The NPD Benchmark for OBDA Systems

Davide Lanti, Martin Rezk, Mindaugas Slusnys, Guohui Xiao, and Diego Calvanese

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

Abstract. In Ontology-Based Data Access (OBDA), queries are posed over a high-level conceptual view, and then translated into queries over a potentially very large (usually relational) data source. 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: performance. To properly evaluate OBDA systems, benchmarks tailored towards the requirements in this setting are needed. OWL benchmarks, which have been developed to test the performance of generic SPARQL query engines, however, fail to evaluate OBDA specific features. In this work, we propose a novel benchmark for OBDA systems based on the Norwegian Petroleum Directorate (NPD). Our benchmark comes with novel techniques to generate, from available 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 Ontology-Based Data Access (OBDA), queries are posed over a high-level conceptual view, and then translated into queries over a potentially very large (usually relational) data source. The conceptual layer is given in the form of an ontology that defines a shared vocabulary. 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 the data. The W3C standard R2RML [10], was created with the goal of providing a standardized language for the specification of mappings in the OBDA setting. The ontology and expose a virtual instance (RDF graph) that can be queried using the de facto query language in the Semantic Web community: SPARQL. To properly evaluate the performance of OBDA systems, benchmarks tailored towards the requirements in this setting are needed. OWL benchmarks, which have been developed to test the performance of generic SPARQL query engines, however, fail at 1) exhibiting a complex real-world ontology, 2) providing challenging real world queries, 3) providing large amounts of real world data, and the possibility to test a system over data of increasing size, and 4) capturing important OBDA-specific measures related to the rewriting-based query answering approach in OBDA.

In this work, we propose a novel benchmark for OBDA systems based on the Norwegian Petroleum Directorate (NPD), which is a real world use-case adopted in the EU project Optique[1]. In the benchmark, which is available online[2], we adopt the NPD

[1] `http://www.optique-project.eu/`

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

Fact Pages as dataset, the NPD Ontology, which has been mapped to the NPD Fact Pages stored in a relational database, and queries over such an ontology developed by domain experts. One of the main challenges we address here has been to develop a data generator for generating datasets of increasing size, starting from the available data. This problem has been studied before in the context of databases [2], where increasing the data size is achieved by encoding domain-specific information into the data generator [9, 4, 5]. One drawback of this approach is that each benchmark requires its ad-hoc generator, and also that it disregards OBDA specific aspects. In the context of triple stores, [25, 14] present an interesting approach based on machine learning. Unfortunately, the approach proposed in these papers is specifically tailored for triple stores, and thus it is not directly applicable to the OBDA settings.

In Section 2, we present the necessary requirements for an OBDA Benchmark. In Section 3, we discuss the requirements for an OBDA instance generator. In Section 4, we present the NPD benchmark 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, we perform a qualitative analysis of the virtual instances obtained using VIG. In Section 6, we carry out a validation of our benchmark, showing that it is more adequate than previous benchmarks not tailored for OBDA. We conclude in Section 7.

2 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 connecting the ontological terms to the data sources. 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 [18, 14, 25]. 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 step step 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* [20]).
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 [16]).
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.

Table 1: Measures for OBDA

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

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 for the original input query.

Note that a variation of the above workflow has actually been proposed in [18], but without identifying a distinct starting phase, and singling out a result translation phase from query execution. There are several approaches to deal with Phase 2 [16, 24]. 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 [16]. 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 [16], and can produce an exponential blow-up in the query size, some systems allow 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 into the form of a black-box with proprietary code with no APIs providing the necessary information, e.g., the access to the rewritten query; or because a system combines one or more phases, e.g., query rewriting and query translation. Based on the above phases, we identify the measures important for *evaluating* OBDA systems in Table 1. The meaning of the *Performance Measures* is clear from the name, but 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 (**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, 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 requirements in Table 2.

Table 2: Benchmark Requirements

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

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 evaluate OBDA systems. We show general statistics in Table 3.

Adolena: Designed in order to extend the South African National Accessibility Portal [15] 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) [13] consists of a university domain ontology, data, and queries. For data generation, the UBA (Univ-Bench

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

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[3]—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 [3] 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 [1] collects comprehensive information about finned fish species. This benchmark is based in 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.

3 Requirements for Generating Virtual Instances

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

[3] In particular, it is not suitable for reasoning w.r.t. existentials.
[4] `http://dbpedia.org/sparql`

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 1. Suppose we have the following database tables, where the primary keys are in **bold font** and the foreign keys are *emphasized*. 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.

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

Consider the following mappings populating the ontology concepts `Employee`, `Branch`, and `ProductSize`, and the object properties `SellsProduct` and `AssignedTo`.

```
M1 :{id} rdf:type :Employee       ← SELECT id from TEmployee
M2 :{branch} rdf:type :Branch     ← SELECT branch FROM TAssignments
M3 :{branch} rdf:type :Branch     ← SELECT branch FROM TEmployee
M4 :{id} :SellsProduct :{product} ← SELECT id, product FROM TSellsProduct
M5 :{size} rdf:type :ProductSize  ← SELECT size FROM TProduct
M6 :{id} :AssignedTo :{task}      ← SELECT id, task
                                    FROM TEmployee
                                    NATURAL JOIN
                                    TAssignments
```

The virtual instance corresponding to the above database and mappings includes the following RDF triples:

```
:1  rdf:type  :Employee.        :1  :SellsProduct  :p1.
:2  rdf:type  :Employee.        :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 an n-to-n relationship, we expect its size to grow roughly quadratically with the growth-factor.
- `SellsProduct`: The size of this property grows roughly with the product of the numbers of `Employee`s and `Product`s. Hence, 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 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 [25, 19]. The algorithm in [25] 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* [8], 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 [12]. 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 inter-table correlation (according to which roughly every employee is 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 also the `Branches` concept should grow linearly with the growth-factor, while preserving the intra- and inter-table correlations, 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 5 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. Table 5 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 correlation* measures the correlation between individuals connected through a property, i.e., the number of individuals/values to which every individual is connected via an object/data property. *Virtual growth* is the expected growth for each ontology term w.r.t. the growth-factor. Observe that these two measures are strongly related to each other. The middle part refers to measures at the physical level that strongly affect the shape of the virtual instance through the form of the mappings. They are based on the sets of attributes of a table used to define individuals and values in the ontology through the mapping. We call such a set of attributes an *IGA* (individual-generating attributes set). Establishing the relevant statistics requires to identify pairs of IGAs through mapping analysis. Specifically, *intra-table IGA correlation* is defined for two IGAs of the same table, both mapped to individuals/values at the virtual level. It is measured for tuples over the IGAs as the virtual correlation of the individuals that are generated via the mapping from the tuples. *Inter-table IGA correlation* is measured for IGAs belonging to two different tables, by taking the intra-table IGA correlation over the join of the two tables. The bottom part refers to measures at the physical level that do not affect correlation at the virtual instance level, but that influence growth at the virtual level and the overall performance of the system. Specifically, *IGA duplication* measures the number of identical copies of tuples over an IGA, while (intra-table and inter-table) *IGA-pair duplication* is measured as the number of identical copies of a tuple over two correlated IGAs. Notice that, for benchmarking purposes, both IGA correlation and IGA duplication are important.

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 5.

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.

Table 5: Relevant measures at the virtual and physical instance level

Measures affecting the virtual instance level	
Virtual Correlation (VC)	**Virtual Growth (VG)**
Correlations between the various elements in the virtual instance.	Function describing how fast concepts (resp., role/data properties) grow w.r.t. the growth-factor.
Measures affecting virtual correlation and virtual growth	
Intra-table IGA Correlation (Intra-C)	**Inter-table IGA Correlation (Inter-C)**
Correlation (obtained through repetition analysis) between IGAs belonging to the same table and generating objects connected through a mapped property.	Correlation (obtained through analysis of the repetitions of *tuples used to join IGAs* and of the joined IGAs) between IGAs belonging to different tables.
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.

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" [6].

4 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 are the *NPD Fact Pages*[6], containing information regarding the petroleum activities on the Norwegian continental shelf. The ontology, the query set, and the mappings to the dataset have all been developed at the University of Oslo [22], and are freely available online[7]. Next we provide more details on each of these items.

The Ontology. The ontology contains OWL axioms specifying comprehensive information about the underlying concepts in the dataset; specifically rich hierarchies of classes and properties, axioms that infer new objects, and disjointness assertions. We took the OWL 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

[5] `http://www.npd.no/en/`
[6] `http://factpages.npd.no/factpages/`
[7] `http://sws.ifi.uio.no/project/npd-v2/`

Table 6: Statistics for the queries considered in the benchmark

query	#operators	#join	#depth	#tw	max(#subclasses)	# opt
Q1	12	4	8	0	0	0
Q2	14	5	9	0	0	0
Q3	10	3	7	0	0	0
Q4	14	5	9	0	0	0
Q5	14	5	8	0	0	0
Q6	18	6	12	2	23	0
Q7	17	7	10	0	0	0
Q8	10	3	7	0	0	0
Q9	10	3	7	0	38	0
Q10	9	2	7	0	0	0
Q11	20	7	12	2	23	0
Q12	26	8	12	4	23	0
Q13	8	2	7	0	0	2
Q14	12	2	7	0	0	2

here on the OWL 2 QL profile[8] of OWL, which guarantees rewritability of unions of conjunctive queries (see, e.g., [7]). This ontology is suitable for benchmarking reasoning tasks, given that *(i)* it is a representative [17] 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 with respect to 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 25 queries obtained by interviewing users of the NPD dataset. Starting from the original NPD query set, we devised 14 queries having different degrees of complexity (see Table 6). In particular, observe that most complex queries involve both classes with a rich hierarchy and tree witnesses, which means that they are particularly suitable for testing the reasoner capabilities. We also fixed some minor issues, e.g., the absence in the ontology of certain concepts present in the queries, removing aggregates (to be tackled in future work), and flattening of nested sub-queries.
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**.

4.1 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

[8] `http://www.w3.org/TR/owl2-profiles/#OWL_2_QL`

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 $|T'| = |T| \cdot (1 + g)$, for each table T of D (where $|T|$ denotes the number of tuples of T). This first approach assumes that each table in the database should grow linearly with respect to the growth factor, which is not true in general, but it holds for NPD. In addition, VIG approximates the measures described Table 5 as shown below.

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 distinct of 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-C), (Inter-C), (Inter-D). Instead of computing (an approximation for) these measures, VIG identifies the domain of each attribute. That is, for each non-fixed domain 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 a string or unordered (e.g., polygons), then simply a random 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 correlations. VIG also preserves standard database constraints, like primary keys, foreign keys, and datatypes, that during the generation phase will help in preserving the IGA correlations. 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* 5.

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" [23]. 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 guarantee 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 $\approx$10hrs to generate 130 Gb of data.

5 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. These are 138 concepts, 28 object properties, and 226 data properties.

We report in Table 7 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 "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. However, it is orders of magnitude better than the random generator. We shall see how this difference strongly affects the results of the benchmark (*Section* 6).

Table 7: Comparison between VIG and a random data generator

type_db	avg dev heuristic	avg dev random	err ≥50% (absolute) heuristic	err ≥50% (absolute) random	err ≥50% (relative) heuristic	err ≥50% (relative) 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 8: Tractable queries (time in ms)

db	avg(ex_time)	avg(out_time)	avg(res_size)	qmpH	#(triples)
NPD	62	113	16766	1507.85	≈2M
NPD2	116	232	35991	896.55	≈6M
NPD5	246	410	70411	554.35	≈12M
NPD10	408	736	124253	305.46	≈25M
NPD50	2138	3208	539461	66.82	≈116M
NPD100	5292	6727	1160972	29.12	≈220M
NPD500	37382	48512	7511516	4.22	≈1.3B
NPD1500	132155	148495	23655243	1.27	≈4B

Virtual Correlation. From our experiments we witnessed that the virtual correlation is preserved for the 28 object properties that are generated from a single table. That is, the correlation remains constant and it grows only in the case of cartesian products on columns with high duplicate ratio and that together form a primary key. More results can be found in the benchmark page.

6 Benchmark Results

We ran the benchmark on the *Ontop* system[9] [21], which, to the best of our knowledge, is the only fully implemented OBDA system that is freely available. In addition, 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*.

MySQL was used as underlying relational database system. The hardware consisted of an HP Proliant server with 24 Intel Xeon X5690 CPUs (144 cores @3.47GHz), 160GB of RAM and a 1TB 15K RPM HD. The OS is Ubuntu 12.04 LTS. Due to space

[9] http://ontop.inf.unibz.it/
[10] http://stardog.com/

Table 9: Hard Queries Rewriting And Unfolding

query	Ext. Reasoning OFF #rw	#un	rw time sec.	un time sec.	Ext. Reasoning ON #rw	#un	rw time sec.	un time sec.
q6	—	48	—	0.1	73	1740	1.8	1.3
q9	—	570	—	0.1	1	150	0	0.03
q10	—	24	—	0.9	1	24	0	0.01
q11	—	1	—	0.1	73	870	0.03	0.7
q12	—	1	—	0.2	10658	5220	525	139

constraints, we present the results for only one running client. We obtained results with the *existential reasoning* on and off.

In order to test the scalability of the systems w.r.t. the growth of the database, we used the data generator described in Section 4.1 and produced several databases, the largest being approximately 1500 times bigger than the original one ("NPD1500" in Table 8, $\approx$ 117 GB of size on disk ($\approx$ 6 Bn triples)).

Table 8 shows the 9 easiest queries from the initial query set, for which the unfolding produces a single select-project-join SQL query. These results were obtained in *Ontop*. The query mix of 9 queries was executed 10 times, 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 the 9 queries can be answered in one hour.

Table 9 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 sub-queries (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.

Table 10 contains results for the 5 hardest queries in *Ontop*. Each query was run once, since qmpH is not so informative in this case. 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 [23]. Hence, purely randomly generated datasets are not appropriate for benchmarking.

Table 11 shows the 6 queries where the performance difference is bigger. As expected, the 3 queries with worst performance in OBDA are those that were affected by the blow-up shown in Table 9. On the other hand, the 3 queries that perform the best are those where the different optimization led to a simple SPJ SQL query. Note how in these 3 queries, in *Ontop*, the overhead caused by the increment of dataset size is minimum.

7 Conclusions and Future Work

The benchmark proposed in this work is the first one that thoroughly analyzes a complete OBDA system in all significant components. So far, little or no work has been done in this direction, as pointed out in [18], since the research community has mostly focused on *rewriting* engines. Thanks to our work, we have gained a better understanding of the current state of the art for OBDA systems: first, we confirm [11] that the unfolding phase is the real bottleneck of modern OBDA systems; second, more research work is needed in order to understand how to improve the design of mappings,

Table 10: Hard queries

query	NPD rp_t/weight R+U (sec./ratio)	NPD2 rp_t/weight R+U (sec./ratio)	NPD5 rp_t/weight R+U (sec./ratio)	NPD10 rp_t/weight R+U (sec./ratio)	NPD10 RAND rp_t/weight R+U (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
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

Table 11: Query executions in Stardog and Ontop (Time in sec.)

Query	NPD1		NPD2		NPD5		NPD10	
	Stardog	Ontop	Stardog	Ontop	Stardog	Ontop	Stardog	Ontop
q1	1.56	0.597	1.616	0.463	1.852	0.683	6.184	0.571
q7	1.585	0.021	2.223	0.041	3.714	0.108	5.199	0.204
q8	0.865	0.08	1.561	0.192	2.504	0.526	6.492	0.669
q6	0.486	1.593	1.592	21.217	2.678	23.272	4.285	88.979
q11	0.394	0.974	1.412	25.693	2.138	197.786	2.584	978.465
q12	0.425	0.934	1.649	275.548	2.65	1396.185	3.464	3292.464
Mater.	54.75s		2m58.014s		9m58.509s		41m23s	
Load	60.935s		4m42.849s		12m8.565s		57m18.297s	

avoiding the use of mappings that give rise to huge queries after unfolding. We conclude by observing that 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.

References

1. 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 2012), vol. 943, pp. 1–15. CEUR, `ceur-ws.org` (2012)
2. Bitton, D., Dewitt, D.J., Turbyfill, C.: Benchmarking database systems: A systematic approach. In: Proc. of VLDB. pp. 8–19 (1983)
3. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. on Semantic Web and Information Systems 5(2), 1–24 (2009)
4. Bruno, N., Chaudhuri, S.: Flexible database generators. In: Proc. of VLDB. pp. 1097–1107 (2005)
5. Bsche, K., Sellam, T., Pirk, H., Beier, R., Mieth, P., Manegold, S.: Scalable generation of synthetic GPS traces with real-life data characteristics. In: Selected Topics in Performance Evaluation and Benchmarking, LNCS, vol. 7755, pp. 140–155. Springer (2013)
6. Calvanese, D., Giese, M., Haase, P., Horrocks, I., Hubauer, T., Ioannidis, Y., Jiménez-Ruiz, E., Kharlamov, E., Kllapi, H., Klüwer, J., Koubarakis, M., Lamparter, S., Möller, R., Neuenstadt, C., Nordtveit, T., Özcep, Ö., Rodriguez-Muro, M., Roshchin, M., Ruzzi, M., Savo, F., Schmidt, M., Soylu, A., Waaler, A., Zheleznyakov, D.: The Optique project: Towards OBDA systems for industry (Short paper). In: Proc. of OWLED. CEUR, `ceur-ws.org`, vol. 1080 (2013)

7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: Tessaris, S., Franconi, E. (eds.) RW Tutorial Lectures, LNCS, vol. 5689, pp. 255–356. Springer (2009)
8. Cosmadakis, S.S., Papadimitriou, C.H.: Updates of relational views. JACM 31(4), 742–760 (1984)
9. Crolotte, A., Ghazal, A.: Introducing skew into the TPC-H Benchmark. In: Topics in Performance Evaluation, Measurement and Characterization, LNCS, vol. 7144, pp. 137–145. Springer (2012)
10. 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/`
11. Di Pinto, F., Lembo, D., Lenzerini, M., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M., Savo, D.F.: Optimizing query rewriting in ontology-based data access. In: Proc. of EDBT. pp. 561–572. ACM Press (2013)
12. 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`
13. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. of Web Semantics 3(2–3), 158–182 (2005)
14. 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)
15. 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)
16. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in *DL-Lite*. In: Proc. of KR. pp. 247–257 (2010)
17. 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)
18. 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)
19. 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)
20. 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)
21. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of ISWC. LNCS, vol. 8218, pp. 558–573. Springer (2013)
22. Skjæveland, M.G., Lian, E.H.: Benefits of publishing the Norwegian Petroleum Directorate's FactPages as Linked Open Data. In: Proc. of Norsk informatikkonferanse (NIK 2013). Tapir (2013)
23. Swami, A., Schiefer, K.B.: On the estimation of join result sizes. In: Proc. of EDBT. LNCS, vol. 779, pp. 287–300. Springer (1994)
24. 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)
25. 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)