The NPD Benchmark: Reality Check for OBDA Systems

Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese
Faculty of Computer Science, Free University of Bozen-Bolzano
Piazza Domenicani 3, Bolzano, Italy
{dlanti,mrezk,xiao,calvanese}@inf.unibz.it

ABSTRACT

In the last decades we moved from a world in which an enterprise had one central database—rather small for todays' standards—to a world in which many different—and big—databases must interact and operate, providing the user an integrated and understandable view of the data. Ontology-Based Data Access (OBDA) is becoming a popular approach to cope with this new scenario. OBDA separates the user from the data sources by means of a conceptual view of the data (ontology) that provides clients with a convenient query vocabulary. The ontology is connected to the data sources through a declarative specification given in terms of mappings. Although prototype OBDA systems providing the ability to answer SPARQL queries over the ontology are available, a significant challenge remains when it comes to use these systems in industrial environments: performance. To properly evaluate OBDA systems, benchmarks tailored towards the requirements in this setting are needed. In this work, we propose a novel benchmark for OBDA systems based on real data coming from the oil industry: the Norwegian Petroleum Directorate (NPD) FactPages. Our benchmark comes with novel techniques to generate, from the NPD data, datasets of increasing size, taking into account the requirements dictated by the OBDA setting. We validate our benchmark on significant OBDA systems, showing that it is more adequate than previous benchmarks not tailored for OBDA.

1. INTRODUCTION

In the last decades we moved from a world in which an enterprise had one central database, to a world in which many different databases must interact and operate, providing the user an integrated view of the data. In this new setting five research areas in the database community became critical [7]: (i) scalable big/fast data infrastructures; (ii) ability to cope with diversity in the data management landscape; (iii) end-to-end processing and understanding of data; (iv) cloud services; and (v) managing the diverse roles of people in the data life cycle. Since the mid 2000s, *Ontology-Based Data Access* (OBDA) has become a popular approach used in three of these five areas— namely (ii), (iii), and (v).

In OBDA, queries are posed over a high-level conceptual view, and then translated into queries over a potentially very large (usually relational and federated) data source. The conceptual layer is given in the form of an ontology that defines a shared vocabulary, hides the structure of the data sources, and can enrich incomplete data with background knowledge. The ontology is connected to the data sources through a declarative specification given in terms of mappings that relate each (class and property) symbol in the ontology to a (SQL) view over (possibly federated) data. The W3C standard R2RML [14] was created with the goal of providing a standardized language for the specification of mappings in the OBDA setting. The ontology together with the mappings exposes a virtual instance (RDF graph) that can be queried using SPARQL, the standard query language in the Semantic Web community.

To make OBDA useful in an industrial setting, OBDA systems must provide answers in a reasonable amount of time, especially in the context of Big Data. However, most research in academia has focused on correct SPARQL-to-SQL translations, and expressivity of the ontology/mapping languages. Little effort (to the best of our knowledge) has been spent in systematically evaluating the performance of OBDA systems. To properly evaluate such performance, benchmarks tailored towards the requirements in this setting are needed. In particular, the benchmark should resemble a typical real-world industrial scenario in terms of the size of the data set, the complexity of the ontology, and the complexity of the queries. In this work, we propose a novel benchmark for OBDA systems based on the Norwegian Petroleum Directorate (NPD) FactPages [1]. The NPD FactPages contains information regarding the petroleum activities on the Norwegian continental shelf. Such information is actively used by oil companies, such as Statoil. The Factpages are synchronized with the NPD's databases on a daily basis. The NPD Ontology [32] has been mapped to the NPD FactPages and stored in a relational database [2]z; whereas the queries over such an ontology have been formulated by domain experts starting from an informal set of questions provided by regular users of the FactPages.

The contributions of this paper are as follows: (1) we identify requirements for benchmarking of OBDA systems in a real world scenario; (2) we identify requirements for data generation in the setting of OBDA; (3) we propose a benchmark that is compliant with the requirements identified; (4) we provide a data generator for OBDA together with an automatized testing platform; and (5) an extensive evaluation using state-of-the-art OBDA systems and triple stores, revealing strength and weaknesses of OBDA.

This work extends the previous workshop publications [12, 23] with a larger and more challenging query set—that includes aggregate queries, new experiments, and an automatized testing platform. This new query set highlights the importance of semantic query optimisation in the SPARQL-to-SQL translation phase.

The rest of the paper is structured as follows. In Section 2, we briefly survey other works related to benchmarking. In Section 3, we present the necessary requirements for an OBDA benchmark. In Section 4, we discuss the requirements for an OBDA instance generator. In Section 5, we present the NPD benchmark[1] and an associated relational database generator that gives rise to a virtual instance through the mapping; we call our generator *Virtual Instance Generator* (VIG). In Section 5.2, we perform a qualitative analysis of the virtual instances obtained using VIG. In Section 6, we describe a set of experiments performed using our benchmark over OBDA systems and triple stores. We conclude in Section 7.

2. RELATED WORK

Benchmarks are used to assess the quality of a system against a number of measures related to its design goals. Although OBDA systems have recently gained popularity, and the interest of a number of important enterprises like Siemens or Statoil (c.f. Optique Project[2]), no benchmark has yet been proposed in this setting. Although there are no guidelines nor benchmarks specific for OBDA, one must observe that these systems integrate both well-established database technologies and Semantic Web features. Driven by this observation, and given that both databases and knowledge-based systems have a vast literature on benchmarking, a natural starting point for deriving requirements for an OBDA benchmark is a synthesis of the requirements coming from both of these worlds.

For the databases world, two of the most popular benchmarks are the *Wisconsin Benchmark* [15] and the *TPC* benchmark [3]. The Wisconsin benchmark specifies a single relation, and columns with different duplicates ratios allow one to easily manipulate the selectivity of the test queries. The *TPC* Benchmark comes in different flavors so as to test database systems in several popular scenarios, like transactions in an order-entry environment (TPC-C), or a brokerage firm with related customers (TPC-E). These benchmarks gained popularity for a number of reasons, prominently because they capture concrete use-cases coming from industry, they are simple to understand and run, and they provide metrics that allow one to clearly identify winners and losers (e.g., cost per transaction or query mixes per hour).

For the Semantic Web world, the situation is much less standardized, and a high number of benchmarks have been proposed. The most popular ones are LUBM [18], and BSBM [9], which are rather simple in the sense that they come either with a simple ontology, or with no ontology at all. These benchmarks do not allow one to properly test the performance of the reasoners in the context of complex and expressive ontologies—which are the vast majority when it comes to real-world applications. This aspect was pointed out in [37], where the authors proposed an extension of the LUBM benchmark (called UOBM) in order to overcome these limitations. Rather than proposing a new benchmark, [19] identifies a number of requirements for benchmarking knowledge base systems. In this work we follow a similar scheme, as we first identify a number of key requirements in OBDA benchmarking and then we validate our benchmark against those requirements.

A recent and relevant effort for what concerns benchmarks in the semantic web context comes from the DBPedia benchmark [26]. In this benchmark, the authors propose a number of key features, like a data generator to produce "realistic" instances of increasing sizes, a number of real-world queries gathered from the DBPedia SPARQL endpoint, and the DBPedia ontology. Although this is an extremely valuable effort in the context of knowledge base systems, yet there are a number of characteristics that make the DBPedia benchmark unsuitable for OBDA benchmarking (see Section 3).

The last effort in order of time comes from the attempt to create a council like TPC in the context of graph-like data management technologies, like Graph Data Base Management Systems or systems based on RDF graphs. The council is called LDBC [3], and it has so far produced two benchmarks related to data publishing and social use-cases. This is a remarkable effort, however the ontologies used in these benchmarks are in RDFS, rather than full OWL 2 QL; therefore, they might miss to test important OBDA-specific pitfalls, like reasoning w.r.t. existentials [29].

3. REQUIREMENTS FOR BENCHMARKING OBDA

In this section we study the requirements that are necessary for a benchmark to evaluate OBDA systems. In order to define these requirements, we first recall that the three fundamental components of such systems are: *(i)* the *conceptual layer* constituted by the ontology; *(ii)* the *data layer* provided by the data sources; and *(iii)* the *mapping layer* containing the declarative specification relating each (class and property) symbol in the ontology to an (SQL) view over (possibly federated) data. It is this mapping layer that decouples the virtual instance being queried, from the physical data stored in the data sources. Observe that triple stores cannot be considered as full-fledged OBDA systems, since they do not make a distinction between physical and virtual layer. However, given that both, OBDA systems and triple stores, are considered as (usually SPARQL) query answering systems, we consider it important that a benchmark for OBDA can also be used to evaluate triple stores. Also, since one of the components of an OBDA system is an ontology, the requirements we identify include those to evaluate general knowledge based systems [25, 19, 36]. However, due to the additional components, there are also notable differences.

Typically OBDA systems follow the workflow below for query answering:

1. *Starting phase*. The system loads the ontology, the mappings, and performs some auxiliary tasks needed to process/answer queries in a later stage. Depending on the system, this phase might be critical, since it might include some reasoning tasks, for example *inference materialization* or the embedding of the inferences into the mappings (*T-mappings* [28]).

2. *Query rewriting phase*. The input query is *rewritten* to a (maybe more complex) query that takes into account the inferences induced by the intensional level of the ontology (we forward the interested reader to [10, 21]).

3. *Query translation (unfolding) phase*. The rewritten query is transformed into a query over the data sources. This is the phase where the mapping layer comes into play [27].

4. *Query execution phase*. The data query is executed over the original data source, answers are produced according to the data source schema, and are translated into answers in terms of the ontology vocabulary and RDF data types, thus obtaining an answer to the original input query.

Note that a variation of the above workflow has actually been proposed in [25], but without identifying a distinct starting phase, and singling out a result translation phase from query execution. It is critical to notice that although optimisation is not mentioned in this

[1] https://github.com/ontop/npd-benchmark/

[2] http://www.optique-project.eu/

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

Table 1: Measures for OBDA

Performance Metrics		
name	triple store	related to phase
Loading Time	(**T**)	1
Rewriting Time	$(\mathbf{T}^*)$	2
Unfolding Time	—	3
Query execution time	(**T**)	4
Overall response time	(**T**)	2, 3, 4
Quality Metrics		
Simplicity R Query	$(\mathbf{T}^*)$	2
Simplicity U Query	—	3
Weight of R+U	$(\mathbf{T}^*)$	2, 3, 4

workflow, it is the most challenging part in the query translation process, and *definitely essential* to make OBDA applicable in production environments.

There are several approaches to deal with Phase 2 [21, 35]. The most challenging task in this phase is to deal with existentials in the right-hand side of ontology axioms. These axioms infer unnamed individuals in the virtual instance that cannot be retrieved as part of the answer, but can affect the evaluation of the query. An approach that has proved to produce good results in practice is the *tree-witness rewriting* technique, for which we refer to [21]. For us, it is only important to observe that *tree-witnesses* lead to an extension of the original query to account for matching in the existentially implied part of the virtual instance. Below, we take the number of tree-witnesses identified in Phase 2 as one of the parameters to measure the complexity of the combination ontology/query. Since existentials do not occur very often in practice [21], and can produce an exponential blow-up in the query size, some systems allow one to turn off the part of Phase 2 that deals with *reasoning with respect to existentials*.

Ideally, an OBDA benchmark should provide *meaningful* measures for each of these phases. Unfortunately, such a fine-grained analysis is not always possible, for instance because the system comes as a black-box with proprietary code with no APIs providing the necessary information, e.g., the access to the rewritten query; or because the system combines one or more phases, e.g., query rewriting and query translation. Based on the above phases, we identify in Table 1 the measures important for *evaluating* OBDA systems. The meaning of the *Performance Metrics* should be clear from their names; instead, we will give a brief explanation of the meaning of the *Quality Metrics*:

- *Simplicity R Query*. Simplicity of the rewritten query in terms of language dependent measures, like the *number of rules* in case the rewritten query is a datalog program. In addition, one can include system-dependent features, e.g., # of tree-witnesses in *Ontop*.

- *Simplicity U Query*. This measures the simplicity of the query over the data source, including relevant SQL-specific metrics like the number of joins/left-join, the number of inner queries, etc.

- *Weight of R+U*. It is the cost of the construction of the SQL query divided by the overall cost.

We label with **(T)** those measures that are also valid for triple stores, and with $(\mathbf{T}^*)$ those that are valid only if the triple store is based on query rewriting (e.g., Stardog). Notice that the two *Simplicity* measures, even when retrievable, are not always suitable for *comparing* different OBDA systems. For example, it might not be possible to compare the simplicity of queries in the various phases, e.g., when such queries are expressed in different languages.

With these measures in mind, the different components of the benchmark should be designed so as to reveal strengths and weaknesses of a system in each phase. The conclusions drawn from the benchmark are more significant if the benchmark resembles a typical real-world scenario in terms of the complexity of the ontology and queries and size of the data set. Therefore, we consider the benchmark requirements in Table 2.

The current benchmarks available for OBDA do not meet several of the requirements above. Next we list some of the best known benchmarks and their shortcomings when it comes to evaluating OBDA systems. We show general statistics in Table 3.

Adolena: Designed in order to extend the South African National Accessibility Portal [20] with OBDA capabilities. It provides a rich class hierarchy, but a quite poor structure for properties. This means that queries over this ontology will usually be devoid of tree-witnesses. No data-generator is included, nor mappings.
Requirements Missing: O1, **Q2**, **D2**, **S1**

LUBM: The Lehigh University Benchmark (LUBM) [18] consists of a university domain ontology, data, and queries. For data generation, the UBA (Univ-Bench Artificial) data generator is available. However, the ontology is rather small, and the benchmark is not tailored towards OBDA, since no mappings to a (relational) data source are provided.
Requirements Missing: O1, **Q2**, **M1**, **M2**, **D1**

DBpedia: The DBpedia benchmark consists of a relatively large—yet, simple[4]—ontology, a set of user queries chosen among the most popular queries posed against the DBpedia [4] SPARQL endpoint, and a synthetic RDF data generator able to generate data having similar properties to the real-world data. This benchmark is specifically tailored to triple stores, and as such it does not provide any OBDA specific components like R2RML mappings, or a data set in the form of a relational database.
Requirements Missing: O1, **O2**, **Q2**, **M1**, **M2**

BSBM: The Berlin SPARQL Benchmark [9] is built around an e-commerce use case. It has a data generator that allows one to configure the data size (in triples), but there is no ontology to measure reasoning tasks, and the queries are rather simple. Moreover, the data is fully artificial.
Requirements Missing: O1, **O2**, **Q2**, **M1**, **M2**, **D1**,

FishMark: FishMark [8] collects comprehensive information about finned fish species. This benchmark is based on the FishBase real world dataset, and the queries are extracted from popular user SQL queries over FishBase; they are more complex than those from BSBM. However, the benchmark comes neither with mappings nor with a data generator. The data size is rather small (≈20M triples).
Requirements Missing: O1, **D2**, **S1**

A specific challenge comes from requirements **D1** and **D2**, i.e., given an initial real-world dataset, together with a rich ontology and mappings, expand the dataset in such a way that it populates the virtual instance in a sensible way (i.e., coherently with the ontology constraints and relevant statistical properties of the initial dataset). We address this problem in the next section.

[4] In particular, it is not suitable for reasoning w.r.t. existentials.

Table 2: Benchmark Requirements

O1 The **ontology** should include rich hierarchies of classes and properties.	**Q1** The **query set** should be based on actual user queries.	**M1** The **mappings** should be defined for elements of most hierarchies.
O2 The **ontology** should contain a rich set of axioms that infer new objects and could lead to inconsistency, in order to test the reasoner capabilities.	**Q2** The **query set** should be complex enough to challenge the query rewriter.	**M2** The **mappings** should contain redundancies, and suboptimal SQL queries to test optimizations.
D1 The **virtual instance** should be based on real world data.	**D2** The size of the **virtual instance** should be tunable.	**S1** The **languages** of the ontology, mapping, and query should be *standard*, i.e., based on R2RML, SPARQL, and OWL respectively.

Table 3: Popular Benchmark Ontologies: Statistics

	Ontology Stats. (Total)			Queries Stats. (Max)		
name	#classes	#obj/data_prop	#i-axioms	#joins	#opt	#tw
adolena	141	16	189	5	0	0
lubm	43	32	91	7	0	0
dbpedia	530	2148	3836	7	8	0
bsbm	8	40	0	14	4	0
fishmark	11	94	174	24	12	0

4. REQUIREMENTS FOR DATA GENERATION

In this section, we present the requirements for an OBDA data generator, under the assumption that we have an initial database that can be used as a seed to understand the distribution of the data that needs to be increased. To ease the presentation, we illustrate the main issues that arise in this context with an example.

EXAMPLE 4.1. *Consider a database $\mathcal{D}$ made of four tables, namely* TEmployee, TAssignment, TSellsProduct, *and* TProduct. *Table 4 shows a fragment of the content of the tables and their schemas, where* ***bold font*** *denotes primary keys and the foreign keys are in italics. We assume that every employee sells the majority of the products, hence the table* TSellsProduct *contains roughly the cross product of the tables* TEmployee *and* TProduct. *Next we present only a fragment of the data.*

Table 4: Database $\mathcal{D}$

TEmployee

id	name	branch
1	John	B1
2	Lisa	B1

TAssignment

branch	task
B1	task1
B1	task2
B2	task1
B2	task2

TSellsProduct

id	*product*
1	p1
2	p2
1	p2
2	p3

TProduct

product	size
p1	big
p2	big
p3	small
p4	big

Table 5 defines the set $\mathcal{M}$ of mapping assertions used to populate the ontology concepts :Employee, :Branch, *and* :ProductSize, *plus the object properties* :SellsProduct *and* :AssignedTo.

The virtual instance corresponding to the database $\mathcal{D}$ and mappings $\mathcal{M}$ includes the following RDF triples:

```
:1  rdf:type   :Employee.
:2  rdf:type   :Employee.
:1  :SellsProduct   :p1.
:1  :SellsProduct   :p2.
:2  :AssignedTo     :t1.
```

Suppose now we want to increase the virtual *RDF graph by a* growth-factor *of* 2. *Observe that this is not as simple as doubling the number of triples in every concept and property, or the number of tuples in every database relation. Let us first analyze the behavior of some of the ontology elements w.r.t. this aspect, and then how the mappings to the database come into play.*

- :ProductSize*: This concept will contain two individuals, namely* :small *and* :big*, independently of the growth-factor. Therefore, the virtual instances of the concept should not be increased when the RDF graph is extended.*
- :Employee *and* :Branch*: Since these classes do not depend on other properties, and since they are not intrinsically constant, we expect their size to grow linearly with the growth-factor.*
- :AssignedTo*: Since this property represents a cartesian product, we expect its size to grow roughly quadratically with the growth-factor.*
- :SellsProduct*: The size of this property grows with the product of the numbers of* :Employee*s and* :Product*s. Therefore, when we double these numbers, the size of* :SellsProduct *will roughly quadruplicate.*

In fact, the above considerations show that we do not have one *uniform growth-factor for the ontology elements. Our choice is to*

Table 5: Mappings $\mathcal{M}$

$\mathcal{M}_1$	`:{id} rdf:type :Employee`	$\leftarrow$	`SELECT id from TEmployee`
$\mathcal{M}_2$	`:{branch} rdf:type :Branch`	$\leftarrow$	`SELECT branch FROM TAssignments`
$\mathcal{M}_3$	`:{branch} rdf:type :Branch`	$\leftarrow$	`SELECT branch FROM TEmployee`
$\mathcal{M}_4$	`:{id} :SellsProduct :{product}`	$\leftarrow$	`SELECT id, product FROM TSellsProduct`
$\mathcal{M}_5$	`:{size} rdf:type :ProductSize`	$\leftarrow$	`SELECT size FROM TProduct`
$\mathcal{M}_6$	`:{id} :AssignedTo :{task}`	$\leftarrow$	`SELECT id, task FROM TEmployee NATURAL JOIN TAssignments`

characterize the growth in terms of the increase in size of those concepts in the ontology that are not intrinsically constant (e.g., `:ProductSize`*), and that do not “depend” on any other concept, considering the semantics of the domain of interest (e.g.,* `:Employee`*). We take this as measure for the growth-factor.*

The problem of understanding how to generate from a given RDF graph new additional triples coherently with the domain semantics is addressed in [36, 26]. The algorithm in [36] starts from an initial RDF graph and produces a new RDF graph, considering key features of the original graph (e.g., the distribution of connections among individuals). However, this approach, and all approaches producing RDF graphs in general, cannot be directly applied to the context of OBDA, where the RDF graph is virtual and generated from a relational database. Trying to apply these approaches indirectly, by first producing a “realistic” virtual RDF graph and then trying to reflect the virtual data into the physical (relational) data-source, is far from trivial due to the correlations in the underlying data. This problem, indeed, is closely related to the view update problem *[13], where each class (resp., role or data property) can be seen as a* view *on the underlying physical data. The view update problem is known to be challenging and actually decidable only for a very restricted class of queries used in the mappings [16]. Note, however, that our setting does not necessarily require to fully solve the view update problem, since we are interested in obtaining a physical instance that gives rise to a virtual instance with certain statistics, but not necessarily to a specific given virtual instance. The problem we are facing nevertheless remains challenging, and requires further research. We illustrate the difficulties that one encounters again on our example.*

- *The property* `:SellsProduct` *grows linearly w.r.t. the size of the table* `TSellsProduct`*, hence also this table has to grow quadratically with the growth-factor. Since* `TSellsProduct` *has foreign keys from the tables* `TEmployee` *and* `TProduct`*, to preserve the fact that every employee must be connected to every product, the two tables* `TEmployee` *and* `TProduct` *have both to grow linearly. It is worth noting that, to produce one* `:SellsProduct` *triple in the virtual instance, we have to insert three tuples in the database.*

- *Since the* `:Branch` *concept should grow linearly with the growth-factor, in order to preserve the duplicates ratio in the* `TAssignment.branch` *column then also the* `TAssignment` *table should grow linearly, and there should always be less branches than employees in* `TEmployee`.

- *Since* `:ProductSize` *does not grow, the attribute* `Size` *must contain only two values, despite the linear growth of* `TProduct`.

The previous example illustrated several challenges that need to be addressed by the generator regarding the *analysis* of the virtual and physical data, and the *insertion* of values in the database. Our goal is to generate a synthetic virtual graph where the cost of the queries is as similar as possible to the cost that the same query would have in a real-world virtual graph of comparable size. Observe that the same virtual graph can correspond to different database instances, that could behave very differently w.r.t. the cost of SQL query evaluation. Therefore, in order to keep the cost of the SPARQL query “realistic”, we need to keep the cost of the translated SQL “realistic” as well.

We are interested in data generators that perform an analysis phase on real-world data, and that use the statistical information learned in the analysis phase for their task. We present first in Table 6 the measures that are relevant in the analysis phase. We then derive the requirements for the data generator by organizing them in two categories: one for the analysis phase, and one for the generation phase.

Measures for the Analysis Phase.

The measures are summarized in Table 6. The table is divided in three parts:

The top part refers to measures relevant at virtual instance level, i.e., those capturing the shape of the virtual instance. *Virtual Multiplicity Distribution* describes the multiplicity of the properties, i.e., given a property p, and a number k, the VMD is the probability that a node n in the domain of p is connected to k elements through p. For instance, the VMD of `:AssignedTo` assigns probability 1 to the number 2. Observe that VMD is affected by the growth of the database (e.g., if the growth factor is 2, and the number of “tasks” grows linearly then the VMD of `:AssignedTo` assigns probability 1 to the number 4). *Virtual growth* is the expected growth for each ontology term w.r.t. the growth-factor. For instance, the virtual growth of `:AssignedTo` is quadratic.

The middle part refers to measures at the physical level that affect the VMD of the properties through the mappings. They are based on the sets of attributes of a table used in the mappings to define individuals and values in the ontology. We call such a set of attributes an *IGA* (individual-generating attributes). We say that two IGAs are *related* if and only if they occur in the same mapping defining the subject and the object of a property. Establishing the relevant statistics requires to identify pairs of IGAs through mapping analysis. *Intra-table Multiplicity Distribution* is defined for two related IGAs of the same table, both mapped to individuals/-values at the virtual level. It is defined for tuples over the IGAs in the same way as the VMD is defined for individuals. For instance, the Intra-MD for the IGAs {id} and {product} with respect to the property `:SellsProduct` assigns probability 1 to the number 2. *Inter-table Multiplicity Distribution* is defined for related IGAs belonging to two different tables. It is calculated like the intra-MD but over the joins specified in the mappings, e.g., the join of `TEmployee` and `TAssignment`.

The bottom part refers to measures at the physical level that do not affect VMD, but that influence growth at the virtual level and the overall performance of the system. Specifically, *IGA duplica-*

tion measures the ratio of identical copies of tuples over an IGA not occurring in a property definition, while (intra-table and inter-table) *IGA-pair duplication* is measured as the ratio of identical copies of a tuple over two related IGAs. For instance, the IGA duplication for the IGA `TAssignment.branch` is 1/2 (half of the tuples are duplicated).

Now we are ready to list the requirements for a data generator for OBDA systems.

Requirements for the Analysis Phase.

The generator should be able to analyze the physical instance and the mappings, in order to acquire statistics to assess the measures identified in Table 6.

Requirements for the Generation Phase.

We list now important requirements for the generation of physical data that gives rise through the mappings to the desired virtual data instance.

Tunable. The user must be able to specify a growth factor according to which the virtual instance should be populated.

Virtually Sound. The virtual instance corresponding to the generated physical data must meet the statistics discovered during the analysis phase and that are relevant at the virtual instance level.

Physically Sound. The generated physical instance must meet the statistics discovered during the analysis phase and that are relevant at the physical instance level.

Database Compliant. The generator must generate data that does not violate the constraints of the RDBMS engine—e.g., primary keys, foreign keys, constraints on datatypes, etc.

Fast. The generator must be able to produce a vast amount of data in a reasonable amount of time (e.g., 1 day for generating an amount of data sufficient to push the limits of the considered RDBMS system). This requirement is important because OBDA systems are expected to operate in the context of "Big-Data" [17].

5. NPD BENCHMARK

The Norwegian Petroleum Directorate[5] (NPD) is a governmental organisation whose main objective is to contribute to maximize the value that society can obtain from the oil and gas activities. The initial dataset that we use is the *NPD Fact Pages* [1], containing information regarding the petroleum activities on the Norwegian Continental Shelf (NCS).

The NPD benchmark consists of an an initial dataset reflecting the content of the FactPages, an ontology, a query set, a set of mappings, a data generator able to meaningfully increase the size of the initial dataset, and an automated testing platform.[6] The ontology, the query set, and the mappings to the dataset have all been developed at the University of Oslo [32], and are freely available online [2]. We adapted each of these, fixing some minor inconsistencies, adding missing mappings, and slightly modifying the query set to make the queries more suitable for an OBDA benchmark. Next we provide more details on each of these items.

Table 7: Statistics for the queries considered in the benchmark

query	#join	#tw	max(#subcls)	# opts	Agg	Filt.	Mod.
Q1	4	0	0	0	N	Y	N
Q2	5	0	0	0	N	Y	N
Q3	3	0	0	0	N	Y	Y
Q4	5	0	0	0	N	Y	Y
Q5	5	0	0	0	N	Y	Y
Q6	6	2	23	0	N	Y	Y
Q7	7	0	0	0	N	Y	N
Q8	3	0	0	0	N	Y	N
Q9	3	0	38	0	N	Y	Y
Q10	2	0	0	0	N	Y	Y
Q11	7	2	23	0	N	Y	Y
Q12	8	4	23	0	N	Y	Y
Q13	2	0	0	2	N	Y	N
Q14	2	0	0	2	N	Y	N
Q15	4	-	0	0	Y	Y	N
Q16	3	-	0	0	Y	Y	N
Q17	8	-	0	0	Y	N	Y
Q18	4	-	0	0	Y	N	N
Q19	8	-	0	0	Y	N	N
Q20	3	-	0	0	Y	N	N
Q21	3	-	0	0	Y	N	N

The Dataset.

The data from FactPages has been translated from CSV files into a structured database [32]. The obtained schema consists of 70 tables with 276 distinct columns ($\sim$ 1000 columns in total), and 94 foreign keys. The schemas of the tables overlap in the sense that several attributes are replicated in several tables. In fact, there are tables with more than 100 columns. The total size of the initial dataset is $\sim$ 50Mb.

The Ontology.

The ontology contains OWL axioms specifying comprehensive information about the underlying concepts in the dataset; in particular, the NPD ontology presents rich hierarchies of classes and properties, axioms that infer new objects, and disjointness assertions. We took the OWL 2 QL fragment of this ontology, and we obtained 343 classes, 142 object properties, 238 data properties, 1451 axioms, and maximum hierarchy depth of 10. Since we are interested in benchmarking OBDA systems that are able to rewrite queries over the ontology into SQL-queries that can be evaluated by a relational DBMS, we concentrate here on the OWL 2 QL profile[7] of OWL, which guarantees rewritability of unions of conjunctive queries (see, e.g., [10]). This ontology is suitable for benchmarking reasoning tasks, given that *(i)* it is a representative [24] and complex real-world ontology in terms of number of classes and maximum depth of the class hierarchy (hence, it allows for reasoning w.r.t. class hierarchies); *(ii)* it is complex w.r.t. properties, therefore it allows for reasoning w.r.t. existentials.
From the previous facts, it follows that the ontology satisfies requirements **O1**, **O2**, **S1**.

The Query Set.

The original NPD SPARQL query set contains 20 queries obtained by interviewing users of the NPD dataset. Starting from the original NPD query set, we devised 21 queries having different degrees of complexity (see Table 7). We also fixed some minor issues in the queries/ontology, e.g., the absence in the ontology of certain concepts present in the queries, fixing type inconsistencies, and flattening of nested sub-queries. In particular, observe that most complex queries involve both classes with a rich hierarchy and tree

[5] `http://www.npd.no/en/`

[6] `https://github.com/ontop/npd-benchmark/`

[7] `http://www.w3.org/TR/owl2-profiles/`

Table 6: Relevant measures at the virtual and physical instance level

Measures affecting the virtual instance level	
Virtual Multiplicity Distribution (VMD)	**Virtual Growth (VG)**
Multiplicity distribution of the properties in the virtual graph.	Function describing how fast concepts (resp., role/data properties) grow w.r.t. the growth-factor.
Measures affecting virtual multiplicity distribution	
Intra-table IGA Multiplicity Distr. (Intra-MD)	**Inter-table IGA Multiplicity Distr. (Inter-MD)**
Multiplicity distribution between IGAs belonging to the same table and generating objects connected through a virtual property.	Multiplicity distribution between IGAs belonging to different tables and connected through a virtual property.
Measures affecting RDBMS performance and virtual growth	
IGA Duplication (D)	
Repeated IGAs	
Intra-table IGA-pair Duplication (Intra-D)	**Inter-table IGA-pair Duplication (Inter-D)**
Repeated pairs of intra-table correlated IGAs.	Repeated pairs of inter-table correlated IGAs.

witnesses, which means that they are particularly suitable for testing the reasoner capabilities. Aggregates are also a source of complexity in the context of OBDA, since they increase the complexity of the semantic query optimisation tasks. These aggregate queries were not part of the first draft of this benchmark [12, 23], and they either add aggregates to queries without them—for instance, q15 is obtained from q1—or they are a fragment of aggregate queries in the original NPD query set—for instance, q17 and q19. Next, we provide some example queries from the benchmark.

The following query (q16) is a simple aggregate query that asks for the number of production licenses granted after year 2000.

```
SELECT (COUNT(?licence ) AS ?licnumber)
WHERE { [ ] a npdv:ProductionLicence ;
        npdv:name ?licence ;
        npdv:dateLicenceGranted ?dateGranted ;
        FILTER(?dateGranted > 2000) }
```

The following query (q6) is a query with tree-witnesses that asks for the wellbores, their length, and the companies that completed the drilling of the wellbore after 2008, and sampled more than 50m of cores.

```
SELECT DISTINCT ?wellbore (?length AS ?lenghtM)
                ?company ?year
WHERE {
  ?wc npdv:coreForWellbore
  [ rdf:type npdv:Wellbore ;
    npdv:name ?wellbore ;
    npdv:wellboreCompletionYear   ?year ;
    npdv:drillingOperatorCompany
    [npdv:name ?company ]] .
  { ?wc npdv:coresTotalLength ?length }

  FILTER(?year >= "2008"^^xsd:integer &&
         ?length > 50
  )
```

When existential reasoning is enabled, query q6 produces 2 tree-witnesses, and gets rewritten into a union of 73 intermediate queries (query rewriting phase). The tree-witnesses arise due to existential axioms containing `npdv:Wellbore` and `npdv:coreForWellbore`.

From the previous facts, it follows that the queries satisfy requirements **Q1**, **Q2**, **S1**.

The Mappings.

The R2RML mapping consists of 1190 assertions mapping a total of 464 among classes, objects properties, and data properties. The SQL queries in the mappings count an average of 2.6 unions of select-project-join queries (SPJ), with 1.7 joins per SPJ. We observe that the mappings have not been optimized to take full advantage of an OBDA framework, e.g., by trying to minimize the number of mappings that refer to the same ontology class or property, so as to reduce the size of the SQL query generated by unfolding the mapping. This gives the opportunity to the OBDA system to apply different optimization on the mappings at loading time.
From the previous facts, it follows that the mappings satisfies requirements **M1**, **M2**, **S1**.

Automatized testing platform.

The benchmark comes with a testing platform (called obda mixer[8]) that allows one to automatize the runs of the tests and the results collection. Mixer comes in form of an easily extensible java project, which can be extended to work with other obda systems as long as they provide a Java API and public interfaces able to return interesting statistics (e.g., unfolding or rewriting times).

5.1 VIG: The Data Generator

Next we present the **V**irtual **I**nstances **G**enerator (VIG) that we implemented in the NPD Benchmark. VIG produces a virtual instance by inserting data into the original database. The generator is general in the sense that, although it currently works with the NPD database, it can produce data also starting from instances different than NPD. The algorithm can be divided into two main phases, namely (i) an *analysis phase*, where statistics for relevant measures on the real-world data are identified; (ii) a *generation phase*, where data is produced according to the statistics identified in the analysis phase.

VIG starts from a non-empty database D. Given a growth factor g, VIG generates a new database D' such that $|T'| \approx |T| \cdot (1 + g)$, for each table T of D (where $|T|$ denotes the number of tuples of T). The size is approximated since, due to foreign key constraints, some tables might require the addition of extra tuples. In other words, the current implementation of the data generator assumes that the size of each table T grows accordingly to the growth factor. This rules out for example the case when the size of a table T depends on the cartesian product of two foreign keys (like in Example 4.1), since in this case the size of T depends quadratically on the sizes of the referred tables. In the case of NPD, however, there are no such tables and therefore the growth for each table is at most linear. Observe that the chosen generation strategy does not imply that every concept or property at the virtual level grows as the growth factor, since the growth depends not only on the content of the tables but also on the shape of the SQL queries defined in

[8] https://github.com/ontop/obda-mixer

the mappings (Example 4.1).

We now describe how VIG approximates the measures described in Table 6.

Measures (D), (Intra-D).

We compute (an approximation for) these measures by *Duplicate Values Discovery.* For each column $T.C$ of a table $T \in D$, VIG discovers the *duplicate ratio* for values contained in that column. The *duplicate ratio* is the ratio $(\|T.C\| - |T.C|)/\|T.C\|$, where $\|T.C\|$ denotes the number of values in the column $T.C$, and $|T.C|$ denotes the number of distinct values in $T.C$. A duplicate ratio "close to 1" indicates that the content of the column is essentially *independent* from the size of the database, and it should not be increased by the data generator.

Measures (Intra-MD), (Inter-MD), (Inter-D).

Instead of computing (an approximation for) these measures, VIG identifies the domain of each attribute. That is, for each column $T.C$ in a table T, VIG analyzes the content of $T.C$ in order to decide the range of values from which *fresh* non-duplicate values can be chosen. More specifically, if the domain of $T.C$ is `String` or simply unordered (e.g., polygons), then a random fresh value is generated. Instead, if the domain is a total order, then fresh values can be chosen from the non-duplicate values in the interval $[\min(T.C), \max(T.C)]$ or in the range of values adjacent to it. Observe that this helps in maintaining the domain of a column similar to the original one, and this in turn helps in maintaining intra- and inter-table multiplicity distribution. VIG also preserves standard database constraints, like primary keys, foreign keys, and datatypes, that during the generation phase will help in preserving the IGA multiplicity distribution. For instance, VIG analyses the loops in foreign key dependencies in the database. Let $T_1 \to T_2$ denote the presence of a foreign key from table T_1 to table T_2. In case of a cycle $T_1 \to T_2 \to \cdots \to T_k \to T_1$, inserting a tuple in T_1 could potentially trigger an infinite number of insertions. VIG performs an analysis on the values contained in the columns involved by the dependencies and discovers the maximum number of insertions that can be performed in the generation phase.
Next we describe the generation phase, and how it meets some of the requirements given in Section 6.

Duplicate Values Generation.

VIG inserts duplicates in each column according to the duplicate ratio discovered in the analysis phase. Each duplicate is chosen with a uniform probability distribution. This ensures, for those concepts that are not dependent from other concepts and whose individual are "constructed" from a single database column, a growth that is equal to the growth factor. In addition, it prevents intrinsically constant concepts from being increased (by never picking a fresh value in those columns where the duplicates ratio is close to 1). Finally, it helps keeping the sizes for join result sets "realistic" [34]. This is true in particular for the NPD database, where almost every join is realized by a single equality on two columns.
Requirement: Physically/Virtually Sound.

Fresh Values Generation.

For each column, VIG picks fresh non-duplicate values from the interval discovered during the analysis phase. If the number of values to insert exceeds the number of different fresh values that can be chosen from the interval $\mathcal{I}$, then values outside the interval are allowed. The choices for the generation of new value guarantees that columns always contain values "close" to the ones already present in the column. This ensures that the number of individual for concepts based on comparisons grows accordingly to the growth factor.
Requirement: Physically/Virtually Sound.

Metadata Constraints.

VIG generates values that do not violate the constraints of the underlying database, like primary keys, foreign keys, or type constraints. The NPD database makes use of geometric datatypes available in MYSQL. Some of them come with constraints, e.g., a polygon is a closed non-intersecting line composed of a finite number of straight lines. For each geometric column in the database, VIG first identifies the minimal rectangular region of space enclosing all the values in the column, and then it generates values in that region. This ensures that artificially generated geometric values will fall in the result sets of selection queries.
Requirement: Database Compliant/Virtually Sound.

Length of Chase Cycles.

In case a cycle of foreign key dependencies was identified during the analysis phase, then VIG stops the chain of insertions according to the boundaries identified in the analysis phase, while ensuring that no foreign key constraint is violated. This is done by inserting either a duplicate or a null in those columns that have a foreign key dependency.
Requirement: Database Compliant.

Furthermore, VIG allows the user to tune the growth factor, and the generation process is considerably fast, for instance, it takes ≈10hrs to generate 130 Gb of data.

5.2 Validation of the Data Generator

In this section we perform a qualitative analysis of the virtual instances obtained using VIG. We focus our analysis on those concepts and properties that either are supposed to grow linearly w.r.t. the growth factor or are supposed not to grow at all. These are 138 concepts, 28 object properties, and 226 data properties.

We report in Table 8 the growth of the ontology elements w.r.t. the growth of databases produced by VIG and by a purely random generator. The first column indicates the type of ontology elements being analyzed, and the growth factor g (e.g., "class_npd2" refers to the population of classes for the database incremented with a growth factor $g = 2$). The columns under "avg dev" show the average deviation of the actual growth from the expected growth, in terms of percentage of the expected growth. The remaining columns report the number and percentage of concepts (resp., object/data properties) for which the deviation was greater than 50%.

Concerning concepts, VIG behaves close to optimally. For properties, the difference between the expected virtual growth and the actual virtual growth is more evident. Nevertheless, VIG performs significantly better than a purely random approach (one order of magnitude for object properties, 2-3 times for data properties). We shall see how this difference strongly affects the results of the benchmark (Section 6).

5.3 Related Work on Data Generation

There are several data generators that come with database and semantic web benchmarks [9, 4, 18, 3]. As explained before, it is not trivial to re-use a semantic web triple generator (e.g., [36]) since in OBDA this implies solving the view update problem. Therefore, we focus on DB data generators. To the best of our knowledge, most of them (such as the ones for TPC or Wisconsin) are tailored to a given DB schema, and moreover such schemas are rather simple (often around 20 tables). A notable example is the TPC-DS data generator. TPC-DS is the latest TPC benchmark with

Table 8: Comparison between VIG and a random data generator

type_db	avg dev heuristic	 random	err $\geq 50\%$ (absolute) heuristic	 random	err $\geq 50\%$ (relative) heuristic	 random
class_npd2	3.24%	370.08%	2	67	1.45%	48.55%
class_npd10	6.19%	505.02%	3	67	2.17%	48.55%
obj_npd2	87.48%	648.22%	8	12	28.57%	42.86%
obj_npd10	90.19%	883.92%	8	12	28.57%	42.86%
data_npd2	39.38%	96.30%	20	46	8.85%	20.35%
data_npd10	53.49%	131.17%	28	50	12.39%	22.12%

Table 9: Tractable queries (MySQL)

db	avg(ex_time) msec.	avg(out_time) msec.	avg(res_size) msec.	qmpH	#(triples)
NPD	44	102	15960	2167.37	≈2M
NPD2	70	182	30701	1528.01	≈6M
NPD10	148	463	81770	803.86	≈25M
NPD50	338	1001	186047	346.87	≈116M
NPD100	547	1361	249902	217.36	≈220M
NPD500	2415	5746	943676	57.80	≈1.4B
NPD1500	6740	18582	2575679	17.66	≈4B

Table 10: Tractable Queries (PostgreSQL)

db	avg(ex_time) msec.	avg(out_time) msec.	avg(res_size) msec.	qmpH	#(triples)
NPD	61	36	$2.3 * 10^4$	5278	≈2M
NPD2	121	71	$4.2 * 10^4$	2684	≈6M
NPD5	173	99	$7.1 * 10^4$	1893	≈12M
NPD10	222	138	$1.1 * 10^5$	1429	≈25M
NPD50	592	355	$2.7 * 10^5$	542	≈116M
NPD100	1066	516	$4.1 * 10^5$	325	≈220M
NPD500	$4.1 * 10^4$	467	$3.3 * 10^5$	12	≈1.3B
NPD1500	$2.6 * 10^5$	3470	$1.15 * 10^6$	1.9	≈4B

Table 11: Hard Queries Rewriting And Unfolding

query	#rw	#un	rw time sec.	un time sec.
	Ext. Reasoning OFF			
q6	1	48	0	0.1
q9	1	570	0	0.1
q10	1	24	0	0.9
q11	1	24	0	0.1
q12	1	48	0	0.2
q13	1	4	0	0.005
q14	1	2	0	0.01
q15	1	4	0	0.03
q16	1	26	0	0.05
q17	1	40	0	0.1
q18	1	38	0	0.2
q19	1	40	0	0.1
q20	1	13	0	0.04
q21	1	13	0	0.06
	Ext. Reasoning ON			
q6	73	1740	1.8	1.3
q9	1	150	0	0.03
q10	1	24	0	0.01
q11	73	870	0.03	0.7
q12	10658	5220	525	139

a scaling, correlation, and skew methodology in the data generator (MUDD) [33]. MUDD takes the distribution of each pair column/table as a manually predefined input, and generates data according to the defined distribution. On the other hand, VIG, collects different statistics about each table, and generates data to keep the statistics constant. MUDD allows a more sophisticate and precise data generation, but it requires a deep understanding of the dataset, and manual settings that can be challenging as the complexity of the schema increases (the TPC-DS schema contains 20 tables, whereas the NPD schema contains 70 tables). We plan to extend our work with distribution analysis so as to replicate the skew that is usually present in real-world data. However, observe that in general skew might not be a crucial factor in determining the shape of virtual instances since repeated triples are removed in the virtual RDF graphs.

6. BENCHMARK RESULTS

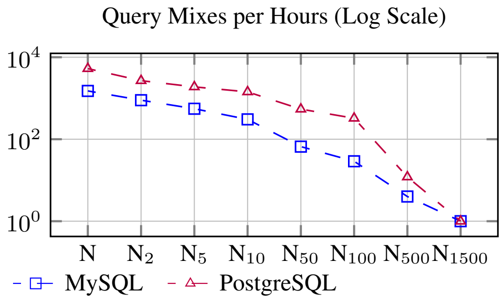

Figure 1: Full summary of Ontop-MySQL vs Ontop-PostgreSQL

We ran the benchmark on the *Ontop* system[9] [29, 22], which, to the best of our knowledge, is the only *fully* implemented OBDA system that is freely available. In addition, we tried a closed OBDA system, *Mastro* [11], that is used in large industrial projects, and two systems that use mappings but that do not provide reasoning, namely OpenLink Virtuoso Views [5] and Morph [6]. Unfortunately, Mastro and Virtuoso do not fully support R2RML mappings and Morph is not able to load the mappings for NPD. Ultrawrap [31] is another commercial OBDA system but we were not granted the right to test it.

In order to provide a meaningful comparison, we looked for a triple store that allows for OWL 2 QL reasoning through query rewriting—Virtuoso does not provide this feature. Thus, we compared *Ontop* with *Stardog 2.1.3*. Stardog[10] is a commercial RDF database developed by Clark&Parsia that supports SPARQL 1.1 queries and OWL 2 for reasoning.

Since Stardog is a triple store, we needed to materialize the virtual RDF graph exposed by the mappings and the database using *Ontop*. For the aggregate queries we used an experimental unreleased version of *Ontop* (V2.0) that does not support existential reasoning in conjunction with aggregates.

MySQL and PostgreSQL were used as underlying relational database systems. The hardware consisted of an HP Proliant server with 24 Intel Xeon X5690 CPUs (144 cores @3.47GHz), 106GB of RAM and a 1TB 15K RPM HD. The OS is Ubuntu 12.04 LTS. Due to space constraints, we present the results for only one running client. We obtained results with the *existential reasoning* on (for non-aggregate queries) and off.

In order to test the scalability of the systems w.r.t. the growth of

[9] `http://ontop.inf.unibz.it/`

[10] `http://stardog.com/`

Table 12: Hard queries-*Ontop*/MySQL

query	NPD	NPD2	NPD5	NPD10	NPD10 RAND
	rp_t/weight R+U	rp_t/weight R+U	rp_t/weight R+U	rp_t/weight R+U	rp_t/weight R+U
	(sec./ratio)	(sec./ratio)	(sec./ratio)	(sec./ratio)	(sec./ratio)
No Existentials Reasoning					
q6	1.5/0.07	8.2/0.02	23/<0.01	51/<0.01	54/<0.01
q9	0.6/0.17	2.3/0.03	4/0.03	50/<0.01	51/<0.01
q10	0.07/0.14	0.1/0.1	0.16/0.06	0.2/0.05	0.3/0.03
q11	0.9/0.1	36/<0.01	198/<0.01	1670/<0.01	70/<0.01
q12	0.8/0.16	41/<0.01	275/<0.01	1998/<0.01	598/<0.01
q13	0.1/0.01	0.2/<0.01	0.2/0.01	0.4/<0.01	1.1/<0.01
q14	0.3/<0.01	1.0/<0.01	1.4/<0.01	0.6/<0.01	5.3/<0.01
q15	1015.7/<0.01	—	—	—	—
q16	1.4/<0.01	13.2/<0.01	32.7/<0.01	171.9/<0.01	406.1/<0.01
q17	—	—	—	—	—
q18	—	—	—	—	—
q19	—	—	—	—	—
q20	211.1/<0.01	1913.4/<0.01	—	—	—
q21	210.9/<0.01	1905.6/<0.01	—	—	—
Existentials Reasoning					
q6	8.5/0.35	18/0.19	36/0.09	85/0.04	88/0.03
q9	0.2/0.2	0.2/0.2	0.2/0.2	0.2/0.2	0.2/0.2
q10	0.1/0.2	0.1/0.2	0.3/0.07	0.7/0.03	1.8/0.01
q11	3/0.2	25/0.03	980/<0.01	980/<0.01	41/0.02
q12	686/0.97	733/0.91	868/0.74	2650/0.24	880/0.74

the database, we used the data generator described in Section 5.1 and produced several databases, the largest being approximately 1500 times bigger than the original one ("NPD1500" in Table 9, ≈ 117 GB of size on disk).

Tables 9, 10, and Figure 1 show 7 queries from the initial query set, for which the unfolding produces a single select-project-join (SPJ) SQL query after being *optimised* by *Ontop*. Such optimisations remove redundant selfjoins, redundant unions, push joins into unions, etc. See [30] for a complete description. These results show the scalability of this approach. The query mix of 7 queries was executed 10 times (in each dataset, NPD1–NPD1500), each time with different filter conditions so that the effect of caching is minimized, and statistics were collected in each execution. We measure the sum of the *query execution time* (avg(ex_time)), the time spent by the system to display the results to the user (avg(out_time)), the number of results (avg(res_size)), and the query mixes per hour (qmpH), that is, the number of times that these 7 queries can be answered in one hour. In this experiments we can see that *Ontop*-PostgreSQL runs orders of magnitude faster than *Ontop*-MySQL whenever the query does not contain Optionals. However, for queries that contain optionals, MySQL performs much better. By looking at the query plans[11] in both DB engines, we found out that MySQL can better optimise the query by eliminating left joins over the same table.

For instance, the SQL translation of query 14 requires 2 left joins over the same table. PostgreSQL materialises the subqueries and then performs both left joins. MySQL, on the other hand, can avoid such redundant left joins over the same table.

Table 11 contains results showing the number of unions of SPJ queries generated after rewriting (#rw) and after unfolding (#un) for the 5 hardest queries. In addition, it shows the time spent by *Ontop* on rewriting and unfolding. Here we can observe how existential reasoning can produce a noticeable performance overhead, by producing queries consisting of unions of more than 5000 subqueries (c.f., q12). This blow-up is due to the combination of rich hierarchies, existentials, and mappings. These queries are meant to be used in future research on query optimization in OBDA.

Tables 12 and 13 contains results for the 13 hardest queries in *Ontop*. Some of these queries take hours to be executed, therefore qmpH is not so informative in this case. Thus, we run each query twice with a timeout of 2 hours on the response time. The dashes in the tables represent timeouts. Observe that the response time tends to grow faster than the growth of the underlying database. This follows from the complexity of the queries produced by the unfolding step, which usually contain several joins (remember that the worst case cardinality of a result set produced by a join is quadratic in the size of the original tables). Column NPD10 RAND witnesses how using a purely random data generator gives rise to datasets for which the queries are much simpler to evaluate. This is mainly due to the fact that a random generation of values tends to decrease the ratio of duplicates inside columns, resulting in smaller join results over the tables [34]. Hence, purely randomly generated datasets are not appropriate for benchmarking.

In Figure 2, we compare the response times in *Ontop* and Stardog. As expected, the queries with worst performance in OBDA (q_6, q_9, q_{10},… etc.) are those that were affected by the blow-up shown in Table 11. In this case, Stardog performs orders of magnitude faster than *Ontop*. These queries should guide the future research in query optimisation in OBDA. On the other hand, the queries that perform well (q_1, q_2, q_3,… etc.) are those where the different optimizations lead to a simple SPJ SQL query. Note that the times required to materialize (by *Ontop*) and load the dataset in Stardog go from 1 min. (NPD1) to 1 hour (NPD10).

7. CONCLUSIONS AND FUTURE WORK

The benchmark proposed in this work is the first one that thoroughly analyzes a complete OBDA system implementation in all significant components, including query rewriting, query unfolding, and query execution. So far, little or no work has been done in this direction, as pointed out in [25]. This benchmark reveals the strengths and pitfalls of OBDA. We confirmed that this approach can be orders of magnitude faster than standard triple stores, fully exploiting the highly optimized DB engines. To achieve such performance, structural and semantic optimizations of the SQL trans-

[11] Available in `http://www.inf.unibz.it/~dlanti/techreportNPD-EDBT.pdf`

Table 13: Hard queries-*Ontop*/PostgreSQL

query	NPD	NPD2	NPD5	NPD10	NPD10 RAND
	rp_t/weight R+U	rp_t/weight R+U	rp_t/weight R+U	rp_t/weight R+U	rp_t/weight R+U
	(sec./ratio)	(sec./ratio)	(sec./ratio)	(sec./ratio)	(sec./ratio)
No Existentials Reasoning					
q06	1.2/0.15	1.3/0.11	3.9/0.04	4.3/0.03	3.6/0.03
q09	6.4/0.28	6.0/0.21	7.5/0.32	8.6/0.19	1.5/0.1
q10	0.2/0.16	0.6/0.03	0.7/0.03	0.9/0.03	1.4/0.01
q11	1.1/0.12	22.1/<0.01	66.1/<0.01	160.3/<0.01	110.9/<0.01
q12	1.9/0.16	20.9/0.01	101.6/<0.01	195.6/<0.01	121.5/<0.01
q13	0.9/0.06	0.2/0.02	0.5/<0.01	0.4/<0.01	0.7/<0.01
q14	453.2/<0.01	—	—	—	—
q15	122.9/<0.01	366.3/<0.01	771.3/<0.01	1491.2/<0.01	1296.9/<0.01
q16	1.5/0.01	17.0/<0.01	64.6/<0.01	237.8/<0.01	588.6/<0.01
q17	—	—	—	—	—
q18	—	—	—	—	—
q19	—	—	—	—	—
q20	1.8/<0.01	5.2/<0.01	10.5/<0.01	20.0/<0.01	19.2/<0.01
q21	1.8/<0.01	5.2/<0.01	10.5/<0.01	19.7/<0.01	12.2/<0.01
Existentials Reasoning					
q06	14.9/0.14	48.6/0.04	52.0/0.07	55.0/0.02	58.0/0.01
q09	0.4/0.16	0.3/0.15	0.7/0.1	1.0/0.1	0.5/0.1
q10	0.2/0.14	0.5/0.07	0.6/0.04	0.8/0.06	1.4/0.02
q11	17.5/0.08	117.3/<0.01	—	—	—
q12	1395.6/0.5	4090.8/0.47	—	—	—

lation are required. However, the results also show that the exponential blowup in the unfolding phase is a major source of performance loss of modern OBDA systems. If this issue is not handled properly, it can prevent OBDA systems from being deployed in production environments. This explosion, however, can be strongly reduced using tuning and optimisation techniques that exploit the information hidden in the data, such as functional dependencies, redundant mappings, etc. We are currently working on this topic.

For a better analysis it is crucial to refine the generator in such a way that domain-specific information is taken into account, and a better approximation of real-world data is produced.

Acknowledgement. This paper is supported by the EU under the large-scale integrating project (IP) Optique (Scalable End-user Access to Big Data), grant agreement n. FP7-318338.

8. REFERENCES

[1] http://factpages.npd.no/factpages/

[2] http://sws.ifi.uio.no/project/npd-v2/

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

[4] http://dbpedia.org/sparql/

[5] http://virtuoso.openlinksw.com/

[6] https://github.com/oeg-upm/morph-rdb

[7] Abadi, D., et al.: The Beckman report on database research (2013), available at http://beckman.cs.wisc.edu/

[8] Bail, S., Alkiviadous, S., Parsia, B., Workman, D., van Harmelen, M., Goncalves, R.S., Garilao, C.: FishMark: A linked data application benchmark. In: Proc. of the Joint Workshop on Scalable and High-Performance Semantic Web Systems (SSWS+HPCSW), vol. 943, pp. 1–15. CEUR, ceur-ws.org (2012)

[9] Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. on Semantic Web and Information Systems 5(2), 1–24 (2009)

[10] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: RW Tutorial Lectures, LNCS, vol. 5689, pp. 255–356. Springer (2009)

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

[12] Calvanese, D., Lanti, D., Rezk, M., Slusnys, M., Xiao, G.: A scalable benchmark for OBDA systems: Preliminary report. In: Proc. of ORE. CEUR, ceur-ws.org (2014)

[13] Cosmadakis, S.S., Papadimitriou, C.H.: Updates of relational views. JACM 31(4), 742–760 (1984)

[14] Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, W3C (Sep 2012), available at http://www.w3.org/TR/r2rml/

[15] DeWitt, D.J.: The Wisconsin benchmark: Past, present, and future. In: The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, 2 edn. (1993)

[16] Franconi, E., Guagliardo, P.: The view update problem revisited. CoRR Technical Report arXiv:1211.3016, arXiv.org e-Print archive (2012), available at http://arxiv.org/abs/1211.3016

[17] Giese, M., Haase, P., Jiménez-Ruiz, E., Lanti, D., Özçep, Ö., Rezk, M., Rosati, R., Soylu, A., Vega-Gorgojo, G., Waaler, A., Xiao, G.: Optique – Zooming in on Big Data access. IEEE Computer (2015)

[18] Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. of Web Semantics 3(2–3), 158–182 (2005)

[19] Guo, Y., Qasem, A., Pan, Z., Heflin, J.: A requirements driven framework for benchmarking semantic web knowledge base systems. IEEE TKDE 19(2), 297–309 (2007)

[20] Keet, C.M., Alberts, R., Gerber, A., Chimamiwa, G.: Enhancing web portals with Ontology-Based Data Access: the case study of South Africa's Accessibility Portal for people with disabilities. In: Proc. of OWLED. CEUR, ceur-ws.org, vol. 432 (2008)

[21] Kontchakov, R., Lutz, C., Toman, D., Wolter, F.,

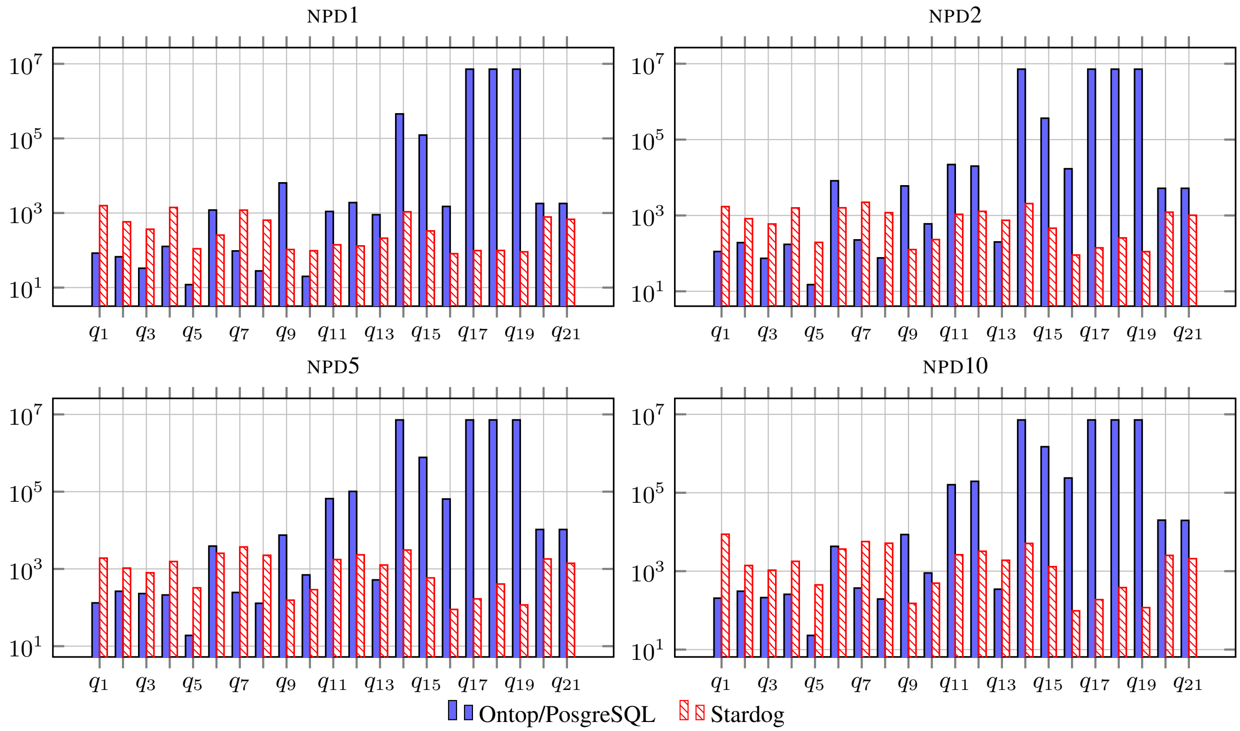

Figure 2: Query Answering over NPD1 to NPD10 (Times in ms, Log Scale)

Zakharyaschev, M.: The combined approach to query answering in *DL-Lite*. In: Proc. of KR. pp. 247–257 (2010)

[22] Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of ISWC. LNCS, vol. 8218, pp. 558–573. Springer (2013)

[23] Lanti, D., Rezk, M., Slusnys, M., Xiao, G., Calvanese, D.: The NPD benchmark for OBDA systems. In: Proc. of SSWS. CEUR, `ceur-ws.org` (2014)

[24] LePendu, P., Noy, N.F., Jonquet, C., Alexander, P.R., Shah, N.H., Musen, M.A.: Optimize first, buy later: Analyzing metrics to ramp-up very large knowledge bases. In: Proc. of ISWC. LNCS, vol. 6496, pp. 486–501. Springer (2010)

[25] Mora, J., Corcho, O.: Towards a systematic benchmarking of ontology-based query rewriting systems. In: Proc. of ISWC. LNCS, vol. 8218, pp. 369–384. Springer (2013)

[26] Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.C.: DBpedia SPARQL Benchmark – Performance assessment with real queries on real data. In: Proc. of ISWC, Volume 1. LNCS, vol. 7031, pp. 454–469. Springer (2011)

[27] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)

[28] Rodríguez-Muro, M., Calvanese, D.: Dependencies: Making ontology based data access work in practice. In: Proc. of AMW. CEUR, `ceur-ws.org`, vol. 749 (2011)

[29] Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Answering SPARQL queries over databases under OWL 2 QL entailment regime. In: Proc. of ISWC. LNCS, vol. 8218. Springer (2014)

[30] Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings (2014), submitted for publication

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

[32] Skjæveland, M.G., Lian, E.H., Horrocks, I.: Publishing the Norwegian Petroleum Directorate's FactPages as Semantic Web data. In: Proc. of ISWC. LNCS, vol. 8219, pp. 162–177. Springer (2013)

[33] Stephens, J.M., Poess, M.: MUDD: A multi-dimensional data generator. In: Proc. of the 4th Int. Workshop on Software and Performance. pp. 104–109. ACM (2004)

[34] Swami, A., Schiefer, K.B.: On the estimation of join result sizes. In: Proc. of EDBT. LNCS, vol. 779, pp. 287–300. Springer (1994)

[35] Venetis, T., Stoilos, G., Stamou, G.B.: Query extensions and incremental query rewriting for OWL 2 QL ontologies. J. on Data Semantics 3(1), 1–23 (2014)

[36] Wang, S.Y., Guo, Y., Qasem, A., Heflin, J.: Rapid benchmarking for semantic web knowledge base systems. In: Proc. of ISWC. LNCS, vol. 3729, pp. 758–772. Springer (2005)

[37] Weithöner, T., Liebig, T., Luther, M., Böhm, S.: What's wrong with OWL benchmarks. In: Proc. of the 2nd Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS). pp. 101–114 (2006)