Scalable Reasoning Infrastructure for Large Scale Knowledge Bases

Hima Karanam, Sumit Neelam, Udit Sharma, Sumit Bhatia, Srikanta Bedathur, L. Venkata Subramaniam, Maria Chang, Achille Fokoue-Nkoutche, Spyros Kotoulas, Bassem Makni, Mariano Rodriguez Muro, Ryan Musa, Michael Witbrock

IBM Research AI
hkaranam@in.ibm.com

1 Introduction

Knowledge Bases, supported by domain and common-sense ontologies, are critical in various AI-centric applications, and are beginning to be integrated as knowledge sources in primarily neural AI research. As information extraction technologies mature, we are seeing the growth of automatically extracted KGs, reaching millions of entities and billions of relations between them. Such KGs are often linked to various upper level ontologies and additional resources that makes it possible to infer sophisticated new, hitherto unknown knowledge from them. Although full-scale First-Order Logic (FOL) reasoning is not widely supported by reasoners, standards such as OWL2 are being increasingly employed to design enterprise systems that can compute limited but useful entailments over sophisticated, large ontologies. Moreover, in AI applications, and particularly applications along a path to general intelligence, hard guarantees such as decidability are of little importance, compared with practical effectiveness in producing correct, useful, explainable inferences; in these cases, expressivity, including the ability to express background knowledge that is not readily and practically expressible in FOL, becomes increasingly important. In this context, the commonly used state-of-the-art reasoners that can operate over knowledge graphs can be viewed as having been restricted in three main directions:

Scalability of these reasoners is quite limited; they support relatively small ontologies and KGs. For instance, there are no available open source reasoners that can exploit big data frameworks such as Spark in order to scale expensive computations common during reasoning tasks, or use flexible graph data management systems built on horizontally scalable platforms such as JanusGraph.

Expressivity of the languages for which effective inference can be performed is low; the majority of reasoners do not support general rules, let alone spatial and temporal reasoning, contextual reasoning, modal and second-order reasoning. These features are implicitly ubiquitous in text encoding human knowledge, and necessary for symbolic and semi-symbolic AI research.

Modelling of complex multi-modal data including by representing uncertainty and imprecision (common during automatic extractions of KGs), and supporting reasoning over them is difficult. Further, in applications where KBs are used to aid natural language generation or understanding, we need ways to link entities/relationships and rules with

their cross-modal counterparts (e.g., natural language support sentences, images, or neural network embeddings), or perhaps use text and images directly in inference with bounded error and suitably expressed, learned, inference "rule" functions.

We discuss our ongoing efforts towards building a software toolkit to support scalable reasoning over large knowledge bases targeted towards industrial applications. We describe our initial ongoing efforts and some preliminary results and discuss the directions for future work.

2 System Architecture

The high level architecture of the proposed system is describe in Figure 1 and we describe the different components in detail below.

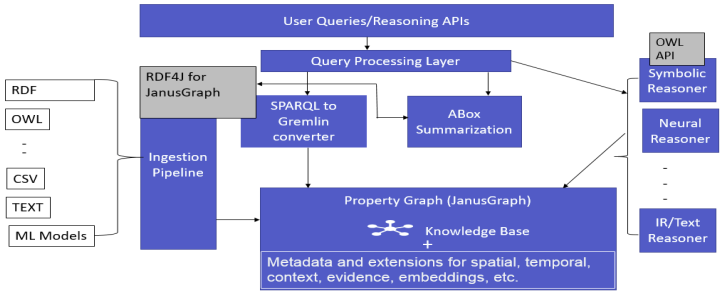

Fig. 1: System Architecture

Ingestion Pipeline and Storage Layer: We chose JanusGraph (a property graph) as our default storage technology to store knowledge base along with additional metadata such as context, evidence etc. Since property graphs can easily capture the context/confidence etc in the form of node or edge properties, it becomes easier to map these information coming from the data curation pipelines easily at one place. During ingestion, user data in various commonly used formats is converted and stored in the back-end storage. In order to ingest OWL/RDF, data we have extended RDF4J framework to add JanusGraph as additional store implementing data store interfaces. This takes RDF/OWL data in various forms and maps it to appropriate property graph model and creates required indices as well.

Query Processing Unit processes the user queries and interprets them to decide the appropriate modules required to answer the query and interacts with various reasoners and backend store to retrieve the final answers. For example in case of symbolic reasoner, this unit will talk to symbolic reasoner using OWL APIs to get the TBox inference to do the query expansion or query rewrite. In cases where complex reasoning is needed, it sends the required ABox summary information to the reasoner to generate the answer. We are planning to implement the ABox summary builder described in the SHER system [2] to work with JanusGraph backend. We have extended RDF4J query interfaces for basic RDF querying.

Reasoners: This component is responsible for standing up different reasoners with appropriate APIs to talk to the query and storage layers. In the current implementation, we are integrating Racer [3] using OWL APIs to support various OWL profiles. We will be extending this to include neural reasoner and default IR/Text reasoner to get a search answer in scenarios where we don't get confident answers from other reasoners.

3 Graph Database for Reasoning

The choice of storage technology is crucial for efficient implementation of underlying reasoning engines. While the in-memory reasoners maintain complete ABox data in memory leading to quick query times, systems designed for dealing large ABoxes store data in persistent storage backends such as relational database or triple stores. At query time, the required data is fetched and passed on to the reasoner for inference.

We adopt a property graph database as our persistent ABox store. Graph databases have become quite popular for data storage due to their flexible schema and close relation of the storage model with real-world data. Such graph databases store data in the form of nodes and edges, provide index-free adjacency, and therefore, perform potentially better on linked datasets. Compared to relational databases, graph databases are better at join intensive queries [7]. Further, triple stores being strongly index based systems are good when relationship exploration is not very deep whereas graph databases are good at finding complex relations, and their compact representation helps in to speedup reasoning tasks by storing inferred information.

4 Query Engine

In the current implementation of our system, the query engine is built using three basic modules for handling symbolic reasoning. Here, we take SPARQL queries and convert them to appropriate Gremlin or reasoning queries to get the final answer.

Query Expansion: This module is responsible for expanding the input query as a union of multiple conjunctive queries in order to find some of the implicit solutions to the query. Query expansion is guaranteed to find all the solutions, when the underlying knowledge base follows DL-lite logic.

SPARQL to Gremlin Translation: Query generated by query expansion module is transformed to the gremlin traversal so that it can be executed directly over the Janus-Graph. Gremlin supports both declarative and imperative queries whereas SPARQL only supports declarative queries. Transforming SPARQL queries into declarative style gremlin queries offer two advantages: *(i)* declarative gremlin queries are formulated using the Match step which consists of one or more traversal patterns similar to triples in a SPARQL query. So, SPARQL triples can be directly transformed to Gremlin patterns using the rule-based mapping [6]; and *(ii)* declarative gremlin queries use a Match Algorithm that sorts different patterns present in the query according to their filtering capabilities, so as to optimize the query execution. Graph meta-data generated during data ingestion phase can be used to differentiate between edge traversals or property traversals at the time of query translation.

Consistency Checker: Any reasoning query over the expressive logic can be reduced to

that of determining consistency of underlying knowledge base [5]. Negated assertion of a query is added to the knowledge base before performing the consistency test. Different versions of tableau algorithms [1] can be used for performing consistency detection depending upon the expressiveness of the underlying knowledge base.

5 Results and Discussions

We report initial results achieved for RDF to Property graph conversion using JanusGraph for LUBM dataset[1] for 100 universities. We compare query execution times for Virtuoso system, manually and automatically generated gremlin queries against JanusGraph for LUBM SPARQL benchmark queries. As note from Table 1, our proposed architecture achieves better performance for most of the queries, except in cases where large number of node scans are required for producing the output because vertices are spread on disk physically. For these experiments, we have used basic schema transformation described in [4] without much focusing on locality of reference. We are currently studying efficient schema transformation also which takes care of this issues. In addition, we are exploring how to integrate the standard logical reasoners with other approximate reasoners based on neural models that could act as a fall-back mechanism in cases where the required information is not present in the knowledge bases.

Table 1: Query execution time for 100U-LUBM dataset (in milliseconds)

Query	Virtuoso	JanusGraph Manual	JanusGraph Auto	Output Size	Query	Virtuoso	JanusGraph Manual	JanusGraph Auto	Output Size
Q1	68	3.6	9	4	Q8	262	822	898	7790
Q2	1006	220495	434692	264	Q9	1958	856103	867483	27247
Q3	55	1.5	2	6	Q10	61	1.3	0.9	4
Q4	79	5.3	26	34	Q11	55	17.8	15	224
Q5	63	58.9	66	719	Q12	51	2.5	27	15
Q6	25040	568078	522321	1048532	Q13	56	33.8	38	473
Q7	71	6.5	5	67	Q14	18585	443954	405736	795970

References

1. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica **69**(1), 5–40 (2001)
2. Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., Srinivas, K.: Scalable highly expressive reasoner (sher). Web Semantics: Science, Services and Agents on the World Wide Web **7**(4), 357–361 (2009)
3. Haarslev, V., Möller, R.: Racer system description. In: International Joint Conference on Automated Reasoning. pp. 701–705. Springer (2001)
4. Hartig, O.: Reconciliation of rdf* and property graphs. arXiv preprint arXiv:1409.3288 (2014)
5. Horrocks, I., Tessaris, S.: Querying the semantic web: a formal approach. In: International Semantic Web Conference. pp. 177–191. Springer (2002)
6. Thakkar, H., Punjani, D., Keswani, Y., Lehmann, J., Auer, S.: A stitch in time saves nine–sparql querying of property graphs using gremlin traversals. arXiv preprint arXiv:1801.02911 (2018)
7. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D.: A comparison of a graph database and a relational database: a data provenance perspective. In: Proceedings of the 48th annual Southeast regional conference. p. 42. ACM (2010)

[1] http://swat.cse.lehigh.edu/projects/lubm/