OPTIMA: A Hybrid OBDA System for Efficiently Querying Large Heterogeneous Data

Chahrazed B. Bachir Belmehdi [a,*], Abderrahmane Khiat [b] and Nabil Keskes [a]
[a] *Department first, ESI-SBA Institute, SBA, Algeria*
E-mail: cb.bachirbelmehdi@esi-sba.dz
[b] *Enterprise Information Systems, Fraunhofer IAIS, Bonn, Germany*
E-mails: abderrahmane.khiat@iais.fraunhofer.de, n.keskes@esi-sba.dz

Abstract. The current decade is witnessing a remarkable evolution in terms of big data virtualization. Data is queried on-the-fly against the original data sources without any prior data materialization. Ontology-Based Big Data Access solutions by design use a fixed model, e.g., TABULAR, as the only Virtual Data Model - a uniform schema that is built on-the-fly to load, transform, and join relevant data. While other data models such as GRAPH or DOCUMENT are more flexible and, thus, can be more suitable for some common types of queries such as join or nested queries. Those queries are, in many cases, hard to predict because they depend on many criteria such as query plan, data model, data size, operations e.g., join, filter. To address the problem of selecting the optimal virtual data model for various queries on large datasets, we develop OPTIMA. OPTIMA is a framework that (1) builds on the principle of ontology-based data access to enable the querying, aggregating, and joining of large heterogeneous data in a distributed manner using a unique query language SPARQL and (2) calls the deep learning method to predict the optimal virtual data model using the features extracted from SPARQL queries. OPTIMA currently leverages state-of-the-art Big Data technologies, Spark, and implements two virtual data models, GRAPH and TABULAR, and supports out-of-the-box five data sources Neo4j, MongoDB, MySQL, Cassandra, and CSV. Extensive experiments show that OPTIMA returns the optimal virtual model with an accuracy of 0.831, thus reducing the query execution time by over 40% in favor of tabular model selection and over 30% for the graph model selection.
OPTIMA is available on GitHub https://github.com/chahrazedbb/OPTIMA

Keywords: Data Virtualization, Big Data, Ontology Based Data Access, Deep Learning

1. Introduction

Data virtualization approaches tackle data integration challenges by creating a virtual data model under which the heterogeneous formats are homogenized *on-the-fly* without data materialization [1]. Ontology-Based Data Access (OBDA) [2] approaches [3] maintain data virtualization with practical knowledge representation models and ontology-based mappings. However, existing solutions dedicated to big data [4][5][6] use by design a fixed model, e.g., TABULAR as the only virtual data model[1] to load and transform the requested data into a uniform model to be joined.

*Corresponding author. E-mail: cb.bachirbelmehdi@esi-sba.dz.
[1] We denote GRAPH and TABULAR to distinguish between Virtual model and data source models.

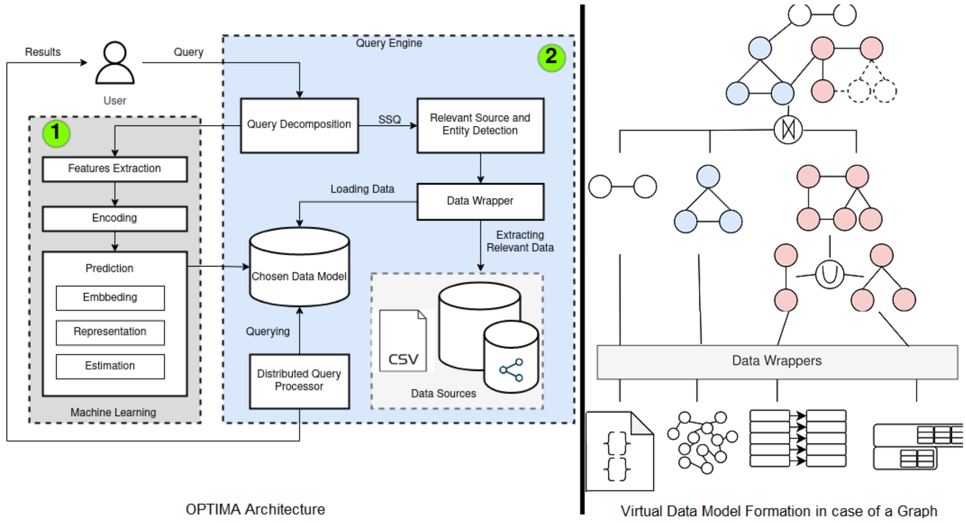


Fig. 1. OPTIMA Framework

Nevertheless, the TABULAR virtual model can have performance downsides for queries involving many join operations on large data [7]. In contrast, other data models, such as GRAPH, perform better for such queries. On the other hand, the TABULAR model performs better for queries that involve selection, or projection [8]. Therefore, there is a need to support different virtual models and select the optimal one based on query behavior, thus saving operational execution time. However, predicting the optimal virtual model is challenging since the selection depends on many criteria, such as query plan, data model, size, and operations.

This challenge raises the following research questions: *based on a given query, RQ1: does the data model affect query execution time? RQ2: which virtual data model is optimal, i.e., the model that has the lowest cost, precisely, the lowest query execution time? and how to select it?"*

To address this research problem, we introduce our operational tool OPTIMA, which was briefly presented and published in our earlier work [9]. OPTIMA is an extensible framework that uses two Virtual models, GRAPH and TABULAR, and supports out-of-the-box five data sources, Property Graph, Relational, Tabular, Document-based, and Wide-Columnar, to query large heterogeneous data in an efficient way. To select the optimal virtual data model GRAPH or TABULAR, we used one hot vector encoding to transform different SPARQL features into hidden representations. Next, we embed these representations into a tree-structured model, which we use to classify the virtual model GRAPH or TABULAR that has the lowest query execution time. Experiments show that OPTIMA reduces the query execution time of over 40% for the TABULAR model selection and over 30% for the GRAPH model selection. We describe each component of our system OPTIMA as illustrated in Figure 1, Label 1; followed by conducted experiment and related work.

2. OPTIMA: Optimal Virtual Model for Querying Large Heterogeneous Data

2.1. Virtual Data Model Prediction

Built on top of OBDA components, this distinctive component implemented in OPTIMA aims to select the optimal virtual data model GRAPH or TABULAR based on the query behavior see Figure 2. The component receives the SPARQL query as input and predicts the optimal virtual data model with the lowest execution time. The deep learning model starts by breaking down the SPARQL query plan into nodes. Each node includes a set of query features that significantly affect the query execution time (e.g., filter). The different features are then encoded using a one-hot vector. Next, we propose a tree-structured model that takes the encoded features of SPARQL query as input to learn each sub-plan representation effectively and outputs the optimal virtual data model, GRAPH or TABULAR, that has the lowest query execution time. Our model consists of an embedding layer to condense the features' vectors and an estimation layer to estimate the optimal virtual data model. In addition, the model includes an intermediate representation layer to capture the correlation between the joined star-shaped queries. Once the optimal model is predicted, the rest of the OBDA components and operations (e.g., join) follow the optimal virtual model predicted GRAPH or TABULAR.

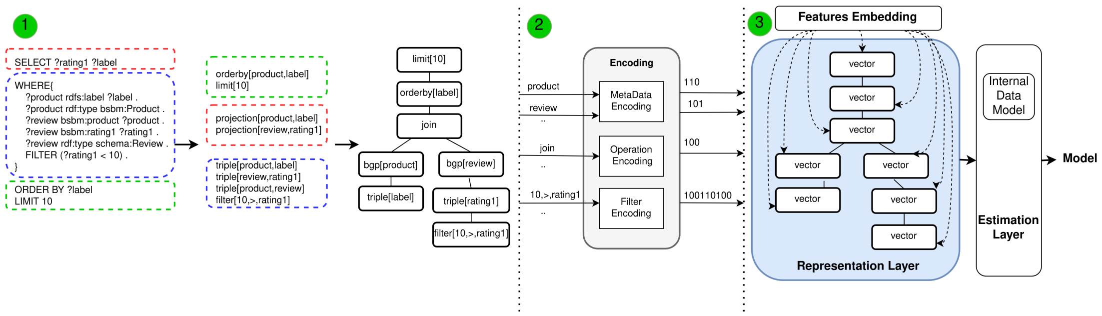


Fig. 2. Architecture of Predictive Model for Optimal Virtual Data Model

2.2. *Query Decomposition and Relevant Entity Detection*

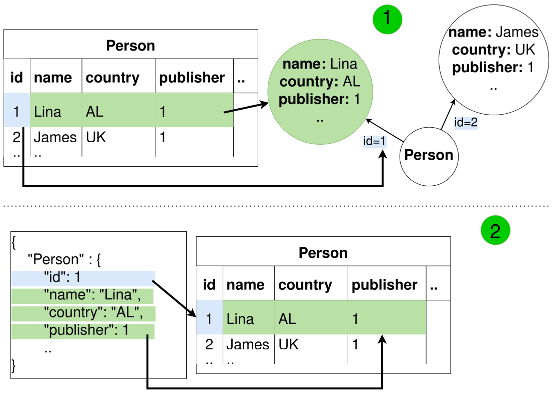
This component decomposes the SPARQL query into star-shaped queries. More precisely, the query's Basic Graph Pattern (BGP) is divided into a set of sub-BGPs, where each sub-BGP contains all the triple patterns sharing the same subject variable. Those sub-BGPs sharing the same subject are called a star-shaped query. Next, this component analyzes each star-shaped query and visits the mappings file to obtain the data source's path and the attributes mapped to each element of the star-shaped query, i.e., relevant entities. This information is then passed to the data wrapper to load relevant entities.

2.3. *Data Wrapper*

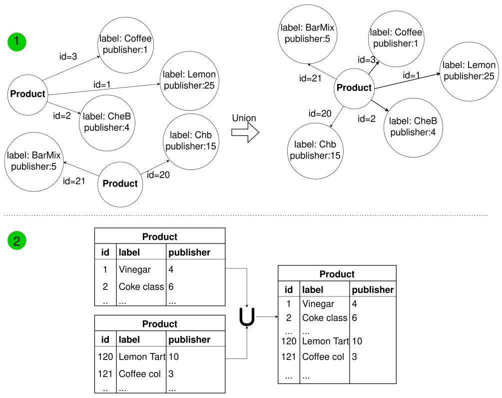
Once the sources and relevant entities are identified using mappings, the data wrapper converts relevant entities (e.g., tables) from their original models to data that comply with the optimal virtual data model predicted, which is actually the data structure of the computation unit of the query engine. This conversion occurs at query-time, which allows for the parallel execution of expensive operations, e.g., join. Query engines implement wrappers called connectors to convert data entities from the source to the virtual data model, performing transformation of data source, e.g., relational model to virtual model, e.g., GRAPH (see Figure 3a).

Each star-shaped query corresponds to one relevant entity, and thus one single virtual data model is created. According to the mapping, this occurs when the relevant entity is retrieved only from one data source. Otherwise, if the relevant entity is retrieved from multiple data sources, then the virtual model for one relevant entity is the union of the temporary virtual models created for each source (see Figure 3b). Below we describe the data sources' model transformation by wrappers into GRAPH and TABULAR.

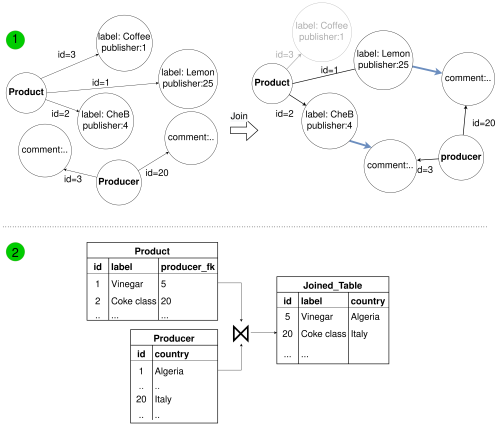
- For the virtual data model of type GRAPH, one relevant entity of a star-shaped query from relational and tabular models is a table with specific columns, which is transformed into one virtual model GRAPH. Figure 3a, Label 1 illustrates the transformation process from relational into GRAPH, following the process described in [10]. For each table row, a vertex is created with the same label as the table's name (e.g., table' Person' corresponds to all vertices with the label "Person") in addition to the root vertex. Edges are created between vertices and the root vertex, whereas the properties of each vertex are the columns of the table (e.g., column 'name' corresponds to property 'name'), and the values of the properties are the table's cell information. The process is applied to a property graph that has the same data structure as the GRAPH. Thus, the transformation process is a direct mapping; the node corresponds to a vertex, the node's properties correspond to the properties of the vertex, and



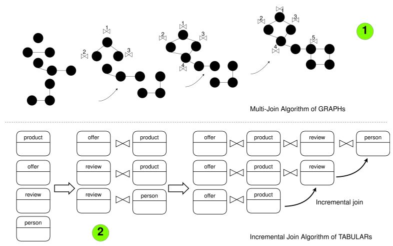
(a) Figure 3a: Transformation to Virtual Models



(b) Figure 3b: Union of Temporary Virtual Models



(a) Figure 4a: Join of Temporary Virtual Models



(b) Figure 4b: Join Algorithms of Virtual Models

the relationships between nodes correspond to edges between vertices. Following a similar process described in [11], the document-based and wide-columnar models (e.g., JSON or XML files) are transformed into the virtual GRAPH's vertices by iteratively going through each element of the object. Those vertices share the same label as the root name (object). Edges are created between vertices and the root vertex. The vertex's properties and their values are mapped to keys and key-values of that element, respectively.

– As for the virtual data model of type TABULAR, the selected object as a relevant entity of documented-based and wide-columnar is parsed to create a virtual TABULAR. As described in [12], the Virtual TABULAR consists of a table with a name similar to the root object's name (e.g., a table 'Person' from object name 'Person'). A new row is inserted by iterating through the object's elements into the corresponding table. The corresponding key-values are saved under the column representing the cell information (see Figure 3a, Label 2). As for the property graph, a Virtual TABULAR is created for each distinct graph that matches the pattern queried, following the approach proposed in [11]. The Virtual TABULAR consists of a table with the same name as the label shared by vertices. A default column' ID' is created to store each vertex id with the same label. A new row is inserted for each vertex into the corresponding table; For each distinct property of the vertex, an additional column is created with the same datatype as the extracted property. The cell information consists of the values extracted from the vertex's properties. Finally, the relational and tabular data structure is the same to some extent as the data structure of the Virtual TABULAR Model. Therefore, the transformation process is a direct mapping between both models.

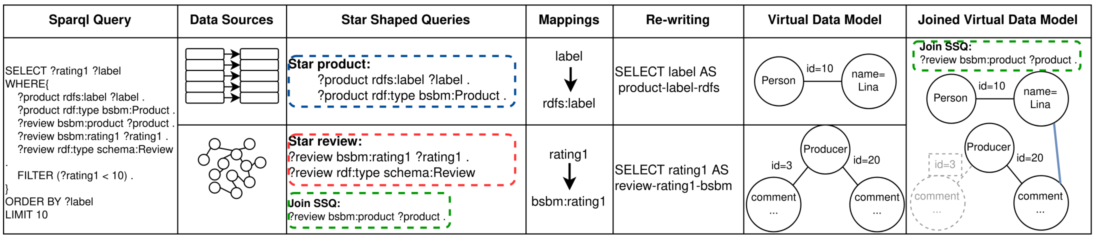
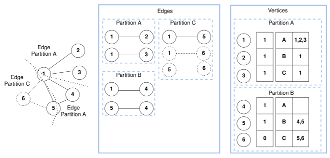
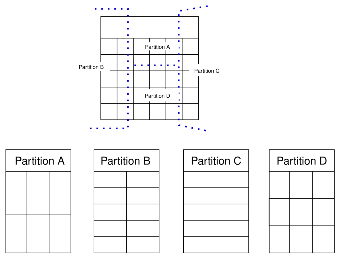


Fig. 5. Process of Joining Heterogeneous Data Sources



(a) Figure 6a: GRAPH Parallel [13]



(b) Figure 6b: TABULAR Parallel

2.4. Distributed Query Processor

Distributed Query Processor is the environment where queries are executed. OPTIMA calls for Graphx and Apache Spark to use two different virtual data models, GRAPH and TABULAR. We consider two types of data models, GRAPH and TABULAR, which allow for (1) graph-parallel computation (Figure 6a) and (2) data-parallel computation (Figure 6b), which affect the query performance. We should point out that we did not employ any query optimization function to choose the most efficient query execution plan; instead, we focused on the join operation. If deep learning predicts that the optimal virtual model is of type GRAPH, then for each relevant entity, one virtual GRAPH model is generated by wrappers. The wrappers use API to access the data source and perform the transformation. OPTIMA joins those GRAPHs into a Final Virtual GRAPH (see Figure Figure 4a) using "multi-join algorithm" (see Figure Figure 4b, Label 1)or TABULARs into a Final Virtual TABULAR using "incremental join algorithm" (see Figure Figure 4b, Label 2). This joining is through connections between star-shaped queries; see Figure 1 Label 2 and Figure 5. However, GRAPH and TABULAR have different structures. For example, the interaction with GRAPH is possible by means of Graph Pattern Matching operations (Cypher-like), while the interaction with TABULAR is possible by SQL-like functions. SPARQL and star-shaped query operations (e.g., limit) are translated into Virtual Data model operations (e.g., "take" in the case of Graphx).

3. Experimental Setup

We conducted an experimental study to evaluate OPTIMA performance compared to the state-of-the-art SPARK-based Sequerall, which uses dataframes (i.e., TABULAR) as a virtual data model. We used five tables to enable up to 4-chain joins. These tables are loaded in five different data sources Cassandra, MongoDB, CSV, Neo4j, and MySQL. Table 1 shows the described information about data. We generated 5150 queries with 0-4 joins,0-45 selection, 0-16 filter, limit, and OrderBy. We take 4120 queries for training the model and 1030 queries for validation. We run the evaluation on Ubuntu 64-bit with an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, allocating 8GB of RAM.

We conducted an empirical study to evaluate OPTIMA performance and address our research problem with the following questions: RQ1: What is the query performance using OPTIMA? RQ2: Is the time of prediction plus the time of query execution using an optimal virtual model equal to the fixed one? RQ3: What is the query performance when using TABULAR versus GRAPH? RQ4: What is the accuracy of OPTIMA and machine learning? RQ5:

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Product	✓	✓		✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓
Offer	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓		✓	✓
Review	✓	✓	✓	✓		✓	✓	✓		✓			✓	✓	✓			✓	✓	✓
Person			✓	✓		✓			✓		✓	✓						✓	✓	✓
Producer	✓		✓	✓		✓	✓	✓			✓				✓			✓	✓	✓
`PROJECT`	✓16	✓5	✓29	✓45	✓24	✓45	✓38	✓38	✓24	✓34	✓4	✓6	✓32	✓34	✓4	✓5	✓9	✓45	✓45	✓5
`FILTER`	✓16		✓12	✓1		✓5	✓1	✓1					✓1	✓1	✓4			✓2	✓3	
`ORDERBY`	✓1	✓1			✓1		✓1		✓1	✓1			✓1	✓1			✓1			
`LIMIT`	✓300	✓2		✓20	✓4	✓20	✓20	✓80			✓10		✓13	✓19	✓1000	✓1000				

Table 1

Tables and Operations involved in Queries.

What is the query performance of OPTIMA compared to the state-of-the-art, e.g., Squerall [6]? RQ6: What is the impact of involving more data sources in a join query? RQ7: What is the resource consumption (CPU, memory) of OPTIMA while running various queries? RQ8: What is the time taken by each transformation process?

3.1. Method

We consider two studies:

- In the first study, we compare OPTIMA's results with SPARK-based Squerall's results. Our comprehensive literature review did not reveal any single work except Squerall that is available and that support most data sources. Squerall uses two big data engines, Presto and SPARK: Presto-based, where the virtual model of Presto engine (which cannot be controlled by users) is used for query processing, and SPARK-based, where DataFrames are created as a virtual data model. To make the results comparable, we choose SPARK-based Squerall and extend it to support Neo4j. We assess the accuracy of OPTIMA in terms of (1) results (accuracy), (2) time, and (3) CPU and memory usage compared to SPARK-based Squerall. We should note that comparing the overall execution time of OPTIMA against an original system, e.g., relational for a given query, is impossible because we are querying various heterogeneous formats and models.
- In the second study, we inspect OPTIMA's main components: machine learning, data wrappers, and query execution. We observed the behavior of query execution for GRAPH and TABULAR in terms of time. For the data wrapper, we investigate the time taken for the transformation process from data sources to GRAPH or TABULAR. As for the machine learning component, we compare our model with the LSTM model in terms of accuracy and time. The LSTM model takes the encoded vectors of SPARQL features as input without any correlation and outputs the data model.

3.2. Experiment 1

In this experiment, we load BSBM* as described above to obtain the results from OPTIMA and SPARK-based Squerall. Then, we run 5150 SPARQL queries and compare the results. This comparison allows us to confirm the correctness of the results returned by OPTIMA. Table 3a shows the results of OPTIMA and SPARK-based Squerall of a complex SPARQL query Q21. The results are the same for both systems, which confirms that OPTIMA is able to support and join large data coming from different datasets.

Table 2 illustrates the execution time returned by both systems. As can be observed, OPTIMA excels Squerall for queries that involve multiple joins. The time difference ranges from 0 to 80000 milliseconds (ms). This difference is due to the predicted virtual data model, e.g., Q19, Q20, in which deep learning predicted that the Virtual model of type GRAPH is optimal. We also observe a small difference in the execution time (ranging from 0 to 30 ms) in favor of Squerall compared to OPTIMA for queries that involve multiple projections, e.g., Q7, Q10. This is explained by the fact that the optimal virtual model is identical to Squerall's, and both Squerall and OPTIMA used the same APIs to call data (wrapper). However, the data model prediction time added to OPTIMA makes it slightly slower than Squerall. Furthermore, the average execution time of Squerall is greater than 4000 ms compared to the average execution time of OPTIMA 2400 ms, as shown in table 3b. These results illustrate the benefits of OPTIMA over existing systems; thus, RQ1 and RQ5 are answered.

System	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
OPTIMA	1291	1254	730	10299	10199	1553	7104	8442	10094	4694	2575	233	4673	4487	2397	2881	1698	4607	2804	5648
Squerall	4098	2519	3091	10283	10191	7984	7089	8427	10088	4684	2561	1400	4644	4469	3885	2875	3314	8742	9059	7407
Time Difference	2807	1265	2361	16	8	6431	15	15	6	10	14	1167	29	18	1488	6	1616	4135	6255	1759

Table 2

Time in ms per Query of OPTIMA & Squerall

To check if deep learning is reducing the overall execution time of OPTIMA by selecting the optimal virtual data model. We first illustrate the time taken by OPTIMA's components: machine learning algorithm, query execution over GRAPH model, and query execution over TABULAR against SPARK-based Squerall. We run OPTIMA and Squerall over 1030 queries. Results are shown in table 5a. The average execution time of the machine learning component is very short, 12 ms, while the average time for GRAPH is 1320 ms and TABULAR is 2862 ms. Results show that GRAPH is faster than TABULAR for most queries, even with prediction time. In summary, only 14% of the queries were initially faster for OPTIMA (using GRAPH as a virtual model) compared to Squerall and became in the later favor. This is explained by the fact that for those queries, there is a slight difference in execution time using GRAPH compared to Squerall. This answers RQ2.

Finally, we record the Resource Consumption (i.e., Memory and CPU) taken by OPTIMA and SPARK-based Squerall. The results reported in Table 3c show that the CPU is not fully used by OPTIMA and SPARK-based Squerall (around 0.21% was used). This means that the complexity of queries does not impact CPU consumption. As for the total memory reserved, OPTIMA consumed about 1GB over 8GB per node, while SPARK-based Squerall used at most 1GB. The same CPU and memory could be explained by the fact that both are using the same query engine - SPARK and the distribution of CPU between the nodes for loading and transformation. This answers RQ7.

Query	OPTIMA	Sqyerall
SELECT DISTINCT ?productLabel ?producerLabel WHERE { product rdfs:label ?productLabel . ?producer rdfs:label ?producerLabel . ?product rdf:type bsbm:Product . ?product bsbm:producer ?producer .}	['Bar Mix Lemon','Coke Classic 355 Ml']	['Bar Mix Lemon','Coke Classic 355 Ml']

(a) table 3a: Query Result Returned by OPTIMA & Squerall

System	Time (ms)
OPTIMA	2400
Squerall	4200

(b) table 3b: Avg Time

Metrics	**OPTIMA**	**Squerall**
CPU average (%)	0.21	0.20
Max memory (GB)	1.0	0.97

(c) table 3c: Resource Consumption

3.3. *Experiment 2*

In this study, we evaluate the main components of OPTIMA.

3.3.1. *Machine Learning Component*

We evaluated our model with an LSTM model to assess our encoding techniques and prediction model. We use 5150 queries, 80% are used for training, and 20% are used for validation. We train both models on the same dataset and compute the accuracy and Cross-entropy loss function. Next, we evaluate the models' efficiency in terms of the models' training time and prediction time. Table 5b shows that our tree-structured based method outperforms the LSTM model with an average accuracy of 0.831 for our model against 0.708 for the LSTM model. The cross-entropy loss is equal to 0.00018 for our model compared to 1.92027 for LSTM. This is because the LSTM model relies on the independent assumption among different operations and attributes, while tree-structured-based methods can capture the correlations between operations and attributes/entities. Our model achieves the best performance, as it captures more correlations. This answers RQ4.

3.3.2. *Query Execution*

As shown in Table 4, the analysis of experimental results indicates that GRAPH is faster than TABULAR in most cases, except for queries like Q8 and Q10. It has comparable to slightly lower performance in Q16. This

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Prediction Time	3	3	4	6	4	5	5	6	2	4	5	1	5	5	4	3	2	4	4	4
GRAPH	1143	1161	1239	1243	306	3181	7168	12237	4977	16681	1211	3567	482	1285	766	2883	6639	1366	3370	1723
TABULAR	4098	2519	3091	10283	10191	7984	7089	8427	10088	4684	2561	1400	4644	4469	3885	2875	3314	8742	9059	7407

Table 4

Time in ms per Query of Prediction, GRAPH & TABULAR

Condition	**Avg. time (ms)**
Machine Learning	12
Only GRAPH	1320
Only TABULAR	2862

(a) Table 5a: Time of Deep Learning, GRAPH & TABULAR

Cost	Loss	Accuracy
LSTM	1.92027	0.708
Our Model	0.00018	0.831

(b) Table 5b: Loss & Accuracy of Deep Learning Models

confirms that the optimal model is very important to reduce the execution time of queries. The total execution time ranges from 50 to 90000 ms, with 90% of all cases being about or below 3000 ms. OPTIMA virtual data model of type GRAPH is faster in queries that involve joins (ranging from 50 to 40000 ms), while the TABULAR model outperforms the GRAPH model in queries involving more projections (ranging from 200 to 90000 ms).

This is explained by the fact that the GRAPH is designed to store connections between data. Therefore, queries do not scan the entire graph to find the nodes that meet the search criteria. It looks only at nodes that are directly connected to other nodes, while SQL-like methods used by the TABULAR model require expensive join operations because they traverse all data to find the data that meets the search criteria. On the other hand, the TABULAR model is faster when handling projections because the data structure is already known, and data can be easily accessed by column names. Conversely, the GRAPH model does not have a predefined structure for the data, and each node attribute has to be examined individually during the projection query.

The number of joins has a decisive impact on query performance; it should be taken into consideration with other factors, e.g., size of involved data, presence of filters, and selected variables. For example, Q2 joins only two data sources, Product and Review (1254 ms) but has comparable performance with Q1 (1291 ms), which joins four entities (Product, Offer, Review, and Producer). This may be due to the presence of filtering in Q1 (16 filters), which significantly reduces intermediate results to join. Q3 involves four data sources, yet it is among the fastest queries. This is because it involves the small entities Person and Producer, which is another reason to reduce intermediate results to join. With five data sources to join, Q4 is among the most expensive queries (10299 ms). This can be attributed to the fact that the filter on Product is selective (?language = "en"), which results in large intermediate results to join, in contrast to Q6 ($?price < 8000$). Although the four-source join Q7 and Q8 involve the small entity Producer, they are the most expensive queries that execute over the GRAPH model; this can be attributed to many projections (38 attributes). Thus, we answer RQ3 and suggest that operations can affect query execution time.

Model	**Neo4j**	**JDBC**	**CSV**	**Cassandra**	**MongoDB**	**Loading**
GRAPH	138	954	196	7695	188	4.327
TABULAR	3275	199	255	5319	330	7.141

Table 6

Time (ms) of Data transformation to GRAPH & TABULAR

3.3.3. Data wrapper Time

To answer RQ8, we evaluate, in this study, the time needed to load the data from data sources to the virtual data model of type GRAPH or TABULAR (see Table 6). Since the transformation process is different, we expect different wrapper behavior. In the table, we illustrate the time needed by each wrapper with the following observations:

– Neo4j connector loads 50000 nodes from Neo4j within 138 ms into GRAPH, compared to 3275 ms in TABULAR. This is explained by the fact that the graph property used by Neo4j has the same structure as the GRAPH model.

- CSV connector loads 50000 rows within 196 ms from CSV files into GRAPH, compared to 255 ms in TABULAR, even though CSV files save data into tables. This can be explained by the fact that the GRAPH virtual model is a schema-less model that loads data directly without the need to preserve data structure, while TABULAR takes time to build the data schema.
- JDBC connector loads 50000 rows from MySQL database within 954 ms into GRAPH, compared to 199 ms in TABULAR. This can be explained by the fact that MySQL uses a relational model with the same data structure as the Virtual TABULAR model.
- MongoDB connector loads 50000 rows from MongoDB within 188 ms into GRAPH, compared to 330 ms in TABULAR. This can be explained by the fact that MongoDB is document-based, i.e., schema-less, the same as the GRAPH Virtual model, unlike the TABULAR model, which needs to build a data schema.
- Cassandra connector loads 50000 rows within 7695 ms into GRAPH, compared to 5319 ms in TABULAR. This can be explained by the fact that Cassandra uses a columnar data model, which is closer to the TABULAR model, even though it is a NoSQL database.

4. Related Work

Our literature review reveals two categories addressing data virtualization. These two categories are namely "ontology-based data access" and "non-ontology-based data access" [6]. Non-ontology-based data access approaches mostly use SQL-like as query language and implement a virtual relational model [14, 15], defining views of relevant data from sources having a relational model. Those views are generated based on mapping assertions that associate the general relational schema with the data source schemata. The shortcomings of these approaches is that the schema modifications and extensions are very rigid due to mappings and may depend on complex constraints. Furthermore, these approaches use Self-Contained Query [16] where users cannot control the structure of the virtual data model. OBDA [17] approaches use SPARQL as a unified access language and detect relevant data from sources to be joined through ontologies and standardized mappings. This provides flexibility in modifying and extending the ontology and mappings with semantic differences found across the data schemata.

We identified commercial systems such as Stardog (www.stardog.com), and Oracle Spatial and Graph (www.oracle.com/technetwork/database/options/spatialandgraph, Mastro [18], Ultrawrap [19] and open-source systems such Ontop [3], Morph [20], which implemented OBDA by querying relational data sources using virtual knowledge graph. These solutions, however, are not designed to query large-scale data sources, e.g., NoSQL stores or HDFS.

Our study's scope focuses on works that query large-scale data sources using OBDA. Optique [4] is an OBDA platform that accesses both static and streaming data. It implements a relational model (implicitly a TABULAR) as a virtual model while querying data sources such as SQL databases and other sources, e.g., CSV, and XML. There was no clear description of how Optique accesses NoSQL stores and distributed file systems (e.g., HDFS). Ontario [5] focuses on query rewriting, planning, and federation, with a strong stress on RDF data as input. Query plans are built and optimized based on a set of heuristics. The virtual model used by Ontario is the GRAPH model (explicitly an RDF). Squerall [6], a recent and close work to OPTIMA, leverages Big Data engines SAPRK and Presto to query on-the-fly of heterogeneous large data sources. The virtual data model imposed by Presto is TABULAR and does not offer users to control it; while SPARK can offer control over the virtual data model, Squerall uses DataFrame as a virtual model, which is TABULAR. However, the decision behind the virtual data model implemented by all these systems is based on use and flexibility and not on solid evidence to improve query processing. No work exists that (1) implements the different optimal virtual models and (2) selects the optimal one based on query behavior. For machine learning, some works [21–23] addressed the cost estimation of SPARQL queries to optimize query execution plan, e.g., performance prediction, however, all these approaches are designed for a single query on one single data source.

5. Conclusion

We implemented OPTIMA - an ontology-based big data access system that reduces query time execution by predicting the optimal virtual data model, GRAPH or TABULAR, based on query behavior. The effective deep learning model built on top of OPTIMA's architecture extracts significant features such as the query plan and operations and predicts the optimal virtual data model that has the lowest query execution time. The experiment showed a reduction in query execution time of over 40% for the TABULAR model and over 30% for the GRAPH model selection.

References

[1] N. Miloslavskaya and A. Tolstoy, Big data, fast data and data lake concepts, *Procedia Computer Science* **88** (2016), 300–305.

[2] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, in: *Journal on Data Semantics X*, Springer, 2008.

[3] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8**(3) (2017), 471–487.

[4] M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jiménez-Ruiz, D. Lanti, M. Rezk, G. Xiao, Ö. Özçep et al., Optique: Zooming in on big data, *Computer* **48**(3) (2015), 60–67.

[5] K.M. Endris, P.D. Rohde, M.-E. Vidal and S. Auer, Ontario: Federated query processing against a semantic data lake, in: *International Conference on Database and Expert Systems Applications*, Springer, 2019, pp. 379–395.

[6] M.N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer and J. Lehman, Squerall: Virtual Ontology-Based Access to Heterogeneous and Large Data Sources, *Proceedings of 18th International Semantic Web Conference* (2019).

[7] J.M. Hellerstein, Optimization techniques for queries with expensive methods, *ACM Transactions on Database Systems (TODS)* **23**(2) (1998), 113–157.

[8] S. Batra and C. Tyagi, Comparative analysis of relational and graph databases, *International Journal of Soft Computing and Engineering (IJSCE)* **2**(2) (2012), 509–512.

[9] C.B.B. Belmehdi, A. Khiat and N. Keskes, OPTIMA: Framework Selecting Optimal Virtual Model to Query Large Heterogeneous Data, in: *Big Data Analytics and Knowledge Discovery - 24th International Conference, DaWaK 2022, Vienna, Austria, August 22-24, 2022, Proceedings*, Vol. 13428, 2022, pp. 209–215.

[10] M. Hunger, From Relational to Graph: A Developer's Guide, *DZone, ed* (2016).

[11] M.N. Mami, S. Scerri, S. Auer and M.-E. Vidal, Towards Semantification of Big Data Technology, in: *Big Data Analytics and Knowledge Discovery*, S. Madria and T. Hara, eds, Springer International Publishing, Cham, 2016, pp. 376–390. ISBN 978-3-319-43946-4.

[12] M. Atay, Y. Sun, D. Liu, S. Lu and F. Fotouhi, Mapping XML data to relational data: A DOM-based approach, *arXiv preprint arXiv:1010.1746* (2010).

[13] J.E. Gonzalez, R.S. Xin, A. Dave, D. Crankshaw, M.J. Franklin and I. Stoica, {GraphX}: Graph Processing in a Distributed Dataflow Framework, in: *11th USENIX symposium on operating systems design and implementation (OSDI 14)*, 2014, pp. 599–613.

[14] R.F. van der Lans, Architecting the multi-purpose data lake with data virtualization, *Denodo whitepapers* (2018).

[15] D. Chatziantoniou and V. Kantere, Just-In-Time Modeling with DataMingler, in: *Proceedings of the ER Demos and Posters 2021 co-located with 40th International Conference on Conceptual Modeling (ER 2021), St. John's, NL, Canada, October 18-21, 2021*, R. Lukyanenko, B.M. Samuel and A. Sturm, eds, CEUR Workshop Proceedings, Vol. 2958, CEUR-WS.org, 2021, pp. 43–48. http://ceur-ws.org/Vol-2958/paper8.pdf.

[16] M.N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer and J. Lehmann, Uniform Access to Multiform Data Lakes using Semantic Technologies, in: *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services, iiWAS 2019, Munich, Germany, December 2-4, 2019*, ACM, 2019, pp. 313–322. doi:10.1145/3366030.3366054.

[17] D. Calvanese, G.D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro and R. Rosati, Ontologies and databases: The DL-Lite approach, in: *Reasoning Web International Summer School*, Springer, 2009, pp. 255–356.

[18] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi and D.F. Savo, The MASTRO system for ontology-based data access, *Semantic Web* **2**(1) (2011), 43–53.

[19] J.F. Sequeda and D.P. Miranker, Ultrawrap: SPARQL execution on relational data, *Journal of Web Semantics* **22** (2013), 19–39.

[20] F. Priyatna, O. Corcho and J. Sequeda, Formalisation and Experiences of R2RML-Based SPARQL to SQL Query Translation Using Morph, in: *Proceedings of the 23rd International Conference on World Wide Web*, 2014, pp. 479–490–.

[21] W.E. Zhang, Q.Z. Sheng, Y. Qin, K. Taylor and L. Yao, Learning-based SPARQL query performance modeling and prediction, *World Wide Web* **21**(4) (2018), 1015–1035. https://doi.org/10.1007/s11280-017-0498-1.

[22] R. Hasan and F. Gandon, A machine learning approach to sparql query performance prediction, in: *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, Vol. 1, IEEE, 2014, pp. 266–273.

[23] R. Singh, Inductive learning-based sparql query optimization, in: *Data Science and Intelligent Applications*, Springer, 2021, pp. 121–135.