Choosing the tools for the management of large and semi-structured knowledge bases has always been considered as a quite crafty task. This is due to the emergence of different solutions in a short period of time, and also to the lack of benchmarking available solutions. In this paper, we use ALASKA, a logical framework, that enables the comparison of different storage solutions at the same logical level. ALASKA translates different data representation languages such as relational databases, graph structures or RDF triples into logics. We use the platform to load semi-structured knowledge bases, store, and perform conjunctive queries over relational and non-relational storage systems.

1 Motivation and Impact

The ONTOLOGY-BASED DATA ACCESS (ODBA) problem [4] takes a set of facts, an ontology and a conjunctive query and aims to find if there is an answer / all the answers to the query in the facts (eventually enriched by the ontology). Several languages have been proposed in the literature where the language expressiveness / tractability trade-off is justified by the needs of given applications. In description logics, the need to answer conjunctive queries has led to the definition and study of less expressive languages, such as the $\mathcal{EL}$ ([1]) and DL-Lite families [2]. Properties of these languages were used to define profiles of the Semantic Web OWL 2 language (`www.w3.org/TR/owl-overview`).

When the above languages are used by real world application, they are encoded in different data structures (e.g. relational databases, Triple Stores, graph structures). Justification for data structure choice include (1) storage speed (important for enriching the facts with the ontology) and (2) query efficiency. Therefore, deciding on what data structure is best for one's application is a tedious task. While storing RDF(S) has been investigated from a database inspired structure [3], other logical languages did not have the same privilege. Even RDF(S), often seen as a *graph*, has not been thoroughly investigated from a *ODBA perspective* wrt *graph structures* and emergence of *graph databases* in the NoSQL world.

This demo will allow to answer the following research question: "*How to design an unifying logic-based architecture for ontology-based data access?*".

2 ALASKA

We thus demonstrate the ALASKA (acronym stands for **A**bstract and **L**ogic-based **A**rchitecture for **S**torage systems and **K**nowledge bases **A**nalysis) platform. ALASKA's goal is to enable and perform ODBA in a logical, generic manner, over existing, heterogenous storage systems. The platform architecture is multi-layered.

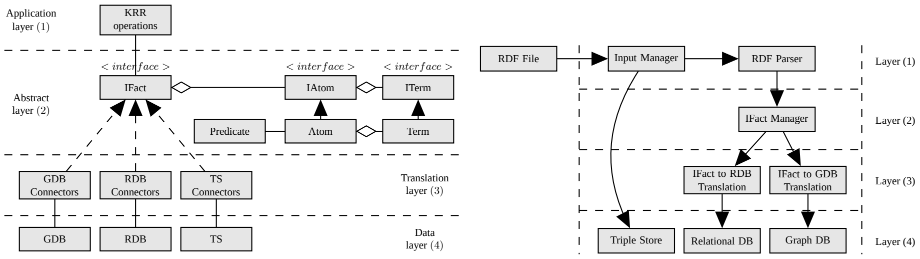


Fig. 1. ALASKA class diagram, and workflow of storage protocol.

The first layer is (1) the **application** layer. Programs in this layer use data structures and call methods defined in the (2) **abstract** layer. Under the abstract layer, the (3) **translation** layer contains functions by which logical expressions are translated into the languages of several storage systems. Those systems, when connected to the rest of the architecture, compose the (4) **data** layer. Performing higher level reasoning operations within this architecture consists of writing programs and functions that use exclusively the formalism defined in the abstract layer. Once this is done, every program becomes compatible to any storage system connected to architecture.

To have a functional architecture, representative storage systems were selected. The systems already connected to ALASKA are listed below (please note that this list is not final and subject to constant updates):

→ **Relational databases:** Sqlite 3.3.6[1], MySQL 5.0.77[2]

→ **Graph databases:** Neo4J 1.8.1[3], DEX 4.7[4], OrientDB 1.0rc6[5], HyperGraphDB 1.1[6]

→ **Triples Stores:** Jena TDB 0.9.4[7]

Figure 1 displays, on the left-hand side, the class diagram of the architecture. On the right-hand side the workflow of knowledge base storing is illustrated. Let us analyse the workflow. We consider a RDF file as input. The RDF file is passed to the Input Manager (layer 1). According to the storage system needs the Input Manager directs it accordingly. If the RDF file will be stored in a Triple Store than the file is directly passed to the Triple Store of choice (layer 4). If the RDF file needs to be stored in a graph database the file is first transformed in an IFact object (layer 2). It is then translated (layer 3) to the language of the system of choice (graph database in this case) before being stored onto disk (layer 4).

Querying in ALASKA follows a similar workflow as the storage. In Figure 2, on the left hand side we show the storing workflow for storing a fact F in either a relational database or a graph database (for simplification reasons). On the right hand side of

[1] http://www.sqlite.org/

[2] http://www.mysql.com/

[3] http://www.neo4j.org/

[4] http://www.sparsity-technologies.com/dex

[5] http://www.orientechnologies.com/orient-db.htm

[6] http://www.hypergraphdb.org/

[7] http://jena.sourceforge.net/

the figure the querying workflow is depicted for graph and relational databases. Let us consider a fact F both stored in a relational database and in a graph database. Let us also consider a query Q. This query can either be expressed in SQL (or in a graph language of choice) and be sent directly to the respective storage system (e.g. the SQL Q query to the F in the relational database). Alternatively, the query can be translated in the abstract logic language and a generic backtrack algorithm used for answering Q in F. This generic backtrack algorithm will solely use the native language "elementary" operations for accessing data.

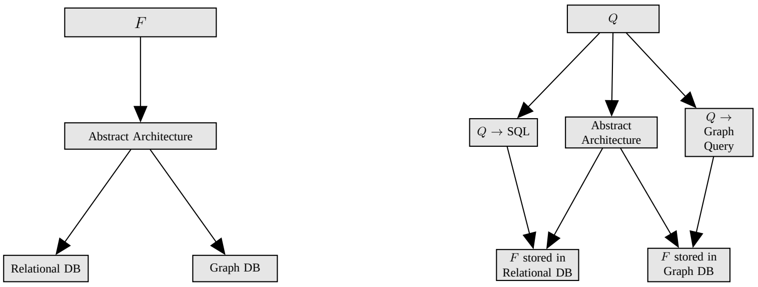


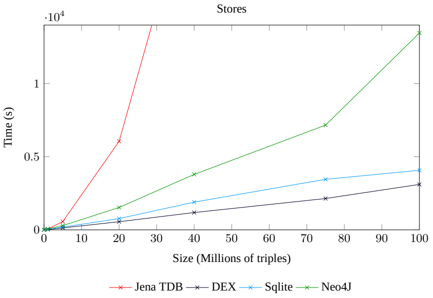
Fig. 2. ALASKA storage and querying workflow.

3 ALASKA Demo Procedure

In a nutshell, the demo procedure of ALASKA goes as follows. Given a knowledge base (user provided or selected amongst benchmarks provided by ALASKA) a set of storage systems of interest are selected by the user. The knowledge base is then transformed in the Abstract Architecture and consequently stored in the selected systems. The storage time per system is then showed to the user (excluding the time needed for translation into Abstract Layer). Once the storage step is finished, users are able to perform conjunctive queries over the knowledge bases and, once again, compare the time of each system for query answering.

Let us consider an example. The knowledge base used here has been introduced by the SP2B project [5]. The SP2B project supplies a generator that creates knowledge bases with a certain parametrised quantity of triples maintaining a similar structure to the original DBLP knowledge base. The generator was used to create 5 knowledge bases of increasing sizes (5 million triples, 20, 40, 75 and respectively 100). Each of the knowledge bases has been stored in Jena, DEX, SQLite and Neo4J. In Figure 3 we show the time for storing the knowledge bases and their respective sizes on disk.

The user can see that the behavior of Jena is worse than the other storage systems. This is due to the Jena RDF parser uses central memory for buffering purposes when loading a file. For comparison, the other systems use the custom made RDF parser of ALASKA. Let us also note that DEX behaves much better than Neo4J and this is due to the fact that ACID transactions are not required for DEX (while being respected by Neo4J). Second, the size of storage is also available to the user. One can see, for instance, that the size of the knowledge base stored in DEX and Neo4J is well under the size of initial RDF file. However, the size of the file stored in Jena is bigger than the one stored in SQLite and bigger than the initial size of the RDF file.



Size of the stored knowledge bases					
System	5M	20M	40M	75M	100M
DEX	55 Mb	214.2 Mb	421.7 Mb	785.1 Mb	1.0 Gb
Neo4J	157.4 Mb	629.5 Mb	1.2 Gb	2.3 Gb	3.1 Gb
Sqlite	767.4 Mb	2.9 Gb	6.0 Gb	11.6 Gb	15.5 Gb
Jena TDB	1.1 Gb	3.9 Gb	7.5 Gb	-	-
RDF File	533.2 Mb	2.1 Gb	4.2 Gb	7.8 Gb	10.4 Gb

Fig. 3. Storage time and KB sizes in different systems

Once the storage step is finished, users are able to perform conjunctive queries. As already explained, querying the newly-stored knowledge base using the native interrogation engine (SQL for relational databases, SPARQL for 3Stores, etc.) is still possible with ALASKA. However, ALASKA also allows the possibility to perform conjunctive queries that access any storage system included in the platform using the same backtrack algorithm. The queries we have used here are:

1. `type(X,Article)`
   Returns all the elements which are of type article.
2. `creator(X,PaulErdoes) . creator(X,Y)`
   Returns the persons and the papers that were written with Paul Erdoes.
3. `type(X,Article) . journal(X,Journal1-1940) . creator(X,Y)`
   Returns the creators of all the elements that are articles and were published in Journal 1 (1940).
4. `type(X,Article) . creator(X,PaulErdoes)`
   Returns all the articles created by Paul Erdoes.

In the graphs in Figure 4 and 5 we show the combination storage and querying algorithm. For instance Jena(BT) stands for using Jena for elemntary access operations and the generic backtrack for querying. SQLite(SQL) uses directly the SQL querying engine over the data stored in SQLite. In the graph corresponding to Q1 we also study the behavior of SQLite using the genric backtrack. For other queries we did not show it because the behavior is much worse that the other systems. We can also observe that for Q1, Q3 and Q4 queries SQLite and Jena behave faster than the graph bases. However, for Q2 this is no longer the case. In this case the fastest system for the generic backtrack is Jena followed by Neo4J and DEX, while SQLite explodes. The intuition behind this behavior is due to the phase transition phenomenon in relational databases but these aspects are out of the scope of this demonstration.

4 Discussion

An abstract platform (ALASKA) was created in order to perform storage operations independently of the data location. In order to enable the comparison between different

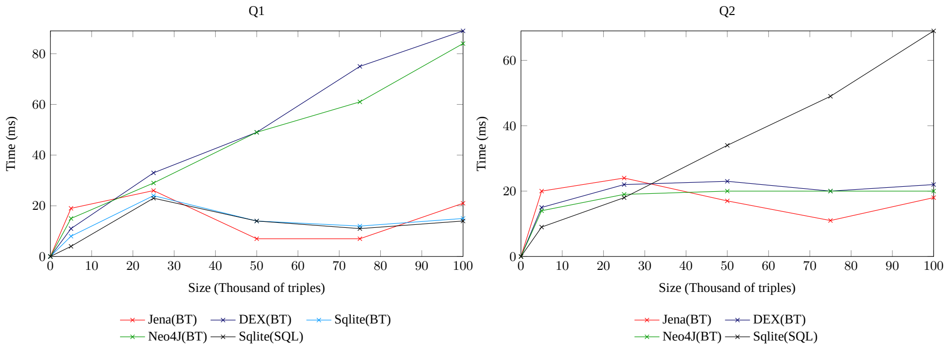


Fig. 4. Querying performance using ALASKA for large knowledge bases: Q1 and Q2

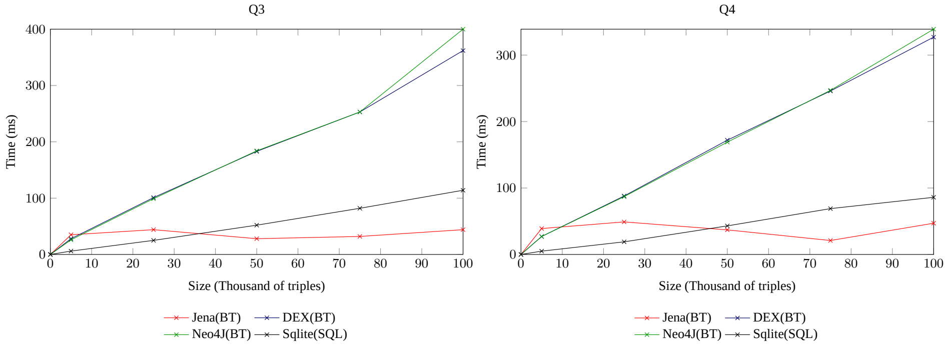


Fig. 5. Querying performance using ALASKA for large knowledge bases: Q3 and Q4

storage paradigms, ALASKA has to translate a knowledge base from a common language (i.e. First Order Logic) into different other representation language. Comparing different storage and querying paradigms becomes then possible. A knowledge base stored in a relational database can be also stored in a graph based database as well as a Triple Store and queried with an in built SPARQL engine etc.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope. In *Proc. of IJCAI 2005*, 2005.
2. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
3. B. Haslhofer, E. M. Roochi, B. Schandl, and S. Zander. Europeana RDF store report. Technical report, University of Vienna, Vienna, Mar. 2011.
4. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, 2002.
5. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. Sp2bench: A sparql performance benchmark. *CoRR*, abs/0806.4627, 2008.