Benchmarking RDF Query Engines: The LDBC Semantic Publishing Benchmark

V. Kotsev[1], N. Minadakis[2], V. Papakonstantinou[2], O. Erling[3], I. Fundulaki[2], and A. Kiryakov[1]

[1] Ontotext, Bulgaria
[2] Institute of Computer Science-FORTH, Greece
[3] OpenLink Software, Netherlands

Abstract. The Linked Data paradigm which is now the prominent enabler for sharing huge volumes of data by means of Semantic Web technologies, has created novel challenges for non-relational data management technologies such as RDF and graph database systems. Benchmarking, which is an important factor in the development of research on RDF and graph data management technologies, must address these challenges. In this paper we present the *Semantic Publishing Benchmark* (*SPB*) developed in the context of the Linked Data Benchmark Council (LDBC) EU project. It is based on the scenario of the BBC media organisation which makes heavy use of Linked Data Technologies such as RDF and SPARQL. In *SPB* a large number of aggregation agents provide the heavy query workload, while at the same time a steady stream of editorial agents execute a number of update operations. In this paper we describe the benchmark's schema, data generator, workload and report the results of experiments conducted using *SPB* for the Virtuoso and GraphDB RDF engines.

Keywords: RDF, Linked Data, Benchmarking, Graph Databases

1 Introduction

Non-relational data management is emerging as a critical need in the era of a new data economy where heterogeneous, schema-less, and complexly structured data from a number of domains are published in RDF. In this new environment where the Linked Data paradigm is now the prominent enabler for sharing huge volumes of data, several data management challenges are present and which RDF and graph database technologies are called to tackle. In this context, *benchmarking* is an important factor in the development of research of the aforementioned technologies that are called to address these new challenges.

To test the performance of SPARQL query evaluation techniques, a number of benchmarks have been proposed in the literature [1, 6, 8, 9, 11–13, 15, 10, 18]. Benchmarks can be used to inform users of the strengths and weaknesses of competing software products, but more importantly, they encourage the advancement of technology by providing both academia and industry with clear targets for improving the performance and functionality of the systems.

Existing RDF benchmarks do not fully cover important RDF and SPARQL intricacies since data remains relational at heart, the workload is generally formed by simple read-only query operations, and the queries have not been specifically designed to stress the systems and in particular their debilities. In addition, existing benchmarks do not consider real use case scenarios (except DBSB [6, 13]) and workloads. A number of benchmarks that actually use real ontologies and datasets employ synthetic workloads to test the performance of RDF systems [10, 18] that do not necessarily reflect real world usage scenarios [1, 9, 11, 12, 15].

In this paper we present the **S**emantic **P**ublishing **B**enchmark (*SPB*) developed in the context of the Linked Data Benchmark Council (LDBC)[4] European project. It is inspired by the Media/Publishing industry, and in particular by BBC's[5] *"Dynamic Semantic Publishing" (DSP)* concept. BBC maintains sets of RDF descriptions of its catalogue of *creative works* such as articles, photos, and videos.

SPB is designed to reflect a scenario where a large number of *aggregation* agents provide the heavy query workload, while at the same time a steady stream of *editorial* agents implement *update* operations that *insert* and *delete* creative works. Journalists use the aggregation agents to *query* existing creative works and use the retrieved data in order to create new ones using the *editorial* agents. The Semantic Publishing Benchmark includes:

- a *data generator* that uses *ontologies* and *reference datasets* provided by BBC, to produce sets of *creative works*. The data generator supports the creation of arbitrarily large RDF datasets in the order of billions of triples that mimic the characteristics of the reference BBC datasets.
- the *workload* in *SPB* is defined by the simultaneous execution of *editorial* and *aggregation* agents, simulating a constant load generated by end-users, journalists, editors or automated engines. The workload is designed to reflect a scenario where a large number of *aggregation* agents provide the heavy query workload, while at the same time a steady stream of *editorial* agents implement *update* operations that *insert* and *delete* creative works.
  The *SPB* queries are defined in a way that tackle the *choke points* [3] that each RDF store needs to address in order to satisfy the requirements raised from real world use cases.
- *performance metrics* that describe how fast an RDF database can execute queries by simultaneously running *aggregation* and *editorial* agents.

The paper is structured as follows: Section 2 discusses briefly *SPB*, Section 3 discusses the experiments we conducted running *SPB* for the Virtuoso [17] and GraphDB [14] RDF engines. Related work is presented in Section 4 and conclusions in Section 5.

[4] http://ldbcouncil.org
[5] British Broadcasting Corporation: http://www.bbc.com/

2 Semantic Publishing Benchmark (*SPB*)

The scenario of *SPB* is based around a media organization that maintains RDF descriptions of its catalogue of *creative works* or journalistic assets (articles, photos, videos), papers, books, movies among others. Creative works are valid instances of BBC *ontologies* that define numerous concepts and properties employed to describe this content.

In this section we discuss the *ontologies* and *reference datasets* (Section 2.1) used by *SPB*, its *data generator* (Section 2.2) that uses the ontologies and reference datasets provided by BBC to produce a set of *creative works*, the *queries* that the benchmark introduces (Section 2.3) and finally the employed *performance metrics* (Section 2.4).

2.1 Ontologies & Reference Datasets

SPB uses seven *core* and three *domain* RDF ontologies provided by BBC. The former define the main entities and their properties, required to describe essential concepts of the benchmark namely, *creative works*, *persons*, *documents*, BBC *products* (news, music, sport, education, blogs), *annotations* (*tags*), *provenance* of resources and *content management system* information. The latter are used to express concepts from a *domain of interest* such as football, politics, entertainment among others.

The ontologies are relatively simple: they contain few classes (74), 29 and 88 *data type* and *object* properties respectively, and shallow class and property hierarchies (65 `rdfs:subClassOf`, 17 `rdfs:subPropertyOf`). More specifically, the class hierarchy has a maximum depth of 3 whereas the property hierarchy has a depth of 1. They also contain restrictions (107 `rdfs:domain` and 117 `rdfs:range` RDFS properties). The BBC ontologies also ontain 8 `owl:oneOf` class axioms that allow one to define a class by enumeration of its instances and 2 `owl:TransitiveProperty` properties. Additionally the simplest possible flavor of OWL has been used (`owl:TransitiveProperty`, `owl:sameAs`) for nesting of geographic locations or relations between entities in the reference dataset. A detailed presentation of the ontologies employed by *SPB* can be found in [4].

SPB also uses *reference datasets* that are employed by the data generator to produce the data of interest. These datasets are snapshots of the real datasets provided by BBC; in addition, a GeoNames and DBPedia reference dataset has been included for further enriching the annotations with geo-locations to enable the formulation of geo-spatial queries, and person data.

2.2 Data Generation

The *SPB* data generator produces RDF descriptions of creative works that are valid instances of the BBC ontologies presented previously. A creative work is described by a number of *data value* and *object value properties*; a creative work also has properties that link it to resources defined in *reference* datasets: those are the *about* and *mentions* properties, and their values can be *any resource*. One of the purposes of the data generator is to produce synthetic large (in the

order of billions of triples) datasets in order to check the ability of the engines to *scale*. The generator models three types of relations in the data:

Clustering of data The clustering effect is produced by generating *creative works* about a *single entity from reference* datasets and for a *fixed period of time*. The number of creative works starts with a high peak at the beginning of the clustering period and follows a smooth decay towards its end. The data generator produces major and minor clusterings with sizes (i.e., number of creative works) of different magnitude. By default five major and one hundred minor clusterings of data are produced for one year period. Example of clusterings of data could be *news items* that are about *events* starting with a high number of journalistic assets related to them and following a decay in time as they reach the end of time period, a tendency that mirrors a real world scenario in which a 'fresh' event is popular and its popularity decreases as time goes by.

Correlations of entities This correlation effect is produced by generating *creative works about two or three entities from reference data* in a *fixed period of time*. Each of the entities is *tagged by creative works* solely at the beginning and end of the correlation period and in the middle of it, both are used as tags for the same creative work. By default fifty correlations between entities are modelled for one year period. Such an example of data correlation could be that several 'popular' persons (e.g., Stross-Cahn and Barroso) are mentioned together by creative works for a certain period of time.

Random tagging of entities Random data distributions are defined with a *bias towards popular entities* created when the *tagging* is performed, that is when values are assigned to *about* and *mentions* creative work properties. This is achieved by *randomly* selecting a 5% of all the resources from reference data and mark them as *popular* when the remaining ones are marked as *regular*. When creating creative works, 30% percent of them are tagged with *randomly selected* popular resources and the remaining 70% are linked to the *regular* ones. Example for random taggings could be every-day events which become less important several days after their start date. Distributions of about and mentions tags analysed from a 'live' dataset provided by the BBC are reproduced in generated data. Table 1 shows the distribution of total about and mentions tags found in creative works and also shows their individual distributions.

Additional modification of current version has been added which triples the number of about and mentions tags used in Creative Works, thus providing a better interconnectedness of entities across whole dataset. This random generation of data concerns only one third of all generated data; the remaining data is generated with correlations and clustering effects modeled as previously described.

The *SPB* data generator operates in a sequence of phases:

1. ontologies and reference datasets are *loaded* in an RDF repository;
2. all instances of the domain ontologies that exist in the reference datasets are retrieved by means of predefined SPARQL queries that will be used as values for the *about* and *mentions* properties of creative works;

Amount	Distribution of about and mentions	Distribution of about tags	Distribution of mentions tags
1	22.33 %	10.06 %	94.77 %
2	32.67 %	23.13 %	3.82 %
3	24.60 %	30.88 %	0.93 %
4	11.63 %	22.78 %	0.31 %
5	3.27 %	10.35 %	0.12 %
6	1.52 %	2.80 %	0.05 %
7	1.00 %	0 %	0 %
8	0.74 %	0 %	0 %
9	0.69 %	0 %	0 %
10	0.47 %	0 %	0 %

Table 1: Distribution of 'about' and 'mentions' tags in creative works, analysed in 'live' data provided by the BBC

3. from the previous set of instances, the *popular* and *regular* entities are selected;
4. the generator produces the creative works according to the three properties discussed previously.

2.3 *SPB* Queries

SPB is designed to reflect a scenario where a large number of *aggregation* agents provide the heavy query workload requesting creative works, while at the same time a steady stream of *editorial* agents implement *update* operations that *insert* and *delete* creative works.

Aggregation Queries *SPB* queries are valuable in the sense that they stress important technical functionalities that systems must tackle. P. Boncz [2, 3] uses the terms *choke points* to refer to *difficulties in the workloads* that address elements of particular technical functionality and can be used for designing useful benchmarks. These choke points arise from *data distributions*, the *queries* and *updates*, and the *workloads* that implement the latter two. Examples of such choke points are *aggregation* and *join performance*, *data access locality*, *expression calculation*, *parallelism*, *concurrency* and *correlated subqueries*.

SPB queries are defined in a way that tackle the choke points that each RDF store needs to address in order to satisfy the requirements raised from real world use cases. The benchmark comes with two types of workloads: the *base* and *advanced* versions. The former tackles challenges related to SPARQL query processing, whereas the latter extends the former by including versioning and backup database functionalities. In this paper we restrict ourselves to the presentation of the base version of *SPB* benchmark since it addresses features interesting for SPARQL query processing. A subset of *SPB* queries are based on real ones obtained from the applications that BBC journalists use to access existing creative works in order to create new ones; the remaining queries are based

on these complex queries and are enhanced with different features of SPARQL 1.1.

We first present the choke points (**CP**) *SPB* queries implement and then we provide a detailed explanation of the *SPB* queries and the choke points they implement.

CP1: JOIN ORDERING This choke point tests if the optimizer can evaluate the trade-offs between the time spent to find the best execution plan and the quality of the output plan. It also tests the ability of the engine to consider *cardinality constraints* expressed by the different kinds of properties defined in the ontologies (such as *functional properties* and *inverse functional properties*).

CP2: AGGREGATION Aggregations are implemented with the use of *sub-selects* in the SPARQL query; the optimizer should recognize the operations included in the sub-selects and evaluate them first.

CP3: OPTIONAL AND NESTED OPTIONAL CLAUSES This choke point tests the ability of the optimizer to produce a plan where the execution of the optional triple patterns is the *last to be performed* since optional clauses do not reduce the size of intermediate results.

CP4: REASONING Reasoning tests are used to check whether the systems handle efficiently RDFS and OWL constructs.

CP5: PARALLEL EXECUTION OF UNIONS This choke point tests the ability of the optimizer to produce plans where *unions* are executed in *parallel*. This is especially helpful if the involved subqueries produce a large number of results.

CP6: OPTIONALS WITH FILTERS This specific choke point tests the ability of the engines to execute *as early as possible* such expressions in order to eliminate a possibly large number of intermediate results. Note that this choke point is similar to **CP3** with the difference that here the expression contains filters that bind variables to specific values. Filters reduce (in some cases significantly) the size of the intermediate results. Such queries (related to CP3 and CP6) are of high importance as they often found in real world scenarios.

CP7: ORDERING This test checks the ability of the optimizer to choose query plan(s) that facilitate the *ordering of results*.

CP8: GEO-SPATIAL PREDICATES This choke point tests the ability of the system to handle queries for *geospatial data*; queries that mention entities within a special geospatial range address this technical challenge.

CP9: FULL TEXT SEARCH Queries that involve the *evaluation of regular expressions* on data value properties of resources address this choke point.

CP10: DUPLICATE ELIMINATION This choke point tests the ability of the system to identify duplicate entries and eliminate them during the creation of intermediate results.

CP11: COMPLEX FILTER CONDITIONS Filter conditions that involve negation, conjunction and disjunction can be used to test if the optimizer is able to split the filter conditions into conjunctions of conditions and execute them in parallel and as soon as possible.

In Table 2 we provide the 12 queries that comprise the *base* version of *SPB*. For each of them we provide a text description and we enumerate the *choke*

points it addresses. All the SPARQL queries can be found online[6]. Queries Q_1, Q_2, Q_3, Q_4, Q_8 use the CONSTRUCT, and Q_5, Q_6, Q_7, Q_9, Q_{10}, Q_{11} and Q_{12} use the SELECT SPARQL modifiers.

Query	Description	Choke Points
Q1	Retrieve the 10 most recent creative works, that are *about* or *mention* different topics. For each creative work a *graph* is returned that comprises of the work's *title*, *shortTitle*, *dateCreated*, *dateModified*, *description*, *primaryFormat* and *primaryContentOf* properties; if the creative work has a *thumbnail*, then return its *thumbnailAltText* and *thumbnailType*. Also retrieve existing properties of the *topics* that the creative work is *about* or *mentions*: *aboutLabel*, *aboutShortLabel*, *aboutPreferredLabel*, *mentionsLabel*, *mentionsShortLabel* and *mentionsPreferredLabel*.	**CP1**, **CP2**, **CP3**, **CP4**, **CP5**
Q2	Retrieve details about a given resource that is an instance of class *Creative Work* and its subclasses, namely its *title*, *dateCreated*, *dateModified*, the topic the resource is *about*, its *primaryContentOf* and the *webDocumentType* thereof.	**CP1**, **CP3**, **CP4**
Q3	Retrieve a list of creative works that are instances of classes *BlogPost* or *NewsItem*, that are *about* a specific topic, the value of *primaryFormat* is one of *TextualFormat*, *InteractiveFormat* or *PictureGalleryFormat*. If the creative work has an *audience* then the creative work should be obtained using this value. Given the retrieved list of creative works, return for each resource its related nodes, and then order the result in descending order of the value of their *creationDate* property.	**CP3**, **CP5**, **CP6**, **CP10**, **CP11**
Q4	Return a list of all creative works with all their properties, that are *about* a given topic, have a given *primaryFormat*, and are instances of a certain subclass of class *CreativeWork*. The results should be ordered by property *creationDate* and only N results should be returned.	**CP1**, **CP7**
Q5	Return the list of most popular topics that creative works of a given type are *about*, have a given *audience*, and they have been created in a specific *time range* (*dateModified* is between a start and an end date). For each retrieved topic get its properties (if they exist) *cannonicalName* and *preferredLabel*; and if any of the labels exist, return it as a result along with the count of topics with this label. The result should be returned on descending order of the count of labels.	**CP1**, **CP2**, **CP3**, **CP7**, **CP11**
Q6	Retrieve all instances of class *CreativeWork* that *mention* a geo-location that is within the boundaries of a given rectangle area. Along with each retrieved creative work retrieve property *geonamesId* and its *lat* (latitude) and *long* (longitude) values. Limit the result to 100 creative works (*geo-spatial query*).	**CP7**, **CP8**, **CP10**, **CP11**
Q7	Retrieve properties *dateModif*, *title*, *category*, *liveCoverage*, *audience* for all creative works that are of a given type. The value of property *dateModif* of the retrieved creative works should be within a certain time-range. Return 100 results ordered in ascending order by their *dateModif*.	**CP1**, **CP7**, **CP11**

[6]https://github.com/ldbc/ldbc_spb_bm_2.0/tree/master/datasets_and_queries/sparql/basic/aggregation_standard

Q_8	Retrieve the graphs of resources that are instances of class *CreativeWork* considering also its subclasses that comprise the resources' *type*, *title*, *description*, *dateCreated*, *dateModified*, *category*, the topic they are *about*, their *primaryContentOf* and the latter's *webDocumentType*. Each of the returned resources should contain in its *title* or *description* a given string. (*full text query*)	**CP3**, **CP4**, **CP9**, **CP11**
Q_9	Retrieve 10 similar creative works, by calculating their similarity score. The creative works should be ordered by the computed score and in descending order of the value of property *dateModified*.	**CP2**, **CP7**, **CP10**
Q_{10}	Retrieve creative works that mention locations in the same province (A.ADM1) as the specified one. Additional constraint on time interval further limits returned results.	**CP1**, **CP4**, **CP5**, **CP11**
Q_{11}	Retrieve a list of the most recent Creative Works that have tagged with entities, related to a specific popular entity from reference dataset. Relations can be (inbound and outbound; explicit or inferred)	**CP1**, **CP4**, **CP5**, **CP7**
Q_{12}	Retrieve the descriptions of the latest creative works tagged with a specific location. Consider that the description of each specific *CreativeWork* is stored in dedicated named graph. The result should include only the explicit statements about the creative work, without *owl:sameAs* equivalence and without statements inferred otherwise .	**CP1**, **CP2**, **CP7**, **CP10**

Table 2: *SPB* Queries and their corresponding Choke Points

Table 3 shows the SPARQL features that each of the queries in the *SPB* query mix implement. By COMPLEX FILTERS we refer to filters that contain conjunction and disjunction of predicates, and by NEGATION we refer to the use of negation associated with the BOUND SPARQL operator employed in the filters. Query Q_1 is a very complex query that contains 11 optionals with 4 of them having nested optional clauses. Query Q_3 contains the majority of the SPARQL features (except aggregates, group by, and regular expressions in filters).

Aggregation agents simulate the retrieval operations performed by journalists, end-users or automated search engines by executing a mix of aggregation queries of type: aggregation, search, statistical, full-text, geo-spatial, analytical, drill-down and faceted search. Each aggregation agent will execute a mix of those query types in a constant loop, until the benchmark run finishes. Each agent executes a query and waits for response (or a time-out), after receiving the response next query is executed (queries executed by agents are not of the same type). Query order of execution is pseudo-randomly chosen following an even distribution for each query defined in the benchmark's configuration.

Update Queries *Editorial* agents simulate the editorial work performed by journalists, editors or automated text annotation engines by executing the following update queries:

Feature	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}
OPTIONALS	7	4	1	-	-	-	-	3	4	-	-	-
NESTED OPTIONALS	4	-	-	-	-	-	-	-	-	-	-	-
UNION	-	-	3	-	-	-	-	-	-	1	1	-
ORDER BY	1	-	1	1	1	-	-	1	1	-	1	2
DISTINCT	1	-	1	1	1	1	-	-	5	-	2	1
LIMIT	1	-	1	1	-	-	-	1	1	-	1	1
NESTED QUERIES	1	-	1	1	-	-	-	-	17	-	1	1
GROUP BY	-	-	-	-	1	-	-	-	4	-	-	-
AGGREGATES	-	-	-	-	2	-	-	-	12	-	-	-
NEGATION	-	-	1	-	-	-	-	-	4	-	-	-
COMPLEX FILTERS	-	-	2	1	1	1	-	1	-	-	-	-
REGEXP	-	-	-	-	-	-	-	2	-	-	-	-

Table 3: Characteristics of *SPB* Queries

- **Insert operations** generate new creative work descriptions (content metadata) following the models and distribution rules defined in Section 2.2. Each creative work is added to the database in a single transaction by execution of an insert SPARQL query.
- **Update operations** update an existing creative work. Update operation consists of two actions, executed in one transaction, following the BBC's use-case for update of a creative work. First action is to delete the context where a creative work description resides along with all its content. Second is to insert the same creative work (using its current ID) with all its properties - current and updated ones.
- **Delete operations** delete an existing creative work. Delete operation will erase the context where a creative work resides along with all of its content.

Each editorial agent will execute a mix of editorial operations in a constant loop, until the benchmark run has finished. Editorial operations executed by an agent are chosen pseudo-randomly following the distribution: 80% INSERT operations, 10% UPDATE operations, 10% DELETE operations. Such distribution follow a similar to live datasets pattern where a massive amount of new data being added to the database (inserts and updates) and a minor amount of data being deleted from it. Such rates are open and freely configurable, thus each audited run should include a statement about the exact distribution values.

2.4 Performance Metrics

Performance metrics produced by the *SPB* benchmark describe how fast an RDF database can execute queries (by simultaneously running *aggregation* agents) while at the same time executing editorial operations (by simultaneously running *editorial* agents) and operating over an amount of generated data in the RDF database. The data generator of *SPB* is capable of producing data in different sizes (up to billions of triples), thus allowing to have performance results for various scales. *SPB* outputs two types of performance metrics:

1. *Minimum*, *Maximum* and *Average* execution times for each individual query and editorial operation during the whole benchmark run.
2. *Average* execution rate per second for all queries and editorial operations.

Performance metrics are produced per-second during the benchmark run and are saved to log files. Further all of the information related to query execution and query results is saved in log files with different level of details. Regarding the performance metric (1), due to space restriction, we only present in the experimental evaluation the *average* execution times for the *SPB* workload, but all the results are available online[7].

2.5 Running the benchmark

SPB comes with *test* and *benchmark* drivers both distributed as a jar file along with the BBC ontologies and reference datasets, the queries and updates discussed earlier and the configuration parameters for running the benchmark and for generating the data. The data generator uses configuration files that must be edited appropriately to set the values regarding the dataset size to produce, the number of aggregation and editorial agents, the query timeout etc. The distributions used by the data generator to produce the datasets could also be edited. The benchmark, that is available on GitHub[8], is very simple to run (once the RDF repository used to store the ontologies and the reference datasets is setup, and the configuration files updated appropriately) from command line.

The *test driver* generates *substitution parameters* for the *SPB* queries during data generation. The use of these parameters ensures that the benchmark is *deterministic*, i.e., *different* runs of the benchmark will employ the *same* values, and hence produce comparable results. During one benchmark run the driver goes through an initialization phase where one instance per benchmark query is created by selecting parameters from the respective query parameter file. Once each query executes the last set of parameters, the next time it is executed (and for the same run), it reiterates from the first set of parameters in the list.

The benchmark driver allows one to specify the size of the sets of substitution parameters per query, which are saved in files along with the generated data. The benchmark produces three kinds of files that contain (a) brief information about each executed query, the size of the returned result, and the execution time (b) the detailed log of each executed query and its result and (c) the benchmarking results.

3 Evaluation

In this section we report the results we obtained when running *SPB* for the RDF engines Virtuoso [17] and GraphDB [14]. We used the Virtuoso Opensource Version 7.50.3213. All the experiments that we conducted for Virtuoso, run in a server with 2 Intel® Xeon® CPU E5-2630 at 2.30GHz (a total of 12 cores/24

[7] http://ldbcouncil.org/benchmarks/spb
[8] https://github.com/ldbc/ldbc_spb_bm_2.0

threads) with 192GB of RAM and running CentOS release 6.2 x86_64. The system has 6 2TB SATA-2 and 2 512GB SATA-3 disks under Patsburg 6-Port SATA AHCI Controller.

We used GraphDB, a semantic graph database management system, Version 6.2 Enterprise Edition. The experiments conducted for GraphDB, run on a single Intel® Xeon® CPU E5-1650 v3 at 3.5GHz (a total of 6 cores/12 threads) with 64GB of main memory and running Ubuntu 14.04 x86_64. The system has 1 500GB SATA-3 in 7200RPM for the operating system only and 2 400GB SSDs, Samsung 845DC PRO.

Our aim is not to make a detailed comparison of the performance of the different systems but provide an idea of how they perform regarding the *SPB* benchmark and this is why we did not use the same configuration for Virtuoso and GraphDB. A complete description of the experiments reported here can also be found online (see Section 2.5).

Datasets: For our experiments we produced three datasets of different scales that correspond to 64M (SF1), 256M (SF3) and 1B (SF5) triples. GraphDB has decided to publish results for SF1 (64M) and SF3 (256M) only based on the most common use-cases in the industry where commodity hardware is widely adopted. Table 4 shows the statistics of the datasets produced by the *SPB* data generator that we will use for our experiments. We provide the number of *explicit* (i.e., generated) and *total* triples for each dataset (implicit and explicit), as well as the number of creative works since these are the resources requested and updated by the *SPB* workload. GraphDB supports forward chaining and computes the closure of the dataset during data generation, consequently we report the implicit triples. The slight deviation in numbers comes from the configuration of the data generator e.g. how many threads have been configured to generate the creative works. Note that the data generator produces creative works, each resource associated with 25 to 29 triples. So threads will stop producing creative works once the dataset size as specified in configuration file has been reached. Nevertheless, if some thread is in the middle of generating a creative work, it will generate that last resource beforing ending.

		#explicit	***#implicit***	***#total***	***#cworks***
Virtuoso	SF1	63.709.210	*N/A*	63.709.210	1.489.675
	SF3	255.659.835	*N/A*	255.659.835	8.821.474
	SF5	999.609.164	*N/A*	999.609.164	37.233.029
GraphDB	SF1	64.000.000	69.112.836	133.112.836	1.489.701
	SF3	256.000.000	226.240.341	482.240.341	8.821.390

Table 4: Data Statistics for Virtuoso & GraphDB for Scale Factors 1, 3 and 5

Configuration: For Virtuoso we measured the performance results on an enterprise grade hardware and *SPB* driver configuration of 22r/2w threads,

while GraphDB has chosen to use a modest system configuration and an *SPB* driver configuration of 8r/2w threads (see Table 5 for the configuration details). Both vendors were free to choose their optimal configuration of the reading and writing threads (aggregation and editorial agents) of the *SPB* driver as well as to select the system configuration which will be used for running the benchmark in order to achieve the best performance results.

	Virtuoso			GraphDB	
Scale Factor	SF1	SF3	SF5	SF1	SF3
Aggregation agents	22	22	22	8	8
Editorial agents	2	2	2	2	2
Data generator workers	4	4	4	8	8
Warm up period (min)	10	10	5	10	10
Duration (min)	20	30	20	20	30
Timout (min)	5	5	5	5	5

Table 5: Configuration for Virtuoso & GraphDB for Scale Factors 1, 3 and 5

Workload: In our experiments we used the *Basic Version* of *SPB* that consists of the 12 queries discussed in Section 2.3. Recall that *SPB* queries are expressed in SPARQL 1.1. In the case of Virtuoso that employs backward reasoning and for queries that involved reasoning we used the option `transitive` and enabled `rdfs_rule_set` option that implements the traversal of RDFS hierarchies along the `rdfs:subClassOf` and `rdfs:subPropertyOf` relations.

Data Generation and Loading Times: The time needed to load the ontologies and reference datasets in both systems is in the order of few milliseconds and we do not report it here. The dataset *generation* and *loading* times, are shown in Table 6. Data generation is slower on Virtuoso's hardware because Virtuoso's server uses HDD drive while hardware configuration for GraphDB uses SSD drive and the data generation process is intensive in terms of I/O operations to the storage device Regarding the loading times, GraphDB loads data much slowly (in the case of SF3 the difference is up to one order of magnitude) because of the forward chaining reasoner which materializes each inferred fact when adding new ones to the database, something not
done by Virtuoso.

Performance Metrics: Table 7 shows the average execution rates per second and Table 8 provides the detailed execution times for each of the 12 benchmark queries. These results are provided for both engines and for all of the scale factors (SF1, SF3 and SF5). For SF1, Virtuoso executes 150 queries per second on 22 aggregation agents (reading threads), whereas GraphDB achieves about 100 queries per second using 8 reading threads.

A large difference in the update rate exists for the two engines. Recall that, as described in Section 2.3, the editorial workload consists of 80% of inserting

		Data Generation	Loading
Virtuoso	SF1	6,65	21,1
	SF3	29,02	90,6
	SF5	118,18	400,58
GraphDB	SF1	2,55	46,1
	SF3	14,51	327,57

Table 6: Data Generation and Loading times (in min) for Virtuoso & GraphDB for Scale Factors 1, 3 and 5

		Query rate (queries/second)	**Update rate (operations/second)**
Virtuoso	SF1	149,0385	156,8325
	SF3	80,6158	92,7072
	SF5	32,2789	72,7192
GraphDB	SF1	100,8436	10,1908
	SF3	29,8986	9,5017

Table 7: Average execution rate per second for all queries and editorial operations for Virtuoso & GraphDB

		Query operations												**Editorial op.**		
		Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}	ins.	upd.	del.
Virtuoso	SF1	280	110	183	43	120	104	99	377	99	227	94	30	10	20	11
	SF3	333	103	296	91	394	281	189	717	113	540	106	77	11	23	12
	SF5	466	104	462	183	1020	985	464	1839	173	2034	166	117	21	53	24
GraphDB	SF1	136	5	10	5	14	90	10	79	361	326	9	24	39	78	41
	SF3	202	6	44	24	88	166	76	150	427	2155	24	125	58	126	64

Table 8: Average execution times (in ms) for each individual query and editorial operation for Virtuoso & GraphDB

new facts, 10% of updating and 10% of deleting of existing ones. GraphDB's low update rate is due to the fact that it uses a forward-chaining materialization of inferred statements which adds an overhead when inserting new data.

Virtuoso uses backward-chaining to compute the necessary implicit triples during query execution; this is performed in memory and inserting a new fact in the database does not trigger any materialization of additional knowledge. Consequently, the update rate is higher. This approach saves time during insertion of new data, but could be tricky during query execution: all queries that require materialization Q_1-Q_8 and Q_{11} are slower in Virtuoso than for GraphDB, as such materialization took place during query execution.

Further increasing the scale factor of generated data from SF1 to SF3, which effectively consists of 239M generated statements and 25M statements of reference data, shows a drop in query performance for both GraphDB (60%) and

VIRTUOSO (45%) which is expected because of the four times larger amount of data that the SPB benchmark operates on.

In the reported experiments, VIRTUOSO stores RDF triples in a single RDF quad table with VIRTUOSO's default index configuration and considers hash join as a possible join type. Considering a hash join as a possibility always slows down compilation, and improves some queries and slows others down, not greatly affecting the average throughput. We run additional (unaudited) experiments with VIRTUOSO where we introduced query plan caching that raises the VIRTUOSO score from 80 queries/per second (qps) to 144 qps for SF3. The rationale for considering hash join in the first place is that analytical workloads such as *SPB*'s heavily rely on this. A good TPC-H score is simply unfeasible without this[9] and hash join is indispensible if RDF is to be a serious contender beyond serving lookups. The decision for using this however depends on accurate cardinality estimates on either side of the join. Previous work [19] advocates altogether doing away with a cost model that is unreliable and hard to find for RDF datasets. The present VIRTUOSO approach is that going to rule based optimization is not the preferred solution, but rather using characteristic sets [7] for reducing triples into wider tables, which also cuts down on plan search space and increases reliability of cost estimation. When looking at execution alone, we see that actual database operations are low in the profile, with memory management taking the top 19%. This is due to construct queries allocating small blocks for returning graphs, which is entirely avoidable.

4 State of the art

In this Section we are going to provide a short overview of the existing RDF benchmarks, and an analysis based on the following dimensions: the *datasets* (including data generators in the case of *synthetic* benchmarks) and *query workloads*. In our presentation we distinguish between benchmarks that use *real datasets* and those that produce *synthetic datasets* using special purpose data generators.

DBPedia[10], UniProt KnowledgeBase (UniProtKB) [10] and YAGO [18] are the most well known and widely used real datasets for benchmarking RDF engines. DBPedia extracts structured information from Wikipedia to create a large dataset for benchmarking RDF engines. Although the DBPedia dataset is one of the reference datasets for the Linked Open Data Cloud, there is no clear set of queries for it. The query workload named DBPSB (DBPedia SPARQL Benchmark) was proposed by the University of Leipzig [6] and was derived from the DBPedia query logs; it consists of mostly simple lookup queries and does not consider more complex features such as *inference*. UniProt [10], a high-quality dataset describing protein sequences and related functional information is expressed in RDF. It is one of the central datasets in the bio-medical part of the Linked Open Data Cloud and uses an OWL ontology expressed in a sub-language

[9] http://www.openlinksw.com/dataspace/doc/oerling/weblog/Orri%20Erling%27s%20Blog/1856

[10] DBPedia: http://dbpedia.org/sparql

of OWL-Lite. The queries are mainly lookup queries [16] but some are also used to test the reasoning capabilities of RDF databases (e.g., *taxonomic queries*). Finally, YAGO [18] is another knowledge base that integrates statements from Wikipedia, Wordnet, WordNet Domains, Universal WordNet and GeoNames ontologies. Similar to Uniprot and DBPedia, the YAGO dataset is not accompanied by a set of queries. Neumann et. al. provided eight mostly lookup and join queries for an earlier version of the YAGO ontology, for benchmarking the RDF-3X engine [8].
SPB uses real reference datasets from the BBC to produce synthetic datasets of arbitrary size (up to billions of triples) that mimic the characteristics of the former. The workload comprises of both real and synthetic queries and updates that consider all SPARQL 1.0 operators that are more complex than the workloads proposed in the previously discussed benchmarks.

In addition to benchmarks using real-world datasets *synthetic RDF benchmarks* have been proposed in the literature. The Lehigh University Benchmark (LUBM) [15] uses a simple university domain ontology, provides a *scalable* data generator that can produce several millions of triples, and a set of *test* queries. The LUBM data is regular and hence the benchmark does not explore any of RDF's distinguishing properties that pose interesting challenges for query optimizers. LUBM's workload consists of mainly simple lookup and join queries that retrieve only data triples. The University Ontology Benchmark (UOBM) [5] is based on LUBM; it tackles complex inference and includes queries that address scalability issues in addition to those studied by LUBM. UOBM uses schemas that introduce constructs from OWL Lite and OWL DL sublanguages of OWL. In contrast to LUBM, UOBM queries are designed specifically to consider multiple lookups and complex joins, and it is required that the queries also include at least one different type of OWL inference. SP2Bench [11] builds upon the DBLP simple bibliographic schema. It comes with a generator that produces arbitrarily large datasets by taking into account the constraints expressed in terms of this schema. Unfortunately, the produced data is again relational-like in the sense that the data is well structured, and no RDF schema constructs are considered. The queries employ different SPARQL 1.0 operators and contain long path chains and bushy patterns, in addition to complex combinations of SPARQL filter, optional and bound expressions. Finally, the Berlin SPARQL Benchmark (BSBM) [1, 12] is a broadly accepted and used benchmark built around an e-commerce scenario. It provides a scalable data generator and a test driver, as well as a set of queries that measure the performance of RDF engines for very large datasets but not their ability to perform reasoning tasks. BSBM is the first benchmark to propose a test driver that can be used in a systematic way to test the performance of RDF engines.
SPB goes one step beyond the aforementioned benchmarks given that the generator can produce arbitrarily large datasets (in the order of billions of triples), proposes a much more complex workload than the previous ones consisting of queries that contain a fairly large number of triple patterns considering all SPARQL 1.0 operators as well as nested queries of high complexity.

5 Concluding Remarks

In this paper we presented the **S**emantic **P**ublishing **B**enchmark (*SPB*) developed in the context of the Linked Data Benchmark Council (LDBC) European project inspired by BBC's *Dynamic Semantic Publishing" (DSP)* concept. We presented in detail the benchmark's schema, data generator, query workload and performance metrics, and reported on the results of a set of experiments we conducted against the Virtuoso and GraphDB RDF engines. In the next versions of *SPB* we plan to use more expressive OWL ontologies for representing information. We also plan to improve the modelling of content-to-entity cardinalities as to enrich query sets and realistic query frequencies by taking advantage of FT data and query logs respectively.

Acknowledgments

The work presented in this paper was funded by the FP7 project LDBC (#317548)

References

1. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. IJSWIS 5(2) (2009)
2. Boncz, P.A.: LDBC: benchmarks for graph and RDF data management. In: IDEAS (2013)
3. Boncz, P.A., Neumann, T., Erling, O.: TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In: TPCTC (2013)
4. Fundulaki, I., Martinez, N., Angles, R., Bishop, B., Kotsev, V.: D2.2.2 Data Generator. Tech. rep., Linked Data Benchmark Council (2013), available at http://ldbcouncil.org/sites/default/files/LDBC_D2.2.2.pdf
5. Ma, L., Yang, Y., Qiu, Z., G, Xie, Pan, Y., Liu, S.: Towards a Complete OWL Ontology Benchmark. In: ESWC (2006)
6. Morsey, M., Lehmann, J., Auer, S., Ngomo, A.C.N.: DBpedia SPARQL Benchmark - Performance assessment with real queries on real data. In: ISWC (2011)
7. Neumann, T., Moerkotte, G.: Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins. In: ICDE (2011)
8. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. VLDB Journal 19(1) (2010)
9. Pham, M.D., Boncz, P., Erling, O.: S3G2: a Scalable Structure-correlated Social Graph Generator. In: TPCTC (2012)
10. Redaschi, N., Consortium, U., et al.: Uniprot in rdf: Tackling data integration and distributed annotation with the semantic web (2009)
11. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL performance benchmark. In: ICDE (2009)
12. Berlin SPARQL Benchmark (BSBM). http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/
13. DBSB. http://aksw.org/Projects/DBPSB
14. GraphDB. http://ontotext.com/products/ontotext-graphdb/graphdb-standard/
15. LUBM. http://swat.cse.lehigh.edu/projects/lubm
16. UniProtKB Queries. http://www.uniprot.org/help/query-fields
17. Virtuoso Universal Server. http://virtuoso.openlinksw.com/
18. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: WWW (2007)
19. Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., Boncz, P.A.: Heuristics-based query optimisation for SPARQL. In: EDBT (2012)