Towards Analytics Aware Ontology Based Access to Static and Streaming Data*

E. Kharlamov[1] Y. Kotidis[2] T. Mailis[3] C. Neuenstadt[4] C. Nikolaou[1] Ö. Özçep[4] C. Svingos[3]
D. Zheleznyakov[1] S. Brandt[5] I. Horrocks[1] Y. Ioannidis[3] S. Lamparter[5] R. Möller[4]

[1]University of Oxford [2]Athens University of Economics and Business [3]University of Athens
[4]University of Lübeck [5]Siemens Corporate Technology

Abstract. Real-time analytics that requires integration and aggregation of heterogeneous and distributed streaming and static data is a typical task in many industrial scenarios such as diagnostics of turbines in Siemens. OBDA approach has a great potential to facilitate such tasks; however, it has a number of limitations in dealing with analytics that restrict its use in important industrial applications. Based on our experience with Siemens, we argue that in order to overcome those limitations OBDA should be extended and become analytics, source, and cost aware. In this work we propose such an extension. In particular, we propose an ontology, mapping, and query language for OBDA, where aggregate and other analytical functions are first class citizens. Moreover, we develop query optimisation techniques that allow to efficiently process analytical tasks over static and streaming data. We implement our approach in a system and evaluate our system with Siemens turbine data.

1 Introduction

Ontology Based Data Access (OBDA) [9] is an approach to access information stored in multiple datasources via an abstraction layer that mediates between the datasources and data consumers. This layer uses an *ontology* to provide a uniform conceptual schema that describes the problem domain of the underlying data independently of how and where the data is stored, and declarative *mappings* to specify how the ontology is related to the data by relating elements of the ontology to queries over datasources. The ontology and mappings are used to *transform* queries over ontologies, i.e., *ontological queries*, into *data queries* over datasources. As well as abstracting away from details of data storage and access, the ontology and mappings provide a declarative, modular and query-independent specification of both the conceptual model and its relationship to the data sources; this simplifies development and maintenance and allows for easy integration with existing data management infrastructure.

A number of systems that at least partially implement OBDA have been recently developed; they include D2RQ [7], Mastro [10], morph-RDB [38], Ontop [39], OntoQF [33], Ultrawrap [41], Virtuoso[1], and others [8, 17]. Some of them were successfully used in various applications including cultural heritage [13], governmental organisations [15], and industry [20, 21]. Despite their success, OBDA systems, however, are not tailored

* This work was partially funded by the EU project Optique (FP7-ICT-318338) and the EPSRC projects MaSI[3], DBOnto, and ED[3].

[1] `http://virtuoso.openlinksw.com/`

towards analytical tasks that are naturally based on data aggregation and correlation. Moreover, they offer a limited or no support for queries that combine streaming and static data. A typical scenario that requires both analytics and access to static and streaming data is diagnostics and monitoring of turbines in Siemens.

Siemens has several service centres dedicated to diagnostics of thousands of power-generation appliances located across the globe [21]. One typical task of such a centre is to detect in real-time potential faults of a turbine caused by, e.g., an undesirable pattern in temperature's behaviour within various components of the turbine. Consider a (simplified) example of such a task:

> *In a given turbine report all temperature sensors that are reliable, i.e., with the average score of validation tests at least 90%, and whose measurements within the last 10 min were similar, i.e., Pearson correlated by at least 0.75, to measurements reported last year by a reference sensor that had been functioning in a critical mode.*

This task requires to extract, aggregate, and correlate static data about the turbine's structure, streaming data produced by up to 2,000 sensors installed in different parts of the turbine, and historical operational data of the reference sensor stored in multiple datasources. Accomplishing such a task currently requires to pose a collection of hundreds of queries, the majority of which are semantically the same (they ask about temperature), but syntactically differ (they are over different schemata). Formulating and executing so many queries and then assembling the computed answers take up to 80% of the overall diagnostic time that Siemens engineers typically have to spend [21]. The use of ODBA, however, would allow to save a lot of this time since ontologies can help to 'hide' the technical details of *how* the data is produced, represented, and stored in data sources, and to show only *what* this data is about. Thus, one would be able to formulate this diagnostic task using only one ontological query instead of a collection of hundreds data queries that today have to be written or configured by IT specialists. Clearly, this collection of queries does not disappear: the OBDA query tranformation will automatically compute them from the the high-level ontological query using the ontology and mappings.

Siemens analytical tasks as the one in the example scenario typically make heavy use of aggregation and correlation functions as well as arithmetic operations. In our running example, the aggregation function min and the comparison operator $\geq$ are used to specify what makes a sensor reliable and to define a threshold for similarity. Performing such operations only in ontological queries, or only in data queries specified in the mappings is not satisfactory. In the case of ontological queries, all relevant values should be retrieved prior to performing grouping and arithmetic operations. This can be highly inefficient, as it fails to exploit source capabilities (e.g., access to pre-computed averages), and value retrieval may be slow and/or costly, e.g., when relevant values are stored remotely. Moreover, it adds to the complexity of application queries, and thus limits the benefits of the abstraction layer. In the case of source queries, aggregation functions and comparison operators may be used in mapping queries. This is brittle and inflexible, as values such as 90% and 0.75, which are used to define 'reliable sensor' and 'similarity', cannot be specified in the ontological query, but must be 'hard-wired' in the mappings, unless an appropriate extension to the query language or the ontology are developed. In order to address these issues, OBDA should become

analytics-aware by supporting declarative representations of basic analytics operations and using these to efficiently answer higher level queries.

In practice this requires enhancing OBDA technology with ontologies, mappings, and query languages capable of capturing operations used in analytics, but also extensive modification of OBDA query preprocessing components, i.e., reasoning and query transformation, to support these enhanced languages.

Moreover, analytical tasks as in the example scenario should typically be executed continuously in data intensive and highly distributed environments of streaming and static data. Efficiency of such execution requires non-trivial query optimisation. However, optimisations in existing OBDA systems are usually limited to minimisation of the textual size of the generated queries, e.g. [40], with little support for distributed query processing, and no support for optimisation for continuous queries over sequences of numerical data and, in particular, computation of data correlation and aggregation across static and streaming data. In order to address these issues, OBDA should become

source and cost aware by supporting both static and streaming data sources and offering a robust query planning component and indexing that can estimate the cost of different plans, and use such estimates to produce low-cost plans.

Note that the existence of materialised and pre-computed subqueries relevant to analytics within sources and archived historical data that should be correlated with current streaming data implies that there is a range of query plans which can differ dramatically with respect to data transfer and query execution time.

In this paper we make the first step to extend OBDA systems towards becoming analytics, source, and cost aware and thus meeting Siemens requirements for turbine diagnostics tasks. In particular, our contributions are the following:

- We proposed analytics-aware OBDA components, i.e., *(i)* ontology language *DL-Lite*$_{\mathcal{A}}^{\mathtt{agg}}$ that extends *DL-Lite*$_{\mathcal{A}}$ with aggregate functions as first class citizens, *(ii)* query language STARQL over ontologies that combine streaming and static data, and *(iii)* a mapping language relating *DL-Lite*$_{\mathcal{A}}^{\mathtt{agg}}$ vocabulary and STARQL constructs with relational queries over static and streaming data.
- We developed efficient query transformation techniques that allow to turn STARQL queries over *DL-Lite*$_{\mathcal{A}}^{\mathtt{agg}}$ ontologies, into data queries using our mappings.
- We developed source and cost aware *(i)* optimisation techniques for processing complex analytics on both static and streaming data, including adaptive indexing schemes and pre-computation of frequent aggregates on user queries, and *(ii)* elastic infrastructure that automatically distributes analytical computations and data over a computational cloud for fastet query execution.
- We implemented *(i)* a highly optimised engine ExaStream capable of handling complex streaming and static queries in real time, *(ii)* a dedicated STARQL2SQL$^{\oplus}$ translator that transforms STARQL queries into queries over static and streaming data, *(iii)* an integrated OBDA system that relies on our and third party components.
- We conducted a performance evaluation of our OBDA system with large scale Siemens simulated data using analytical tasks.

Due to space limitations we could not include all the relevant material in this paper and refer the reader to its online extended version for further details [26].

2 Analytics Aware OBDA for Static and Streaming Data

In this section we first introduce our analytics-aware ontology language *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ (Sec. 2.1) for capturing static aspects of the domain of interest. In *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ ontologies, aggregate functions are treated as first class citizens. Then, in Sec 2.2 we will introduce a query language STARQL that allows to combine static conjunctive queries over *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ with continuous diagnostic queries that involve simple combinations of time aware data attributes, time windows, and functions, e.g., correlations over streams of attribute values. Using STARQL queries one can retrieve entities, e.g., sensors, that pass two 'filters': static and continuous. In our running example a static 'filter' checks whether a sensor is reliable, while a continuous 'filter' checks whether the measurements of the sensor are Pearson correlated with the measurements of reference sensor. In Sec. 2.3 we will explain how to translate STARQL queries into data queries by mapping *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ concepts, properties, and attributes occurring in queries to database schemata and by mapping functions and constructs of STARQL continuous 'filters' into corresponding functions and constructs over databases. Finally, in Sec. 2.4 we discuss how to optimise resulting data queries.

2.1 Ontology Language

Our ontology language, *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$, is an extension of *DL-Lite*$_{\mathcal{A}}$ [9] with concepts that are based on aggregation of attribute values. The semantics for such concepts adapts the closed-world semantics [32]. The main reason why we rely on this semantics is to avoid the problem of empty answers for aggregate queries under the certain answers semantics [11, 30]. In *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ we distinguish between individuals and data values from countable sets Δ and D that intuitively correspond to the datatypes of RDF. We also distinguish between atomic roles P that denote binary relations between pairs of individuals, and attributes F that denote binary relations between individuals and data values. For simplicity of presentation we assume that D is the set of rational numbers. Let agg be an aggregate function, e.g., min, max, count, countd, sum, or avg, and let $\circ$ be a comparison predicate on rational numbers, e.g., $\geq, \leq, <, >, =$, or $\neq$.

***DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ Syntax.** The grammar for concepts and roles in *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ is as follows:

$$B \to A \mid \exists R, \quad C \to B \mid \exists F, \quad E \to \circ_r(\mathsf{agg}\ F), \quad R \to P \mid P^-,$$

where F, P, agg, and $\circ$ are as above, r is a rational number, A, B, C and E are atomic, basic, extended and aggregate concepts, respectively, and R is a basic role.

A *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ ontology $\mathcal{O}$ is a finite set of axioms. We consider two types of axioms: *aggregate* axioms of the form $E \sqsubseteq B$ and *regular* axioms that take one of the following forms: *(i) inclusions* of the form $C \sqsubseteq B$, $R_1 \sqsubseteq R_2$, and $F_1 \sqsubseteq F_2$, *(ii) functionality* axioms $(\mathsf{funct}\ R)$ and $(\mathsf{funct}\ F)$, *(iii)* or *denials* of the form $B_1 \sqcap B_2 \sqsubseteq \bot$, $R_1 \sqcap R_2 \sqsubseteq \bot$, and $F_1 \sqcap F_2 \sqsubseteq \bot$. As in *DL-Lite*$_{\mathcal{A}}$, a *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ *dataset* $\mathcal{D}$ is a finite set of assertions of the form: $A(a)$, $R(a, b)$, and $F(a, v)$.

We require that if $(\mathsf{funct}\ R)$ (resp., $(\mathsf{funct}\ F)$) is in $\mathcal{O}$, then $R' \sqsubseteq R$ (resp., $F' \sqsubseteq F$) is *not* in $\mathcal{O}$ for any R' (resp., F'). This syntactic condition, as wel as the fact that we do not allow concepts of the form $\exists F$ and aggregate concepts to appear on the right-hand side of inclusions ensure good computational properties of *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$. The former is inherited from *DL-Lite*$_{\mathcal{A}}$, while the latter can be shown using techniques of [32].

Consider the ontology capturing the reliability of sensors as in our running example:

$$precisionScore \sqsubseteq testScore, \quad \geq_{0.9} (\mathsf{min}\ testScore) \sqsubseteq Reliable, \tag{1}$$

where $Reliable$ is a concept, $precisionScore$ and $testScore$ are attributes, and finally $\geq_{0.9} (\mathsf{min}\ testScore)$ is an aggregate concept that captures individuals with one or more $testScore$ values whose minimum is at least 0.9.

***DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ Semantics.** We define the semantics of *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ in terms of first-order interpretations over the union of the countable domains Δ and D. We assume the unique name assumption and that constants are interpreted as themselves, i.e., $a^{\mathcal{I}} = a$ for each constant a; moreover, interpretations of regular concepts, roles, and attributes are defined as usual (see [9] for details) and for aggregate concepts as follows:

$$(\circ_r(\mathsf{agg}\ F))^{\mathcal{I}} = \{a \in \Delta \mid \mathsf{agg}\{\!\{v \in D \mid (a, v) \in F^{\mathcal{I}}\}\!\} \circ r\}.$$

Here $\{\!\{ \cdot \}\!\}$ denotes a multi-set. Similarly to [32], we say that an interpretation $\mathcal{I}$ is a *model* of $\mathcal{O} \cup \mathcal{D}$ if two conditions hold: *(i)* $\mathcal{I} \models \mathcal{O} \cup \mathcal{D}$, i.e., $\mathcal{I}$ is a first-order model of $\mathcal{O} \cup \mathcal{D}$ and *(ii)* $F^{\mathcal{I}} = \{(a, v) \mid F(a, v)$ is in the deductive closure of $\mathcal{D}$ with $\mathcal{O}\}$ for each attribute F. Here, by deductive closure of $\mathcal{D}$ with $\mathcal{O}$ we assume a dataset that can be obtained from $\mathcal{D}$ using the chasing procedure with $\mathcal{O}$, as described in [9]. One can show that for *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ satisfiability of $\mathcal{O} \cup \mathcal{D}$ can be checked in time polynomial in $|\mathcal{O} \cup \mathcal{D}|$.

As an example consider a dataset consisting of assertions: $precisionScore(s_1, 0.9)$, $testScore(s_2, 0.95)$, and $testScore(s_3, 0.5)$. Then, for every model $\mathcal{I}$ of these assertions and the axioms in Eq. (1), it holds that $(\geq_{0.9} (\mathsf{min}\ precisionScore))^{\mathcal{I}} = \{s_1\}$, $(\geq_{0.9} (\mathsf{min}\ testScore))^{\mathcal{I}} = \{s_1, s_2\}$, and thus $\{s_1, s_2\} \subseteq Reliable^{\mathcal{I}}$.

Query Answering. Let $\mathcal{Q}$ be the class of conjunctive queries over concepts, roles, and attributes, i.e., each query $q \in \mathcal{Q}$ is an expression of the form: $q(\vec{x})$:- $\mathsf{conj}(\vec{x})$, where q is of arity k, conj is a conjunction of atoms $A(u)$, $E(v)$, $R(w, z)$, or $F(w, z)$, and u, v, w, z are from $\vec{x}$. Following the standard approach for ontologies, we adapt certain answers semantics for query answering:

$$\mathsf{cert}(q, \mathcal{O}, \mathcal{D}) = \{\vec{t} \in (\Delta \cup D)^k \mid \mathcal{I} \models \mathsf{conj}(\vec{t}) \text{ for each model } \mathcal{I} \text{ of } \mathcal{O} \cup \mathcal{D}\}.$$

Continuing with our example, consider the query: $q(x)$:- $Reliable(x)$ that asks for reliable sensors. The set of certain answers $\mathsf{cert}(q, \mathcal{O}, \mathcal{D})$ for this q over the example ontology and dataset is $\{s_1, s_2\}$.

We note that by relying on Theorem 1 of [32] and the fact that each aggregate concept behaves like a *DL-Lite* closed predicate of [32], one can show that conjunctive query answering in *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ is tractable, assuming that computation of aggregate functions can be done in time polynomial in the size of the data (see more details in [26]). We also note that our aggregate concepts can be encoded as aggregate queries over attributes as soon as the latter are interpreted under the closed-world semantics. We argue, however, that in a number of applications, such as monitoring and diagnostics at Siemens [21], explicit aggregate concepts of *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ give us significant modelling and query formulation advantages (see more details in [26]).

```
1  PREFIX ex : <http://www.siemens.com/onto/gasturbine/>
2
3  CREATE PULSE examplePulse WITH START = NOW, FREQUENCY = 1min
4
5  CREATE STREAM StreamOfSensorsInCriticalMode AS
6  CONSTRUCT GRAPH NOW { ?sensor a :InCriticalMode }
7
8  FROM STATIC ONTOLOGY ex:sensorOntology, DATA ex:sensorStaticData
9  WHERE { ?sensor a ex:Reliable }
10
11 FROM STREAM   sensorMeasurements          [NOW - 1min, NOW]-> 1sec
12               referenceSensorMeasurements 1year <-[NOW - 1min, NOW]-> 1sec,
13 USING PULSE   examplePulse
14 SEQUENCE BY   StandardSequencing AS MergedSequenceOfMeasurementes
15 HAVING EXISTS i IN MergedSequenceOfMeasurementes
16       (GRAPH i { ?sensor ex:hasValue ?y. ex:refSensor ex:hasValue ?z })
17       HAVING PearsonCorrelation(?y, ?z) > 0.75
```

Fig. 1: Running example query expressed in STARQL

2.2 Query Language

STARQL is a query language over ontologies that allows to query both streaming and static data and supports not only standard aggregates such as count, avg, etc but also more advanced aggregation functions from our backend system such as Pearson correlation. In this section we illustrate on our running example the main language constructs and semantics of STARQL (see [26, 35] for more details on syntax and semantics of STARQL).

Each STARQL query takes as input a static *DL-Lite*$_{\mathcal{A}}^{\text{agg}}$ ontology and dataset as well as a set of live and historic streams. The output of the query is a stream of timestamped data assertions about objects that occur in the static input data and satisfy two kinds of filters: *(i)* a conjunctive query over the input static ontology and data and *(ii)* a diagnostic query over the input streaming data—which can be live and archived (i.e., static)— that may involve typical mathematical, statistical, and event pattern features needed in real-time diagnostic scenarios. The syntax of STARQL is inspired by the W3C standardised SPARQL query language; it also allows for nesting of queries. Moreover, STARQL has a formal semantics that combines open and closed-world reasoning and extends snapshot semantics for window operators [3] with sequencing semantics that can handle integrity constraints such as functionality assertions.

In Fig. 1 we present a STARQL query that captures the diagnostic task from our running example and uses concepts, roles, and attributes from our Siemens ontology [19, 21–25, 28] and Eq. (1). The query has three parts: declaration of the output stream (Lines 5 and 6), sub-query over the static data (Lines 8 and 9) that in the running example corresponds to '*return all temperature sensors that are reliable, i.e., with the average score of validation tests at least 90%*' and sub-query over the streaming data (Lines 11–17) that in the running example corresponds to '*whose measurements within the last 10 min Pearson correlate by at least 0.75 to measurements reported by a reference sensor last year*'. Moreover, in Line 1 there is declarations of the namespace that is used in the sub-queries, i.e., the URI of the Siemens ontology, and in Line 3 there is a declaration of the pulse of the streaming sub-query.

Regarding the semantics of STARQL, it combines open and closed-world reasoning and extends snapshot semantics for window operators [3] with sequencing semantics that can handle integrity constraints such as functionality assertions. In particular, the

window operator in combination with the sequencing operator provides a sequence of datasets on which temporal (state-based) reasoning can be applied. Every temporal dataset frequently produced by the window operator is converted to a sequence of (pure) datasets. The sequence strategy determines how the timestamped assertions are sequenced into datasets. In the case of the presented example in Fig. 1, the chosen sequencing method is *standard sequencing* assertions with the same timestamp are grouped into the same dataset. So, at every time point, one has a sequence of datasets on which temporal (state-based) reasoning can be applied. This is realised in STARQL by a sorted first-order logic template in which state stamped graph patterns are embedded. For evaluation of the time sequence, the graph patterns of the static `WHERE` clause are mixed into each state to join static and streamed data. Note that STARQL uses semantics with a real temporal dimension, where time is treated in a non-reified manner as an additional ontological dimension and not as ordinary attribute as, e.g., in SPARQLStream [8].

2.3 Mapping Language and Query Transformation

In this section we present how ontological STARQL queries, Q_{starql}, are transformed into semantically equivalent continuous queries, $Q_{\mathsf{sql}^\oplus}$, in the language SQL$^\oplus$. The latter language is an expressive extension of SQL with the appropriate operators for registering continuous queries against streams and updatable relations. The language's operators for handling temporal and streaming information are presented in Sec. 3.

As schematically illustrated in Eq. (2) below, during the transformation process the static conjunctive Q_{StatCQ} and streaming Q_{Stream} parts of Q_{starql}, are first independently *rewritten* using the 'rewrite' procedure that relies on the input ontology $\mathcal{O}$ into the union of static conjunctive queries Q'_{StatUCQ} and a new streaming query Q'_{Stream}, and then *unfolded* using the 'unfold' procedure that relies on the input mappings $\mathcal{M}$ into an aggregate SQL query Q''_{AggSQL} and a streaming SQL$^\oplus$ query Q''_{Stream} that together give an SQL$^\oplus$ query $Q_{\mathsf{sql}^\oplus}$, i.e., $Q_{\mathsf{sql}^\oplus} = \mathsf{unfold}(\mathsf{rewrite}(Q_{\mathsf{starql}}))$:

$$\begin{aligned} Q_{\mathsf{starql}} \approx Q_{\mathsf{StatCQ}} \wedge Q_{\mathsf{Stream}} &\xrightarrow[\mathcal{O}]{\mathsf{rewrite}} Q'_{\mathsf{StatUCQ}} \wedge Q'_{\mathsf{Stream}} \\ &\xrightarrow[\mathcal{M}]{\mathsf{unfold}} Q''_{\mathsf{AggSQL}} \wedge Q''_{\mathsf{Stream}} \approx Q_{\mathsf{sql}^\oplus}. \end{aligned} \tag{2}$$

In this process we use the rewriting procedure of [9], while the unfolding relies on mappings of three kinds: *(i) classical*: from concepts, roles, and attributes to SQL queries over relational schemas of static, streaming, or historical data, *(ii) aggregate*: from aggregate concepts to aggregate SQL queries over static data, and *(iii) streaming*: from the constructs of the streaming queries of STARQL into SQL$^\oplus$ queries over streaming and historical data. Our mapping language extends the one presented in [9] for the classical OBDA setting that allows only for the classical mappings.

We now illustrate our mappings as well as the whole query transformation procedure.

Transformation of Static Queries. We first show the transformation of the example static query that asks for reliable sensors. The rewriting of this query with the example ontology axioms from Equation (1) is the following query:

$$\mathsf{rewrite}(\mathit{Reliable}(x)) = \mathit{Reliable}(x) \vee (\geq_{0.9} (\mathsf{min}\ \mathit{testScore}))(x).$$

In order to unfold '$\mathsf{rewrite}(\mathit{Reliable}(x))$' we need both classical and aggregate mappings. Consider four classical mappings: one for the concept '$\mathit{Reliable}$' and three for the attributes '$\mathit{testScore}$' and '$\mathit{precisionScore}$', where sql_i are some SQL queries:

$$\begin{aligned} \mathit{Reliable}(x) &\leftarrow \mathsf{sql}_1(x), & \mathit{testScore}(x,y) &\leftarrow \mathsf{sql}_3(x,y), \\ \mathit{precisionScore}(x,y) &\leftarrow \mathsf{sql}_2(x,y), & \mathit{testScore}(x,y) &\leftarrow \mathsf{sql}_4(x,y). \end{aligned}$$

We define an aggregate mapping for a concept $E = \circ_r(\mathsf{agg}\ F)$ as $E(x) \leftarrow \mathsf{sql}_E(x)$, where $\mathsf{sql}_E(x)$ is an SQL query defined as

$$\mathsf{sql}_E(x) = \mathsf{SELECT}\ \ x\ \ \mathsf{FROM}\ \ \mathsf{SQL}_F(x,y)\ \ \mathsf{GROUP\ BY}\ \ x\ \ \mathsf{HAVING}\ \ \mathsf{agg}(y) \circ r \quad (3)$$

where $\mathsf{SQL}_F(x,y) = \mathsf{unfold}(\mathsf{rewrite}(F(x,y)))$, i.e., the SQL query obtained as the rewriting and unfolding of the attribute F. Thus, a mapping for our example aggregate concept $E = (\geq_{0.9}\ (\mathsf{min}\ \mathit{testScore}))$ is

$$\mathsf{sql}_E(x) = \mathsf{SELECT}\ x\ \mathsf{FROM}\ \mathsf{SQL}_{\mathit{testScore}}(x,y)\ \mathsf{GROUP\ BY}\ x\ \mathsf{HAVING}\ \mathsf{min}(y) \geq 0.9$$

where $\mathsf{SQL}_{\mathit{testScore}}(x,y) = \mathsf{sql}_2(x,y)\ \mathsf{UNION}\ \mathsf{sql}_3(x,y)\ \mathsf{UNION}\ \mathsf{sql}_4(x,y)$.

Finally, we obtain

$$\mathsf{unfold}(\mathsf{rewrite}(\mathit{Reliable}(x))) = \mathsf{sql}_1(x)\ \mathsf{UNION}\ \mathsf{sql}_E(x).$$

Note that one can encode *DL-Lite*$_{\mathcal{A}}^{\mathsf{agg}}$ aggregate concepts as standard *DL-Lite*$_{\mathcal{A}}$ concepts using mappings. We argue, however, that such an approach has practical disadvantages compared to ours as it would require to create a mapping for each aggregate concept that can be potentially used, thus overloading the system (see more details in [26]).

Transformation of Streaming Queries. The streaming part of a STARQL query may involve static concepts and roles such as *Rotor* and *testRotor* that are mapped into static data, and dynamic ones such as *hasValue* that are mapped into streaming data. Mappings for the static ontological vocabulary are classical and discussed above. Mappings for the dynamic vocabulary are composed from the mappings for attributes and the mapping schemata for STARQL query clauses and constructs. The mapping schemata rely on user defined functions of SQL$^{\oplus}$ and involve windows and sequencing parameters specified in a given STARQL query which make them dependent on time-based relations and temporal states. Note that the latter kind of mappings is not supported by traditional OBDA systems.

For instance, a mapping schema for the '`GRAPH i`' STARQL construct (see Line 16, Fig. 1) can be defined based on the following classical mapping that relates a dynamic attribute $\mathit{ex{:}hasVal}$ to the table *Msmt* about measurements that among others has attributes *sid* and *sval* for storing sensor IDs and measurement values:

$$\mathit{ex{:}hasVal}(\mathit{Msmt.sid}, \mathit{Msmt.sval}) \leftarrow \mathsf{SELECT}\ \mathit{Msmt.sid}, \mathit{Msmt.sval}\ \ \mathsf{FROM}\ \mathit{Msmt}.$$

The actual mapping schema for '`GRAPH i`' extends this mapping as following:

$$\begin{aligned} \mathsf{GRAPH}\ \ i\ \ \{?\mathit{sensor}\ \ \mathit{ex{:}hasVal}\ \ ?y\} \leftarrow\ & \mathsf{SELECT}\ \ \mathit{sid}\ \ \mathit{as}\ \ ?\mathit{sensor},\ \ \mathit{sval}\ \ \mathit{as}\ \ ?y \\ & \mathsf{FROM}\ \ \mathsf{Slice}(\mathit{Msmt}, i, r, sl, st), \end{aligned}$$

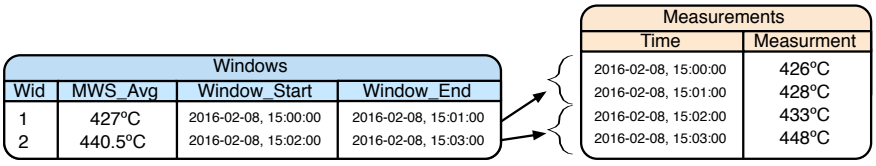


Fig. 2: Schema for storing archived streams and MWSs

where the left part of the schema contains an indexed graph triple pattern and the right part extends the mapping for $ex{:}hasVal$ by applying a function $Slice$ that describes the relevant finite slice of the stream $Msmt$ from which the triples in the i^{th} RDF graph in the sequence are produced and uses the parameters such as the window range r, the slide sl, the sequencing strategy st and the index i. (See [34] for further details.)

2.4 Query Optimisation

Since a STARQL query consists of analytical static and streaming parts, the result of its transformation by the rewrite and unfold procedures is an analytical data query that also consists of two parts and accesses information from both live streams and static data sources. A special form of static data are archived-streams that, though static in nature, accommodate temporal information that represents the evolution of a stream in time. Therefore, our analytical operations can be classified as: *(i) live-stream operations* that refer to analytical tasks involving exclusively live streams; *(ii) static-data operations* that refer to analytical tasks involving exclusively static information; *(iii) hybrid operations* that refer to analytical tasks involving live-streams and static data that usually originate from archived stream measurements. For static-data operations we rely on standard database optimisation techniques for aggregate functions. For live-stream and hybrid operations we developed a number of optimisation techniques and execution strategies.

A straightforward evaluation strategy on complex continuous queries containing *static-data operations* is for the query planner to compute the static analytical tasks ahead of the live-stream operations. The result on the static-data analysis will subsequently be used as a filter on the remaining streaming part of the query.

We will now discuss, using an example, the *Materialised Window Signatures* technique for hybrid operations. Consider the relational schema depicted in Fig. 2 which is adopted for storing archived streams and performing hybrid operations on them. The relational table `Measurements` represents the archived part of the stream and stores the temporal identifier (`Time`) of each measurement and the actual values (attribute `Measurement`). The relational table `Windows` identifies the windows that have appeared up till now based on the existing window-mechanism. It contains a unique identifier for each window (`Wid`) and the attributes that determine its starting and ending points (`Window_Start`, `Window_End`). The necessary indices that will facilitate the complex analytic computations are materialised. The depicted schema is flexible to query changes since it separates the windowing mechanism —which is query dependent— from the actual measurements.

In order to accelerate analytical tasks that include hybrid operations over archived streams, we facilitate precompution of frequently requested aggregates on each archived window. We name these precomputed summarisations as *Materialised Window Signatures* (*MWSs*). These MWSs are calculated when past windows are stored in the backend and are later utilised while performing complex calculations between these windows and a live stream. The summarisation values are determined by the analytics under consideration. E.g., for the computation of the Pearson correlation, we precompute the

average value and *standard deviation* on each archived window measurements; for the cosine similarity, we precompute the *Euclidean norm* of each archived window; for finding the absolute difference between the average values of the current and the archived windows, we precompute the *average* value, etc.

The selected MWSs are stored in the Windows relation with the use of additional columns. In Fig. 2 we see the MWS summary for the `avg` aggregate function being included in the relation as an attribute termed `MWS_Avg`. The application can easily modify the schema of this relation in order to add or drop MWSs, depending on the analytical workload.

When performing hybrid operations between the current and archived windows, some analytic operations can be directly computed based on their MWS values with no need to access the actual archived measurements. This provides significant benefits as it removes the need to perform a costly join operation between the live stream and the, potentially very large, `Measurements` relation. On the opposite, for calculations such as the Pearson correlation coefficient and the cosine similarity measures, we need to perform calculations that require the archived measurements as well, e.g., for computing cross-correlations or inner-products. Nevertheless, the MWS approach allows us to avoid recomputing some of the information on each archived window such as its *avg* value and *deviation* for the Pearson correlation coefficient, and the Euclidean norm of each archived window for the cosine similarity measure. Moreover, in case when there is a selective additional filter on the query (such as the avg value exceeds a threshold), by creating an index on the `MWS` attributes, we can often exclude large portions of the archived measurements from consideration, by taking advantage of the underlying index.

3 Implementation

In this section we discuss our system that implements the OBDA extensions proposed in Sec. 2. In Fig. 3 (Left), we present a general architecture of our system. On the application level one can formulate STARQL queries over analytics-aware ontologies and pass them to the query compilation module that performs query rewriting, unfolding, and optimisation. Query compilation components can access relevant information in the ontology for query rewriting, mappings for query unfolding, and source specifications for optimisation of data queries. Compiled data queries are sent to a query execution layer that performs distributed query evaluation over streaming and static data, post-processes query answers, and sends them back to applications. In the following we will discuss two main components of the system, namely, our dedicated STARQL2SQL$^{\oplus}$ translator that turns STARQL queries to SQL$^{\oplus}$ queries, and our native data-stream management system ExaStream that is in charge of data query optimisation and distributed query evaluation.

STARQL to SQL$^{\oplus}$ Translator. Our translator consists of several modules for transformation of various query components and we now give some highlights on how it works. The translator starts by turning the window operator of the input STARQL query and this results in a *slidingWindowView* on the backend system that consists of columns for defining *windowID* (as in Fig. 2) and *dataGraphID* based on the incoming data tuples. Our underlying data-stream management system ExaStream already provides *user defined functions* (UDFs) that automatically create the desired streaming views, e.g., the *timeSlidingWindow* function as discussed below in the ExaStream part of the section.

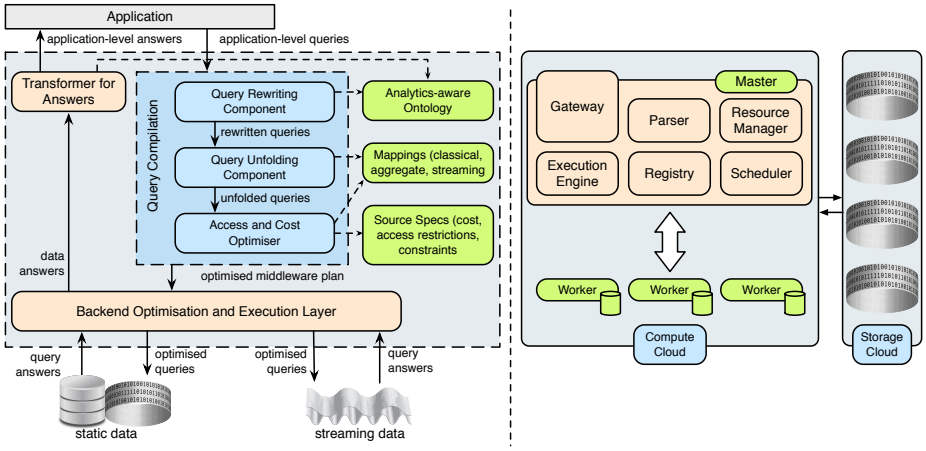


Fig. 3: (Left) General architecture. (Right) Distributed stream engine of EXASTREAM

The second important transformation step that we implemented is the transformation of the STARQL `HAVING` clause. In particular, we normalise the `HAVING` clause into a relational algebra normal form (RANF) and apply the described slicing technique illustrated in Sec. 2.3, where we unfold each state of the temporal sequence into slices of the *slidingWindowView*. For the rewriting and unfolding of each slice, we make use of available tools using the OBDA paradigm in the static case, i.e., the Ontop framework [39]. After unfolding, we join all states together based on their temporal relations given in the `HAVING` sequence.

EXASTREAM Data-Stream Management System. Data queries produced by the STARQL2SQL$^{\oplus}$ translation, are handled by EXASTREAM which is embedded in EXAREME, a system for elastic large-scale dataflow processing in the cloud [29, 42].

EXASTREAM is built as a streaming extension of the SQLite database engine, taking advantage of existing Database Management technologies and optimisations. It provides the declarative language SQL$^{\oplus}$ for querying data streams and relations. SQL$^{\oplus}$ extends SQL with *UDF*s that incorporate the algorithmic logic for transforming SQLite into a *Data Stream Management Systems* (*DSMS*). E.g., the *timeSlidingWindow* operator groups tuples from the same time window and associates them with a unique window id. In contrast to other *DSMS*s, the user does not need to consider low-level details of query execution. Instead, the system's *query planner* is responsible for choosing an optimal plan depending on the query, the available stream/static data sources, and the execution environment.

EXASTREAM system exploits parallelism in order to accelerate the process of analytical tasks over thousands of stream and static sources. It manages an elastic cloud infrastructure and dynamically distributes queries and data (including both streams and static tables) to multiple worker nodes that process them in parallel. The architecture of EXASTREAM's distributed stream engine is presented in Fig. 3 (Right). One can see that queries are registered through the Asynchronous Gateway Server. Each registered query passes through the EXASTREAM parser and then is fed to the Scheduler module. The Scheduler places the stream and relational operators on worker nodes based on the node's load. These operators are executed by a Stream Engine instance running on each node.

4 Evaluation

The aim of our evaluation is to study how the MWS technique and query distribution to multiple workers accelerate the overall execution time of analytic queries that correlate a live stream with multiple archived stream records.

Evaluation Setting. We deployed our system to the Okeanos Cloud Infrastructure[2]. and used up to 16 virtual machines (VMs) each having a 2.66 GHz processor with 4GB of main memory. We used streaming and static data that contains measurements produced by $100,000$ thermocouple sensors installed in 950 Siemens power generating turbines. For our experiments, we used three *test queries* calculating the similarity between the current live stream window and 100,000 archived ones. In each of the test queries we fixed the window size to 1 hour which corresponds to 60 tuples of measurements per window. The first query is based on the one from our running example (see Fig. 1) which we modified so that it can correlate a live stream with a varying number of archived streams. Recall that this query evaluates window measurements similarity based on the Pearson correlation. The other two queries are variations of the first one where, instead of the Pearson correlation, they compute similarity based on either the *average* or the *minimum* values within a window. We defined such similarities between vectors (of measurements) $\vec{w}$ and $\vec{v}$ as follows: $|\text{avg}(\vec{w}) - \text{avg}(\vec{v})| < 10°C$ and $|\text{min}(\vec{w}) - \text{min}(\vec{v})| < 10°C$. The archived streams windows are stored in the `Measurements` relation, against which the current stream is compared.

MWS Optimisation. This set of experiments is devised to show how the MWS optimisation affects the query's response time. We executed each of the three test queries on a single VM-worker with and without the MWS optimisation. In Fig. 4 (Left) we present the results of our experiments. The reported time is the average of 15 consecutive live-stream execution cycles. The horizontal axis displays the three test queries with and without the MWS optimisation, while the vertical axis measures the time it takes to process 1 live-stream window against all the archived ones. This time is divided to the time it takes to join the live stream and the `Measurements` relation and the time it takes to perform the actual computations. Observe that the MWS optimisation reduces the time for the Pearson query by 8.18%. This is attributed to the fact that some computations (such as the avg and standard deviation values) are already available in the `Winodws` relation and are, thus, omitted. Nevertheless, the join operation between the live stream and the very large `Measurements` relation that takes 69.58% of the overall query execution time can not be avoided. For the other two queries, we not only reduce the CPU overhead of the query, but the optimiser further prunes this join from the query plan as it is no longer necessary. Thus, for these queries, the benefits of the MWS technique are substantial.

Intra-query Parallelism. Since the MWS optimisation substantially accelerates query execution for the two test queries that rely on average and minimum similarities, query distribution would not offer extra benefit, and thus these queries were not used in the second experiment. For complex analytics such as the Pearson correlation that necessitates access to the archived windows, the EXASTREAM backend permits us to accelerate queries by distributing the load among multiple worker nodes. In the second experiment we use

[2] `https://okeanos.grnet.gr/home/`

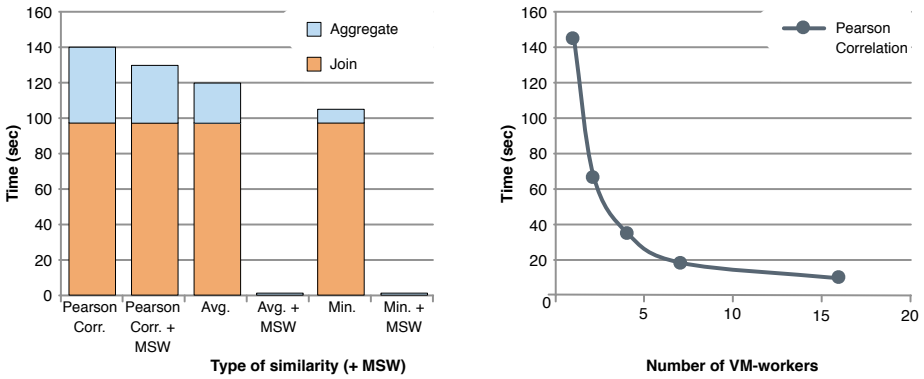


Fig. 4: (Left) Effect of MWS optimisation (Right) Effect of intra-query parallelism

the same setting as before for the Pearson computation without the MWS technique, but we vary this time the number of available workers from 1 to 16. In Fig. 4 (Right), one can observe a significant decrease in the overall query execution time as the number of VM-workers increases. EXASTREAM distributes the `Measurements` relation between different worker nodes. Each node computes the Pearson coefficient between its subset of archived measurements and the live stream. As the number of archived windows is much greater than the number of available workers, intra-query parallelism results is significant decrease to the time required to perform the join operation.

To conclude this section, we note that MWSs gave us significant improvements of query execution time for all test queries and parallelism would be essential in the cases where MWSs do not help in avoiding the high cost of query joins since it allows to run the join computation in parallel. Due to space limitations, we do not include an experiment examining the query execution times w.r.t. the number of archived windows. Nevertheless, based on our observations, scaling up the number of archived windows by a factor of n has about the same effect as scaling down the number of workers by $1/n$.

5 Related Work

OBDA System. Our proposed approach extends existing OBDA systems since they either assume that data is in (static) relational DBs, e.g [15, 39], or streaming, e.g., [8, 17], but not of both kinds. Moreover, we are different from existing solutions for unified processing of streaming and static semantic data e.g. [36], since they assume that data is natively in RDF while we assume that the data is relational and mapped to RDF.

Ontology language. The semantic similarities of *DL-Lite*$_{\mathcal{A}}^{\text{agg}}$ to other works have been covered in Sec. 2. Syntactically, the aggregate concepts of *DL-Lite*$_{\mathcal{A}}^{\text{agg}}$ have counterpart concepts, named local range restrictions (denoted by $\forall F.T$) in *DL-Lite*$_{\mathcal{A}}$ [4]. However, for purposes of rewritability, these concepts are not allowed on the left-hand side of inclusion axioms as we have done for *DL-Lite*$_{\mathcal{A}}^{\text{agg}}$, but only in a very restrictive semantic/syntactic way. The semantics of *DL-Lite*$_{\mathcal{A}}^{\text{agg}}$ for aggregate concepts is very similar to the epistemic semantics proposed in [11] for evaluating conjunctive queries involving aggregate functions. A different semantics based on minimality has been considered in [30]. Concepts based on aggregates functions were considered in [5] for languages $\mathcal{ALC}$ and $\mathcal{EL}$ with concrete domains, but they did not study the problem of query answering.

Query language. While already several approaches for RDF stream reasoning engines do exist, e.g., CSPARQL [6], RSP-QL [1] or CQELS [37], only one of them supports an

ontology based data access approach, namely SPARQLstream [8]. In comparison to this approach, which also uses a native inclusion of aggregation functions, STARQL offers more advanced user defined functions from the backend system like Pearson correlation.

Data Stream Management System. One of the leading edges in database management systems is to extend the relational model to support for continuous queries based on declarative languages analogous to SQL. Following this approach, systems such as TelegraphCQ [14], STREAM [2], and Aurora [16] take advantage of existing Database Management technologies, optimisations, and implementations developed over 30 years of research. In the era of big data and cloud computing, a different class of *DSMS* has emerged. Systems such as Storm and Flink offer an API that allows the user to submit dataflows of user defined operators. EXASTREAM unifies these two different approaches by allowing to describe in a declarative way complex dataflows of (possibly user-defined) operators. Moreover, the Materialised Window Signature summarisation, implemented in EXASTREAM, is inspired from data warehousing techniques for maintaining selected aggregates on stored datasets [18, 31]. We adjusted these technique for complex analytics that blend streaming with static data.

6 Conclusion, Lessons Learned, and Future Work

We see our work as a first step towards the development of a solid theory and new full-fledged systems in the space of analytics-aware ontology-based access to data that is stored in different formats such as static relational, streaming, etc. To this end we proposed ontology, query, and mapping languages that are capable of supporting analytical tasks common for Siemens turbine diagnostics. Moreover, we developed a number of backend optimisation techniques that allow such tasks to be accomplished in reasonable time as we have demonstrated on large scale Siemens data.

The lessons we have learned so far are the encouraging evaluation results over the Siemens turbine data (presented in Section 4). Since our work is a part of an ongoing project that involves Siemens, we plan to continue implementation and then deployment of our solution in Siemens. This will give us an opportunity to do further performance evaluation as well as to conduct user studies.

Finally, there is a number of important further research directions that we plan to explore. On the side of analytics-aware ontologies, we plan to explore bag instead of set semantics for ontologies since bag semantics is natural and important in analytical tasks; we also plan to investivate how to support evolution of such ontologies [12, 27] since OBDA systems are dynamic by its nature. On the side of analytics-aware queries, an important further direction is to align them with the terminology of the W3C RDF Data Cube Vocabulary and to provide additional optimisations after the alignment. As for query optimisation techniques, exploring approximation algorithms for fast computation of complex analytics between live and archived streams is particularly important. That is because these algorithms usually provide quality guarantees about the results and in the average case require much less computation. Thus, we intend to examine their effectiveness in combination with the MWS approach.

7 References

[1] D. D. Aglio, E. D. Valle, J.-P. Calbimonte, and O. Corcho. RSP-QL Semantics: A Unifying Query Model to Explain Heterogeneity of RDF Stream Processing Systems. *IJSWIS* 10(4) (2015).

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. STREAM: The Stanford Stream Data Manager. *SIGMOD*. 2003.

[3] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query execution. *VLDBJ* 15.2 (2006).

[4] A. Artale, V. Ryzhikov, and R. Kontchakov. DL-Lite with Attributes and Datatypes. *ECAI*. 2012.

[5] F. Baader and U. Sattler. Description Logics with Aggregates and Concrete Domains. *IS* 28.8 (2003).

[6] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. C-SPARQL: A Continuous Query Language for RDF Data Streams. *Int. J. Sem. Computing* 4.1 (2010).

[7] C. Bizer and A. Seaborne. D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs. *ISWC*. 2004.

[8] J. Calbimonte, Ó. Corcho, and A. J. G. Gray. Enabling Ontology-Based Access to Streaming Data Sources. *ISWC*. 2010.

[9] D. Calvanese, G. Giacomo, and D. Lembo. Ontologies and Databases: The DL-Lite Approach. *Reas. Web*. 2009.

[10] 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).

[11] D. Calvanese, E. Kharlamov, W. Nutt, and C. Thorne. Aggregate Queries Over Ontologies. *ONISW*. Oct. 2008.

[12] D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. Evolution of DL-Lite Knowledge Bases. *ISWC*. 2010.

[13] D. Calvanese, P. Liuzzo, A. Mosca, J. Remesal, M. Rezk, and G. Rull. Ontology-based Data Integration in EPNet: Production and Distribution of Food During the Roman Empire. *Eng. Appl. of AI* 51 (2016).

[14] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing. *SIGMOD*. 2003.

[15] C. Civili, M. Console, G. De Giacomo, D. Lembo, M. Lenzerini, L. Lepore, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, V. Santarelli, and D. F. Savo. MASTRO STUDIO: Managing Ontology-Based Data Access Applications. *PVLDB* 6.12 (2013).

[16] D. Abadi, D. Carney, et al. Aurora: A Data Stream Management System. *SIGMOD*. 2003.

[17] L. Fischer, T. Scharrenbach, and A. Bernstein. Scalable Linked Data Stream Processing via Network-Aware Workload Scheduling. *SSWKBS@ISWC*. 2013.

[18] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab, and Sub-totals. *Data mining and knowl. discovery* 1.1 (1997).

[19] E. Kharlamov, S. Brandt, E. Jimenez-Ruiz, Y. Kotidis, S. Lamparter, T. Mailis, C. Neuenstadt, Ö. Özçep, C. Pinkel, C. Svingos, D. Zheleznyakov, I. Horrocks, Y. Ioannidis, and R. Möller. Ontology-Based Integration of Streaming and Static Relational Data with Optique. *SIGMOD* (2016).

[20] E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. L. C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, E. Bjørge, and I. Horrocks. Enabling Ontology Based Access at an Oil and Gas Company Statoil. *ISWC*. 2015.

[21] E. Kharlamov, N. Solomakhina, Ö. L. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, and S. Watson. How Semantic Technologies Can Enhance Data Access at Siemens Energy. *ISWC*. 2014.

[22] E. Kharlamov, S. Brandt, M. Giese, E. Jiménez-Ruiz, Y. Kotidis, S. Lamparter, T. Mailis, C. Neuenstadt, Ö. L. Özçep, C. Pinkel, A. Soylu, C. Svingos, D. Zheleznyakov, I. Horrocks, Y. E. Ioannidis, R. Möller, and A. Waaler. Enabling semantic access to static and streaming distributed data with optique: demo. *DEBS*. 2016.

[23] E. Kharlamov, S. Brandt, M. Giese, E. Jiménez-Ruiz, S. Lamparter, C. Neuenstadt, Ö. L. Özçep, C. Pinkel, A. Soylu, D. Zheleznyakov, M. Roshchin, S. Watson, and I. Horrocks. Semantic Access to Siemens Streaming Data: the Optique Way. *ISWC (P&D)*. 2015.

[24] E. Kharlamov, E. Jiménez-Ruiz, C. Pinkel, M. Rezk, M. G. Skjæveland, A. Soylu, G. Xiao, D. Zheleznyakov, M. Giese, I. Horrocks, and A. Waaler. Optique: Ontology-Based Data Access Platform. *ISWC (P&D)*. 2015.

[25] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, P. Haase, I. Horrocks, H. Kllapi, M. Koubarakis, Ö. L. Özçep, M. Rodriguez-Muro, R. Rosati, M. Schmidt, R. Schlatte, A. Soylu, and A. Waaler. Optique: Towards OBDA Systems for Industry. *ESWC (Selected Papers)*. 2013.

[26] E. Kharlamov, Y. Kotidis, T. Mailis, C. Neuenstadt, C. Nicolaou, Ö. Özçep, C. Svingos, D. Zheleznyakov, S. Brandt, I. Horrocks, Y. Ioannidis, S. Lamparter, and R. Möller. Towards Analytics Aware Ontology Based Access to Static and Streaming Data (Extended Version). *CoRR* (2016).

[27] E. Kharlamov, D. Zheleznyakov, and D. Calvanese. Capturing model-based ontology evolution at the instance level: The case of DL-Lite. *J. Comput. Syst. Sci.* 79.6 (2013).

[28] E. Kharlamov et al. Optique 1.0: Semantic Access to Big Data: The Case of Norwegian Petroleum Directorate FactPages. *ISWC (P&D)*. 2013.

[29] H. Kllapi, P. Sakkos, A. Delis, D. Gunopulos, and Y. Ioannidis. Elastic Processing of Analytical Query Workloads on IaaS Clouds. *arXiv*. 2015.

[30] E. V. Kostylev and J. L. Reutter. Complexity of Answering Counting Aggregate Queries Over DL-Lite. *J. of Web Sem.* 33 (2015).

[31] Y. Kotidis and N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. *SIGMOD*. 1999.

[32] C. Lutz, I. Seylan, and F. Wolter. Mixing Open and Closed World Assumption in Ontology-Based Data Access: Non-Uniform Data Complexity. *DL*. 2012.

[33] K. Munir, M. Odeh, and R. McClatchey. Ontology-Driven Relational Query Formulation Using the Semantic and Assertional Capabilities of OWL-DL. *KBS* 35 (2012).

[34] C. Neuenstadt, R. Möller, and Ö. L. Özçep. OBDA for Temporal Querying and Streams with STARQL. *HiDeSt*. 2015.

[35] Ö. Özçep, R. Möller, and C. Neuenstadt. A Stream-Temporal Query Language for Ontology Based Data Access. *KI*. 2014.

[36] D. L. Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. *ISWC*. 2011.

[37] D. Le-Phuoc, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter, and M. Fink. Linked Stream Data Processing Engines: Facts and Figures. *ISWC*. 2012.

[38] F. Priyatna, O. Corcho, and J. Sequeda. Formalisation and Experiences of R2RML-Based SPARQL to SQL Query Translation Using Morph. *WWW*. 2014.

[39] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-Based Data Access: Ontop of Databases. *ISWC*. 2013.

[40] M. Rodrıguez-Muro and D. Calvanese. High Performance Query Answering Over DL-Lite Ontologies. *KR*. 2012.

[41] J. Sequeda and D. P. Miranker. Ultrawrap: SPARQL Execution on Relational Data. *JWS* 22 (2013).

[42] M. M. Tsangaris, G. Kakaletris, H. Kllapi, G. Papanikos, F. Pentaris, P. Polydoras, E. Sitaridi, V. Stoumpos, and Y. E. Ioannidis. Dataflow Processing and Optimization on Grid and Cloud Infrastructures. *IEEE Data Eng. Bull.* 32.1 (2009).