Chimera: A Bridge Between Big Data Analytics and Semantic Technologies

Matteo Belcao[1(✉)], Emanuele Falzone[1(✉)], Enea Bionda[2(✉)], and Emanuele Della Valle[1(✉)]

[1] Politecnico di Milano, DEIB, Milan, Italy
matteo.belcao@mail.polimi.it,
{emanuele.falzone,emanuele.dellavalle}@polimi.it
[2] Ricerca sul Sistema Energetico - RSE S.p.A., Milan, Italy
enea.bionda@rse-web.it

Abstract. In the last decades, Knowledge Graph (KG) empowered analytics have been used to extract advanced insights from data. Several companies integrated legacy relational databases with semantic technologies using Ontology-Based Data Access (OBDA). In practice, this approach enables the analysts to write SPARQL queries both over KGs and SQL relational data sources by making transparent most of the implementation details. However, the volume of data is continuously increasing, and a growing number of companies are adopting distributed storage platforms and distributed computing engines. There is a gap between big data and semantic technologies. Ontop, one of the reference OBDA systems, is limited to legacy relational databases, and the compatibility with the big data analytics engine Apache Spark is still missing. This paper introduces Chimera, an open-source software suite that aims at filling such a gap. Chimera enables a new type of *round-tripping* data science pipelines. Data Scientists can query data stored in a data lake using SPARQL through Ontop and SparkSQL while saving the semantic results of such analysis back in the data lake. This new type of pipelines semantically enriches data from Spark before saving them back.

Keywords: Ontology based data access · Semantic technologies · Big data · Apache spark · Knowledge graph · Analytics

1 Introduction

The fast growth of the analytic sector of these years and the exponential increment of the data volumes lead to the massive adoption of new large-scale analytics engines to store and process large volumes of relational data. At the same time, semantic reasoning on domain ontologies allowed for gathering advanced insights on heterogeneous data.

Until nowadays, these two worlds were totally separated. As depicted in Fig. 1, the Knowledge Engineers (KEs) use the Web Ontology Language (OWL) to design Knowledge Graphs (KGs), which capture the knowledge of the domain experts, and store them in RDF Repositories. On the contrary, Data Engineers

A. Hotho et al. (Eds.): ISWC 2021, LNCS 12922, pp. 463–479, 2021.
https://doi.org/10.1007/978-3-030-88361-4_27

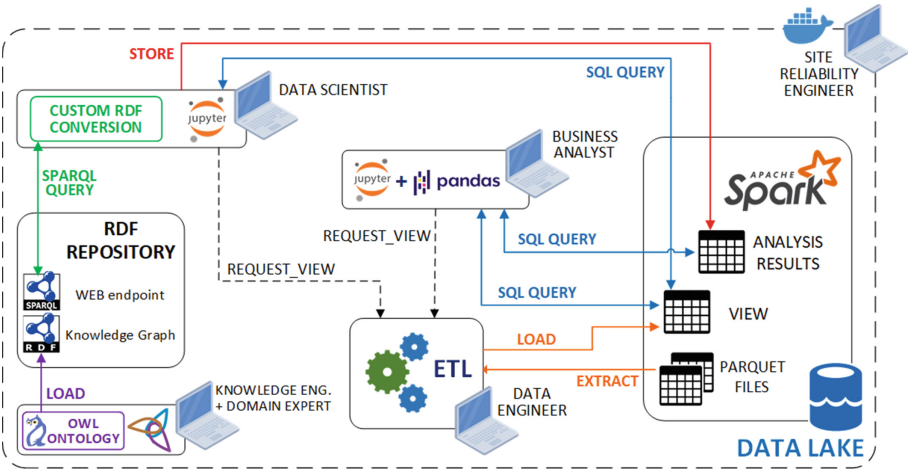

Fig. 1. The classical analytical pipeline involving manual ETL operations and custom SPARQL integration.

(DEs) ingest and store relational data in data lakes and create and maintain ETLs for Business Analysts (BAs) and Data Scientists (DS). DSs, who want to perform KG-empowered analytics, have to combine manually 1. semantic data, coming from SPARQL queries evaluated over KGs, and 2. relational data, coming from SQL queries performed over the data lake. The BAs need to read and analyze both the raw data in the data lake and DSs' analysis results. Furthermore, the Site Reliability Engineers (SREs) experience difficulty maintaining infrastructure efficiency, ensuring performance and scalability.

This scenario resembles several problems. First of all, the ETL tasks are problem-dependent: whenever a DS or a BA needs to perform a new analytical query, they ask DEs for a new ETL. Each ETL may require several days of work and many meetings between DEs and analysts. Moreover, the DSs have to persist their analysis results in the data lake to make them available for the BAs. Furthermore, only a combination of multiple tools can achieve such a result.

In the past decades, the scientific community focused on researching new methodologies to combine the advantages of relational databases and the reasoning capabilities offered by ontologies. The research effort led to the definition of the Ontology-Based Data Access (OBDA) paradigm [2,20]. OBDA offers users a conceptual layer that abstracts specific aspects related to the data source using a convenient query vocabulary. In this way, the OBDA paradigm lowers the complexity of data analysis tasks. It enables domain experts to create analytical queries without the need for advanced SQL knowledge or data professionals' help. DSs and BAs can write queries over an ontology that makes transparent the task of retrieving data from several tables and joining them.

Nowadays, among the OBDA systems [4,13,17] Ontop [3] is one of the first to be offered as a commercial solution[1]. Ontop's current scope is relational

[1] Ontop project: https://ontop-vkg.org, Ontopic company: https://ontopic.biz/.

databases. Several real-world users cannot benefit from Ontop to integrate data stored in data lakes that engines such as Apache Spark [23] can process.

The scientific literature documents several attempts to integrate Ontop in real large-scale analytics projects. Statoil [8] and Siemens Energy [9] implemented OBDA solutions for performing analytical queries in big data scenarios by using custom implementations of Ontop. Moreover, the University of New South Wales [22] developed a system for inferring diabetes on new potential patients using reasoning over their EHRs (E-Health Records) accessed via Ontop. However, a reusable framework is still missing.

In this paper, we present Chimera[2], a software suite that aims at better connecting the big data world with semantic technologies. It provides components for enabling KG-empowered analytics to scale to big data technologies, using Apache Spark [23] as a query processing engine for accessing the data stored in a data lake. The Chimera suite is composed of: 1. Ontop$_{Spark}$, an extended version of Ontop that allows formulating SPARQL queries over an Apache Spark cluster, and 2. `PySPARQL`, a python library that can materialize a SPARQL response as a Spark DataFrame or a GraphFrame.

Chimera introduces new possibilities for implementing *round-trip* data analysis pipelines. Those pipelines query the data lake with SPARQL to get a semantically enriched response by leveraging Ontop and materialize the responses as new Spark tables to query in another iteration (namely, *trip*). Notably, *round-trip* pipelines require to combine both Ontop$_{Spark}$ and `PySPARQL`. As a result, companies that already have Apache Spark as an analytical engine and would like to use ontologies to improve their analysis, but complain about the complexity of managing the integration of several tools, can integrate semantic technologies into their business processes.

The paper is structured as follows. Section 2 shows the most adopted OBDA technologies and their advancements in supporting big data technologies. Section 3 formalizes the requirements that we elicited, while Sect. 4 illustrates Chimera. Section 5 describes a running example that showcases the components' capabilities in an IoT scenario, and Sect. 6 illustrates a real industrial deployment of Chimera. Section 7 discusses related work. Finally, Sect. 8 concludes, discussing future developments and maintenance plans.

2 Ontology Based Data Access

The OBDA paradigm enables creating analytical SPARQL queries based on data physically stored in relational formats by writing queries over an ontology that makes transparent the task of retrieving data from several SQL tables.

The most successful OBDA systems are Mastro [4], UltraWrap [17], MorphRDB [13], and Ontop. All of them support the connection over JDBC to legacy relational databases. Ontop and Mastro are also available as Protégé plugins.

Specifically, Ontop [3] is one of the first to be offered as a commercial solution. It enables to perform data integration by exposing relational databases as Virtual

[2] https://chimera-suite.github.io/.

Knowledge Graph (VKG) [21]. The VKG mechanism allows creating an RDF representation of relational data without allocating additional space. All the data remains in its original format and location. This approach enables Ontop to avoid triple materialization. It answers SPARQL queries by directly translating them into SQL queries over the data sources However, VKGs support is limited to ontologies expressed in OWL2QL. This expressivity does not forbid users to use more expressive OWL2 profiles. Users interested in doing so shall use another reasoner (e.g., Hermit) to materialize T-box and A-box inference. And they shall store the resulting KG in an RDF repository that supports SPARQL 1.1 Federation Queries[3]. In this way, the RDF repository SPARQL endpoint can answer queries that refer to the KG and the relational data accessed via OBDA using the VKG approach. As we will explain, Chimera requires identifying a subset of the KG's T-Box to use in Ontop as a *DB-descriptive ontology*.

Ontop supports all the W3C standards related to OBDA, and comes in different distributions. The most popular are the Protégé extension and the *Ontop-CLI*, which offers both a web and a terminal interface.

To perform the VKG translations, the Ontop engine needs a set of user-defined RDF-to-SQL mappings expressed in R2RML or in the Ontop mapping language [3]. R2RML is a W3C standard language for expressing bindings from SQL tuples to RDF triples, and Ontop already integrates the mapping engine. The other viable option is to write the mapping using the Ontop native mapping language, which has a more compact notation than R2RML. Each of such mappings consists of a source SQL query, and an RDF triple target with placeholders for the values of the attributes of the source query [2]. If the SQL mapping query involves aggregate operators (e.g., `AVG()`) with a `GROUP BY` clause, the endpoint automatically instantiates an intermediate view layer between the database and the Virtual Knowledge Graphs. However, the mappings involving an intermediate layer must have the data types explicitly stated.

Ontop needs three files to respond to SPARQL queries by building the corresponding VKGs: 1. an ontology file (usually .owl or .ttl) containing the ontological concepts in OWL2QL profile needed by the Ontop reasoner, 2. a configuration file for correctly instantiating a JDBC connection to the database, and 3. a mapping file for the RDF-to-SQL translation of the VKGs.

3 Requirements

This section presents our requirement analysis. The goal of Chimera is to build a general framework applicable to a variety of industrial scenarios. Chimera shall improve the support of KG-empowered analytical solutions to big data and enable the creation of *round-tripping* data pipelines. Therefore, Chimera has to be scalable and problem-agnostic to encourage the adoption of many companies.

The developed solution should fulfill the following requirements, which Table 1 maps to the respective actors:

R1 Scalability: it must be possible to instantiate any component of the solution many times on different machines, according to the user's needs.

[3] https://www.w3.org/TR/sparql11-federated-query/.

R2 Generality and Abstraction: the solution must be problem-agnostic so that a broad set of industrial scenarios can benefit from it.
R3 Ease of Deployment: the solution must be available as a Docker image, enabling easy deployment and configuration with Docker.
R4 Industrial Adoption: The solution's deployment must be based on industrially supported software to ensure support, documentation, and updated over time.
R5 Notebooks support: several data analysis libraries are available for python hence the solution must allow executing and managing SPARQL queries from Jupyter notebooks.
R6 Standards Compliance: the solution must adhere to SPARQL 1.1 standards. The Knowledge Graph has to be modeled using OWL2 standards. The DB-descriptive ontology in Ontop must be in the OWL2QL profile. The mapping of SQL datatypes to XML Schema datatypes must be compliant with the W3C recommendations and support R2RML mapping language.
R7 Semantic access to data: the relational data must be exposed to the RDF world by adopting an OBDA approach. In particular, it must be possible to access both the starting relational tables and the results of the analyzes.
R8 Semantic results persisted: it must be possible to persist the results of the data analyzes, which are built upon SPARQL query results, in the data lake.
R9 Ease of configuration: The solution must be easy to configure. Furthermore, it must be easy to instantiate a notebook connection to both the SPARQL endpoint and the big data engine.

Table 1. Requirements matrix

Requirement	KE	BA	DS	DE	SRE
R1 Scalability	X	X	X	X	X
R2 Generality and Abstraction	X				X
R3 Ease of Deployment					X
R4 Industrial Adoption					X
R5 Notebooks support		X	X		
R6 Standards Compliance	X	X	X	X	
R7 Semantic access to data	X	X	X	X	
R8 Semantic results persisted	X		X		
R9 Ease of configuration	X			X	

4 Chimera

The requirements listed in Sect. 3 led to Chimera's design and development. Indeed, Chimera enables performing KG-empowered analysis using python notebooks: Ontop$_{Spark}$ allows running SPARQL queries over a Spark instance, while `PySPARQL` allows for managing a SPARQL result in Spark.

Section 4.1 illustrates the *round-tripping* analysis while depicting Chimera's infrastructure. Section 4.2 and Sect. 4.3 provide a detailed description of Ontop$_{Spark}$ and `PySPARQL`, respectively.

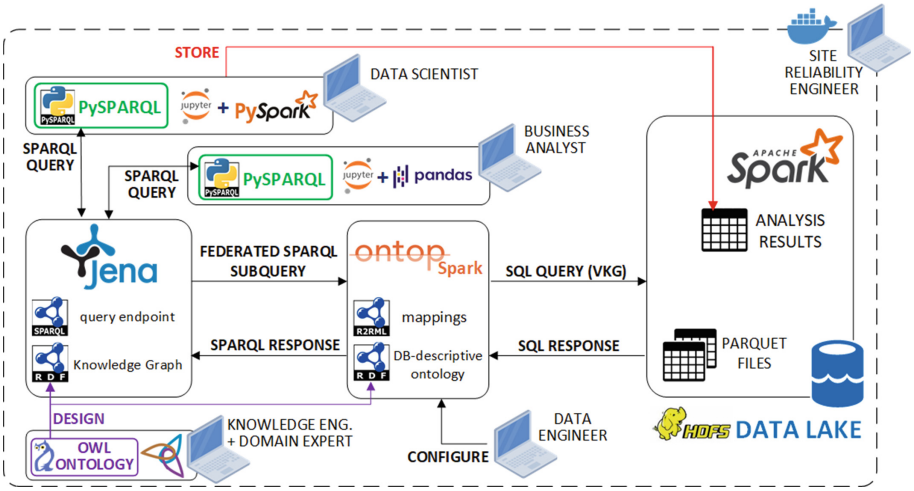

Fig. 2. The interaction between the components of the infrastructure.

4.1 Infrastructure

DSs and BAs can adopt the Chimera components into their data-science pipeline to perform KG-empowered analysis tasks. A reference deployment is available on GitHub[4] where users can find a series of `docker-compose` files that allows for easily integrating Chimera in existing big data infrastructures[5].

In particular, using Ontop$_{Spark}$ and `PySPARQL`, it is possible to accomplish what we have previously called *round-trip* analysis (R7). To better understand the *round-trip* process, it is crucial to know where the information sources are. Consider Fig. 2. The data lake is a Spark warehouse made of several Apache Parquet[6] files distributed on HDFS[7] and managed as Spark tables. Multiple servers assure the needed degree of parallelization (R1). A Spark Thrift Server exposes a JDBC endpoint that allows users to send SQL queries to the SparkSQL[8] engine, which, in turn, generates code to process the data in the data lake. On the other hand, the semantic data is stored according to the functionality it provides. Jena Fuseki[9] stores the Knowledge Graph and Ontop the DB-descriptive ontology. Notably, we chose HDFS and Jena Fuseki for our convenience, but Chimera users are free to choose alternatives. They can change HDFS in any other distributed file system as long as Apache Spark supports it (R4). Moreover, they can adopt any other RDF repository instead of Jena Fuseki, as long as it complies with SPARQL 1.1 (R2).

[4] https://github.com/chimera-suite/infrastructure.
[5] Chimera supports several Spark versions, starting from `2.4.0` to `3.1.1`. Users can change the version by selecting the appropriate image tags.
[6] https://parquet.apache.org/.
[7] https://hadoop.apache.org/.
[8] https://spark.apache.org/sql/.
[9] https://jena.apache.org/documentation/fuseki2/.

The DSs can write their analytical SPARQL queries on notebooks (R5) using `PySPARQL`, which sends them to Jena Fuseki. The query part inside the `SERVICE` clause, known as *federated query*, is resolved by Ontop$_{Spark}$ using the OBDA approach, which retrieves the data from Spark and translates the SQL responses into RDF triples using the R2RML mapping file (R6) and the DB-descriptive ontology. Once the triples are back from Ontop$_{Spark}$, Jena Fuseki has to enrich them by using the Knowledge Graph and send back the results to the notebook (R7). At this point, the result is available to the user in the form of Spark DataFrame or GraphFrame, which can be further analyzed using the notebook and, if necessary, persisted in the data lake (R8) by executing a `PySPARQL` function. The materialization task ends the *round-trip* circle. The DSs can start a new analysis iteration if needed.

DSs and BAs can also issue SPARQL queries to the Fuseki or the Ontop$_{Spark}$ endpoints (R2), but, in this way, they lose the opportunity to materialize the data in new Spark tables, hence to use the available data science libraries.

The following two sections detail Ontop$_{Spark}$ and `PySPARQL` showing their architecture and explaining how to employ them for KG-empowered analysis.

4.2 Ontop$_{Spark}$

The Ontop$_{Spark}$ extension enables Ontop to perform OBDA on relational data using Apache Spark. Given in input the mappings between RDF statements and SparkSQL queries, it is possible to submit SPARQL queries (R6) to the Ontop endpoint and obtain a response from SparkSQL based on the tables that form the data lake (R7). The integration of a distributed data processing engine such as Apache Spark allows exploiting the Ontop data integration capabilities at its maximum potential because it brings all the advantages of parallel and distributed computation to the task of answering a SPARQL query.

Ontop$_{Spark}$ performs SparkSQL queries by interacting with the Spark Thrift Server through standard JDBC calls. For this extension, we decided to use a third-party JDBC driver[10]. The extension work mainly consists of implementing seven Java classes, which we inserted inside the Ontop source code in the *ontop-rdb* package that contains all the extensions for the supported relational databases. First of all, we implemented the *SparkSQLDBMetadataProvider* that reads the database metadata by interacting with the JDBC driver. It was hard to implement because none of the drivers we explored have full support for all standard JDBC calls. In the end, it has been necessary to retrieve the default schema and the metadata. Other relevant implemented classes are the *SparkSQLDBTypeFactory* and the *SparkSQLDBFunctionSymbolFactory*. The first one has been tested to be compliant with the W3C recommendations[11](R6), while the second one translates SPARQL functions into SparkSQL functions.

Ontop$_{Spark}$ was developed according to the Ontop official guidelines[12] (R4) to ease the integration in the original codebase and is available under the Apache

[10] https://repo1.maven.org/maven2/org/apache/hive/hive-jdbc/.
[11] https://www.w3.org/2001/sw/rdb2rdf/wiki.
[12] https://ontop-vkg.org/dev/db-adapter.html#required-implementations.

License 2.0. Furthermore, it supports R2RML (R9) mappings and all the W3C standards related to OBDA of the Ontop project [2] (R6).

Ontop$_{Spark}$ is available in two different packages, namely *Ontop$_{Spark}$-Protege* and *Ontop$_{Spark}$-CLI*. The first one is an extension that allows building ontologies and mappings using the graphical interface of Protégé (R2, R4), while the second exposes a SPARQL endpoint (web GUI or CLI) used for industrial deployment (R4). We use the *Ontop$_{Spark}$-CLI* package for building a Docker image (R1, R3) that is freely available on DockerHub[13]. To complete the project and make it readily available to the adopters, we have also created a GitHub repository[14] containing the Ontop$_{Spark}$ source code and documentation.

4.3 `PySPARQL`

The `PySPARQL` module allows the users to query a SPARQL endpoint and process the response inside Apache Spark (R7). `PySPARQL` leverages `pyspark`[15] to manage Spark DataFrames, and uses well known libraries such as `SPARQLWrapper`[16] and `rdflib`[17] to handle the communication with a SPARQL endpoint and manage the result, respectively (R6). We tested the module with multiple Spark versions. `PySPARQL` is available on PyPi[18], and interested readers can access its code, tests, and a comprehensive documentation on its public git repository[19]. It is released under Apache License 2.0. Hereafter, we briefly discuss how it works. You can find a concrete code example in Sect. 5.

`PySPARQL` takes in input a SPARQL query as a python string and executes the query against the configured SPARQL endpoint. The library retrieves the results and materializes them inside the configured Spark Session. Users shall specify the endpoint configuration at initialization time and change it during the program execution (R9). In particular, the output type directly depends on the SPARQL query type. `SELECT` queries return Spark DataFrames in which the columns directly correspond to the variables declared in the SPARQL query (R2). However, `PySPARQL` does not convert the value types. The users can then process the DataFrame inside Spark and, if necessary, save the DataFrame as a Spark table (R8). `CONSTRUCT` queries return either a DataFrame or a GraphFrame depending on what the user chooses to materialize. In both cases the data resemble the constructed graph (R2). In particular, `PySPARQL` can materialize three types of DataFrame:

– TiplesDataFrame: it contains the triples of the constructed graph. It has three columns corresponding to `?subject` , `?object` and `?predicate` .

[13] https://hub.docker.com/r/chimerasuite/ontop.
[14] https://github.com/chimera-suite/OntopSpark.
[15] https://spark.apache.org/docs/3.0.1/api/python/.
[16] https://sparqlwrapper.readthedocs.io/.
[17] https://rdflib.readthedocs.io/.
[18] https://pypi.org/project/PySPARQL/.
[19] https://github.com/chimera-suite/PySPARQL.

- VertexDataFrame: it contains the literals associated with each vertex of the constructed graph. It has an ID column reporting the Internationalized Resource Identifier (IRI) of the vertex and a variable number of columns. Each column represents the IRI of the predicate that relates the vertex to a literal.
- EdgesDataFrame: it contains the relationships between vertexes. It has three columns reporting to source vertex, relationship, and target vertex IRIs.

The VextexDataFrame and EdgesDataFrame are combined to construct a GraphFrame. The user can choose the preferred output type, and the materialization happens at runtime (R1). The users can then process the results inside Spark and, optionally, save the DataFrame inside a Spark table (R8).

At the time of writing, `PySPARQL` only supports `SELECT` and `CONSTRUCT` queries. However, we plan to support `ASK` and `DESCRIBE` queries in a future release. `PySPARQL` is included in the Chimera Jupyter notebook (R5), published on Dockerhub[20] (R3).

5 Running Example

This section presents a minimal use-case of IoT anomaly detection using an extended version of the Stanford pizza ontology[21] [12]. Section 5.1 introduces the scenario, while Sect. 5.2 shows how to handle Chimera to make a *round-tripping* analysis pipeline. We published an extended version of this section as a tutorial on GitHub[22].

5.1 Scenario

PizzaInternational is a restaurant chain with many locations worldwide. To achieve a high-quality standard in all the restaurants, PizzaInternational installed new advanced ovens that integrate IoT sensors. Consequently, to accommodate the growing volumes of data produced by the ovens, they updated the IT infrastructure. It now resembles the one depicted in Fig. 2. The ovens' observations, such as temperature and cooking time, are stored in a data lake built upon HDFS and Apache Spark. As it often happens in Industry 4.0 projects, multiple Spark tables contain information about the observations of the pizzas cooked in the restaurants. Moreover, those tables have a different schema. A Spark JDBC endpoint, exposed by the Apache Thrift Server, allows users to query the data lake using SparkSQL queries.

A knowledge engineer team selected the Pizza Ontology and asked Peppo, a famous Neapolitan pizza chef, to add the optimal cooking parameters. Figure 3 illustrates a portion of the pizza Knowledge Graph. Each kind of pizza has its temperature and cooking time annotations. For example, according to Peppo's

[20] https://hub.docker.com/r/chimerasuite/jupyter-notebook.
[21] https://protege.stanford.edu/ontologies/pizza/pizza.owl.
[22] https://github.com/chimera-suite/use-case.

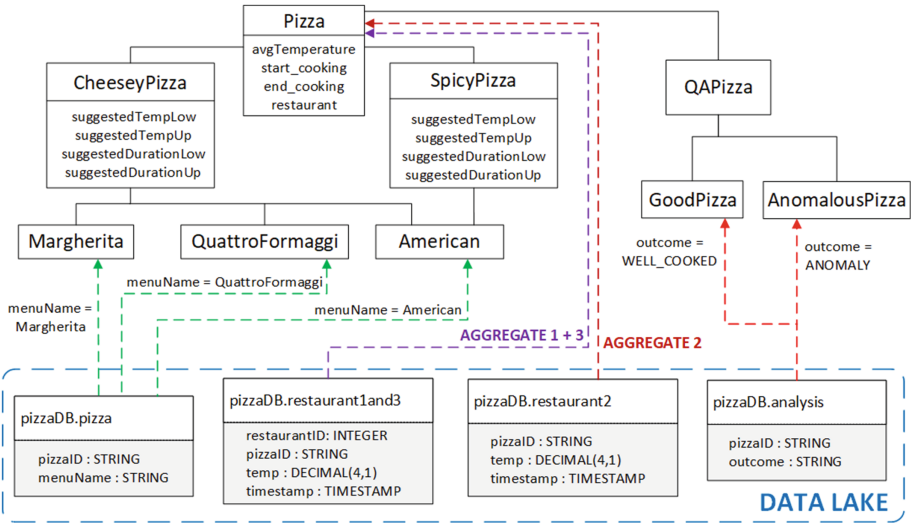

Fig. 3. Portion of the pizza ontology. The mappings are the dashed coloured lines.

experience, restaurants shall cook a *CheeseyPizza* at low temperatures to avoid burning the cheese. In contrast, they can cook a *SpicyPizza* with higher temperatures, but for a shorter period. As a consequence, some pizzas have cooking suggestions inherited from multiple classes of pizzas. For instance, an *American* pizza is both a *CheeseyPizza* and a *SpicyPizza*. It is correct to cook it adhering to both the *CheeseyPizza* and *SpicyPizza* cooking suggestions. Consequently, precise small ranges of cooking temperatures and duration should be satisfied to obtain a `"WELL COOKED"` *American* pizza, both from a *CheeseyPizza* and *SpicyPizza* perspective. The team stored the resulting Knowledge Graph in Apache Jena and exposed a SPARQL endpoint using Fuseki.

Being PizzaInternational a data-driven company, the executives are interested in exploiting the collected data to check the cooking performance across all the restaurants. Given such a requirement, the central branch's DSs have decided to make an explorative analysis of the cooking quality across all the restaurants and save those results inside Spark tables for the BAs, who needs to create a report for the executives showing which are the most critical restaurants. In the following section, we discuss how the adoption of Chimera can ease such a result.

5.2 Solution

The first part of this section shows how to configure the pipeline and write a notebook that sends analytical SPARQL queries to Jena Fuseki using `PySPARQL`. The last part shows how to persist a Spark DataFrame in the data lake and retrieve it again on a second iteration plotting data in a notebook.

```
mappingId   temperature-example
target      :{pizzaID} :temperature {temperature}^^xsd:decimal .
source      SELECT pizzaID, AVG(temp) AS temperature
            FROM pizzaDB.restaurant2 GROUP BY pizzaID
```

Listing 1.1. Example of Ontop aggregate mapping.

```
SELECT ?pizzaID ?outcome
WHERE {
  ?pizzaType :suggestedTempLow ?tempLow; :suggestedDurationLow ?durLow;
             :suggestedTempUp ?tempUp; :suggestedDurationUp ?durUp .
  SERVICE <http://ontop:8080/sparql> {
        ?pizzaID a ?pizzaType; :temperature ?avgTemp;
        ?pizzaID :start_cooking ?start; :end_cooking ?end. }
  BIND ((?end-?start) AS ?cookDuration)
  BIND( IF ((?avgTemp >= ?tempLow && ?avgTemp <= ?tempUp) &&
            (?cookDuration >= ?durLow && ?cookDuration <= ?durUp)
            ,"WELL_COOKED","ANOMALY") AS ?outcome) }
```

Listing 1.2. SPARQL query and results storage in a Spark table

To use Ontop$_{Spark}$, the DEs, together with the KEs, have to define some configuration files. In particular, they have to create a DB-descriptive ontology for describing the Spark tables data and to define the mappings between SQL and RDF that the Ontop reasoner will process to create the VKGs. Since the restaurants' table schemas are different, there is a need for multiple mappings. For example, in Fig. 3, the mapping *AGGREGATE 1+3* is different from *AGGREGATE 2*. For this example, we choose to use the Ontop native mapping language. Listing 1.1 shows a mapping that transforms a SQL aggregation, which returns the average cooking temperatures for each pizza in an assertion of the observed temperature for each pizza.

The way that Ontop manages this particular mapping deserves special attention: if the SQL statement involves aggregate operators such as `Count()` and `AVG()` with a `GROUP BY` clause, the Ontop automatically instantiates an intermediate view layer between the database and the VKGs.

The DSs, who are familiar with notebooks and python, use `PySPARQL` to express the analytical query in SPARQL and retrieve the results as a Spark DataFrame. This approach ensures flexibility because they can use any python data science library while benefitting from Peppo's experience captured in the KG. Using the code in Listings 1.2 and 1.3, the DSs retrieve the anomalous pizzas and store the results in a Spark table.

Notably, Ontop$_{Spark}$ answers the part of the query inside the `SERVICE` clause using the VKG approach. It retrieves the tuples stored in the Spark tables by making SparkSQL queries and translating the results into instances and assertions using the SQL-to-RDF mappings. Also, the KG stored in Jena Fuseki enriches the Ontop$_{Spark}$ result by adding semantic information. In the exam-

```
wrapper = PySPARQLWrapper(spark_session, fuseki_url)
resultDF = wrapper.query(query).dataFrame
resultDF.write.mode("overwrite").saveAsTable("pizzadb.analysis")
```

Listing 1.3. SPARQL query and results storage in a Spark table

```
SELECT (COUNT(?anomalous)/(COUNT(?pizzaChecked)) as ?count) ?
    restaurant
WHERE {
    SERVICE <http://ontop:8080/sparql> {
        {?anomalous a :AnomalousPizza; :restaurant ?restaurant.}
        UNION
        {?pizzaChecked a :QAPizza; :restaurant ?restaurant.} }
} GROUP BY ?restaurant
```

Listing 1.4. SPARQL query in the notebook for finding the anomalous pizzas

ple depicted in Listing 1.2, the *federated query* retrieves the pizza instances from Spark tables with average temperatures and starting/ending cooking times. Moreover, the remaining part of the query uses the KG to express the decision rules in the `BIND` clause to determine the pizza quality outcome. Furthermore, in the example, the `:American` pizzas instances are both `:CheeseyPizza` and `:SpicyPizza` , so it is correct to cook them according to the optimal parameters asserted for both the classes. Consequently, some `:American` pizzas are `"WELL COOKED"` for the `:CheeseyPizza` class but not for the `:SpicyPizza` class and vice versa. Completed the analysis, the DSs can use the code depicted in Listing 1.3 to publish the results as a Spark table for the BAs.

The BAs are interested in creating a histogram that shows the most critical restaurants. In this case, they need both the DSs results from the first *round-trip* iteration and the original restaurants' data, which relates pizzas to restaurants. Spark tables store both the information. Listing 1.4 shows the `PySPARQL` code that executes the SPARQL query that retrieves the anomalous pizzas for each restaurant. It refers to the pizzas' quality concepts present in the KG.

Fuseki and Ontop$_{Spark}$ manage the query procedure as in the previous example. However, this time the BAs does not store the results in a Spark table. They use a *barplot* to prepare the visualization in Fig. 4 for the executives.

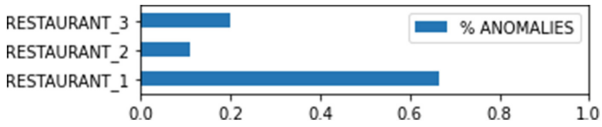

Fig. 4. The % of anomalous pizzas for each restaurant

6 Real World Deployment

Ricerca sul Sistema Energetico S.p.A.[23] (RSE) is the largest Italian public company for industrial research in the energy sector. They used Chimera to develop a solution [1] for monitoring the cables' energy workloads in the Milan city's Unareti[24] distribution network.

RSE developed a KG containing around 7 million triples representing Milan's metropolitan area. It used the methodology presented in [18] and the vocabulary specified in the IEC 61968/61970 and 62325-A standards, which the energy sector calls the Common Information Model (CIM) [19]. The KG stores the network topology and all the information about the various pieces of electrical equipment. From a bird view, nodes represent thousands of power providers, consumers, switches, breakers, etc., whereas links represent tens of thousands of conducting equipment connecting them.

Moreover, RSE collected a power load dataset containing more than 300 million time-series entries, stored it in Amazon S3, and uses the managed Spark offered by Databricks[25] as a data analytics platform.

RSE adopted Chimera because it was searching for a solution able to reduce the friction between big data and semantic technologies. In particular, RSE stated the following two requirements: 1. Let the DS team make analytical predictions in Databricks by querying both the Knowledge Graph and the big data repository. 2. Let the KE team make SPARQL queries to access the CIM network graph and retrieve the results of the Data Scientists' analysis.

Thanks to the current deployment of Chimera, RSE DSs and KEs can now build complex *round-tripping* data analysis pipelines. RSE's DSs can use `PySPARQL` in Databricks notebooks to write analytical SPARQL queries that use Ontop$_{Spark}$ to access both the KG and the time-series stored in Databricks. Moreover, they can save the resulting DataFrames as Spark tables so that RSE's KEs can leverage them in another round of analysis.

7 Related Work

The literature reports several attempts to applying distributed big data processing technologies to improve SPARQL queries' analytical capabilities.

Optique [6], a project funded by the European Commission's Seventh Framework Program (FP7), was the first successful endeavor to create an OBDA system designed for big data scenarios. The Optique underlying architecture mainly uses 1. Ontop, which manages the OBDA data access using R2RML mappings, and 2. Exareme [5], which acts as a back-end query execution component handling large-scale data processing tasks. Unfortunately, Exareme is more a research prototype than an industry-adopted software. Furthermore, it lacks the parallelization capabilities of a big data processing engine such as Apache

[23] http://www.rse-web.it/.
[24] https://www.unareti.it/.
[25] https://databricks.com/.

Spark [23]. Differently from Optique, Chimera opts for notebook centrality and proposes `PySPARQL` as a bridge between SPARQL and Spark. Other OBDA systems such as Mastro [4], UltraWrap [17], and Morph-RDB [13] are not involved in relevant projects supporting Apache Spark.

Ontop$_{Spark}$ uses a relatively small KG to access massive volumes of data not initially designed for being translated as RDF triples using the OBDA approach. To be noticed that Ontop$_{Spark}$ is not a distributed RDF datastore based on Apache Spark; thus, we have excluded from our comparison systems like SHARD [14] and PigSPARQL [15]. The more recent SPARQLGX [7] and S2RDF [16] distributed SPARQL evaluators, despite using Apache Spark as a query engine and a mapping mechanism called Virtual Partitioning, store data in Spark tables with a rigid structure that is similar to an RDF triplestore.

Semantic ANalytics StAck (SANSA) [10] is a scalable big data engine for RDF processing that uses Apache Spark and Flink as query engines. In particular, they developed Squerall [11], an OBDA tool that allows querying several SQL databases by using Spark as a wrapper for making SPARQL-to-SQL conversions. We have identified SANSA as Chimera's main competitor, as the latter tool claims to offer the same features. In building Chimera, we bid on extending Ontop because of its broad and active GitHub community and the recent start of a commercial offering. We hope to benefit from an OBDA engine that will mature update after update.

To demonstrate the differences between the two solutions, we ran a series of tests[26]. The comparison dataset has been taken from the official Sansa-stack repo[27] and comprises five CSV files inspired by the BSBM[28] benchmark. We used the nine original testing queries of SANSA (`Q1` to `Q10`, `Q9` not available) plus two additional SPARQL queries to highlight the differences between Ontop$_{Spark}$ and Squerall. Since Ontop$_{Spark}$ needs an Ontology for running, we needed to adjust the SANSA's mappings accordingly.

Figure 5 demonstrates that the query execution times are comparable (we run each query 10 times). The differences between the two solutions are probably related to design implementation choices. Ontop$_{Spark}$ was unable to execute the query `Q6` (missing implementation of the *regex()* function) and `Q10` because the query made by Squerall violates the RDF entailment regime of SPARQL; it asks for triples whose subject is a literal. However, thanks to the integrated reasoner and the full compliance with the RDF syntax and OWL2QL standard, Ontop$_{Spark}$ has been able to execute query `Q(?s ?p)`, which retrieves all the subjects and predicates given a fixed object, and query `Q(?s ?p ?o)`, which retrieves the full RDF materialization of the dataset under OWL2QL *entailment regime*. The queries achieved an average execution time of 12.7 sec for `Q(?s ?p)` and 183 sec for `Q(?s ?p ?o)`, respectively. It was not possible to execute the two queries mentioned above with Squerall.

[26] https://github.com/chimera-suite/OntopSpark-evaluation.
[27] https://github.com/SANSA-Stack/SANSA-Stack/tree/develop/sansa-query/sansa-query-spark/src/test/resources/datalake.
[28] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/.

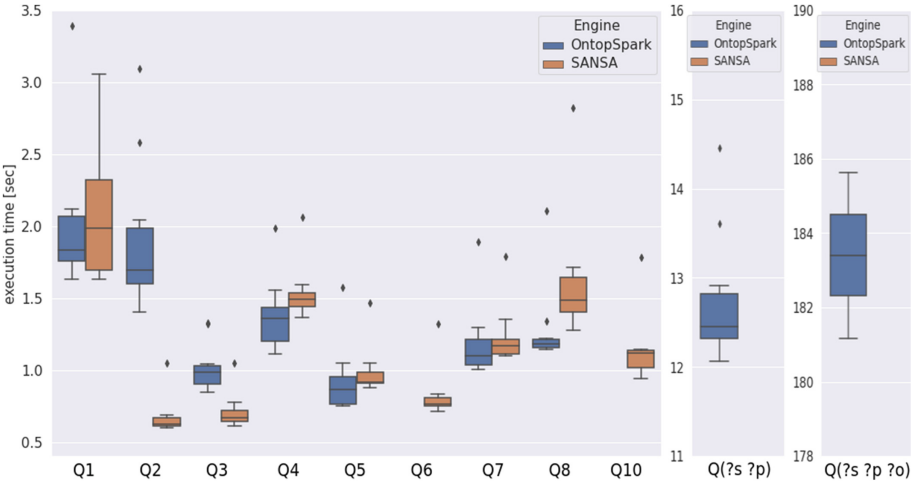

Fig. 5. Query execution times comparison wrt. Squerall(SANSA)

8 Conclusions and Future Work

This paper presents Chimera[29], an open-source software suite for KG-empowered analytics tasks in big data scenarios. Chimera allows querying a Spark data lake with SPARQL using Ontop$_{Spark}$ to get a semantically enriched response by leveraging reasoning on a KG. Also, python notebook users can leverage `PySPARQL` to automatically get the SPARQL query response converted in a Spark DataFrame/GraphFrame, which can be further analyzed using the notebook and persisted into Spark on need. Even if they are two separated components, the synergy between them enables building *round-tripping* pipelines where semantic technologies enrich data going back and forth from Spark.

To provide evidence of the benefits of Chimera, we presented two scenarios where the integration of big data technologies with an OBDA approach helps solving analytical problems. In particular, we presented a running example that demonstrates the technology in a relevant environment showing how to perform *round-trip* analysis using Chimera. Moreover, we presented a real-world deployment, which proves the usage of Chimera in an operational environment.

All the Chimera's components are available on GitHub[30]: Ontop$_{Spark}$ (we opened a pull request[31] and merged the code into Ontop) and `PySPARQL`. We also published the Docker images of Ontop$_{Spark}$ and Chimera Jupyter notebook, which integrates `PySPARQL`, on Dockerhub[32]. Thanks to RSE co-founding and Politecnico di Milano's resources, we are committed to maintain ad improve

[29] https://chimera-suite.github.io/.
[30] https://github.com/chimera-suite.
[31] https://github.com/ontop/ontop/pull/422.
[32] https://hub.docker.com/u/chimerasuite.

Chimera for the following years. The future development plans include Ontop-Stream, an Ontop extension for performing Streaming-OBDA, using continuous RSP-QL queries, over relational data streams coming from heterogeneous data sources (Kafka, Kinesis, Hive) ingested into Apache Flink dynamic tables.

Acknowledgements. This work has been partially financed by the Research Fund for the Italian Electrical System in compliance with the Decree of Minister of Economical Development April 16, 2018. We also thank Marco Balduini for initiating Chimera.

References

1. Bionda, E., et al.: The smart grid semantic platform: synergy between iec common information model (cim) and big data. In: 2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe). IEEE (2019)
2. Calvanese, D., et al.: OBDA with the ontop framework. In: SEBD, pp. 296–303. Curran Associates, Inc. (2015)
3. Calvanese, D., et al.: Ontop: answering SPARQL queries over relational databases. Semant. Web **8**(3), 471–487 (2017)
4. Calvanese, D., et al.: The MASTRO system for ontology-based data access. Semant. Web **2**(1), 43–53 (2011)
5. Chronis, Y., et al.: A relational approach to complex dataflows. In: EDBT/ICDT Workshops. CEUR Workshop Proceedings, vol. 1558. CEUR-WS.org (2016)
6. Giese, M., et al.: Optique: zooming in on big data. Computer **48**(3), 60–67 (2015)
7. Graux, D., Jachiet, L., Genevès, P., Layaïda, N.: SPARQLGX: efficient distributed evaluation of SPARQL with apache spark. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 80–87. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46547-0_9
8. Kharlamov, E., et al.: Ontology based data access in statoil. J. Web Semant. **44**, 3–36 (2017)
9. Kharlamov, E., et al.: Semantic access to streaming and static data at siemens. J. Web Semant. **44**, 54–74 (2017)
10. Lehmann, J., et al.: Distributed semantic analytics using the SANSA stack. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10588, pp. 147–155. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_15
11. Mami, M.N., Graux, D., Scerri, S., Jabeen, H., Auer, S., Lehmann, J.: Squerall: virtual ontology-based access to heterogeneous and large data sources. In: Ghidini, C., et al. (eds.) ISWC 2019. LNCS, vol. 11779, pp. 229–245. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30796-7_15
12. Noy, N.F., McGuinness, D.L., et al.: Ontology development 101: A guide to creating your first ontology (2001)
13. Priyatna, F., Corcho, Ó., Sequeda, J.F.: Formalisation and experiences of r2rml-based SPARQL to SQL query translation using morph. In: WWW, pp. 479–490. ACM (2014)
14. Rohloff, K., Schantz, R.E.: High-performance, massively scalable distributed systems using the mapreduce software framework: the SHARD triple-store. In: PSI EtA, p. 4. ACM (2010)
15. Schätzle, A., Przyjaciel-Zablocki, M., Lausen, G.: Pigsparql: mapping SPARQL to pig latin. In: SWIM, p. 4. ACM (2011)

16. Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., Lausen, G.: S2RDF: RDF querying with SPARQL on spark. Proc. VLDB Endow. **9**(10), 804–815 (2016)
17. Sequeda, J.F., Miranker, D.P.: Ultrawrap: SPARQL execution on relational data. J. Web Semant. **22**, 19–39 (2013)
18. Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A. (eds.): Ontology Engineering in a Networked World. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-24794-1
19. Uslar, M., Specht, M., Rohjans, S., Trefke, J., González, J.M.: The Common Information Model CIM: IEC 61968/61970 and 62325-A practical introduction to the CIM. Springer Science & Business Media (2012)
20. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: a survey. In: IJCAI, pp. 5511–5519. ijcai.org (2018)
21. Xiao, G., Ding, L., Cogrel, B., Calvanese, D.: Virtual knowledge graphs: an overview of systems and use cases. Data Intell. **1**(3), 201–223 (2019)
22. Yu, H., Liaw, S., Taggart, J., Khorzoughi, A.R.: Using ontologies to identify patients with diabetes in electronic health records. In: International Semantic Web Conference (Posters & Demos). CEUR Workshop Proceedings, vol. 1035, pp. 77–80. CEUR-WS.org (2013)
23. Zaharia, M., et al.: Apache spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016)