Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints

Christian Bizer[1] and Andreas Schultz[1]

[1] Freie Universität Berlin, Web-based Systems Group,
Garystr. 21, 14195 Berlin, Germany
christian.bizer@fu-berlin.de, aschultz@mi.fu-berlin.de

Abstract. The SPARQL Query Language for RDF and the SPARQL Protocol for RDF are implemented by a growing number of storage systems and are used within enterprise and open web settings. As SPARQL is taken up by the community there is a growing need for benchmarks to compare the performance of storage systems that expose SPARQL endpoints via the SPARQL protocol. Such systems include native RDF stores, systems that map relational databases into RDF, and SPARQL wrappers around other kinds of data sources. This paper introduces the Berlin SPARQL Benchmark (BSBM) for comparing the performance of these systems across architectures. The benchmark is built around an e-commerce use case in which a set of products is offered by different vendors and consumers have posted reviews about products. The benchmark query mix illustrates the search and navigation pattern of a consumer looking for a product. After giving an overview about the design of the benchmark, the paper presents the results of an experiment comparing the performance of D2R Server, a relational database to RDF wrapper, with the performance of Sesame, Virtuoso, and Jena SDB, three popular RDF stores.

Keywords: Benchmark, Scalability, Semantic Web, SPARQL, RDF, relational database to RDF mapping

1 Introduction

The Semantic Web is increasingly populated with larger amounts of RDF data. Within the last year, the W3C Linking Open Data[1] community effort has stimulated the publication and interlinkage of datasets amounting to over 2 billion RDF triples. Semantic Web technologies are also increasingly used within enterprise settings[2]. A publicly accessible example of this trend is the Health Care and Life Science demonstration held at the 16th International World Wide Web Conference which involved datasets amounting all together to around 450 million RDF triples[3].

A growing number of storage systems have started to support the SPARQL Query Language for RDF [1] and the SPARQL Protocol for RDF [2]. These systems include

[1] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[2] http://www.w3.org/2001/sw/sweo/public/UseCases/
[3] http://esw.w3.org/topic/HCLS/Banff2007Demo

native RDF triple stores as well as systems that map relational databases into RDF[4]. In the future, these systems may also include SPARQL wrappers around other kinds of data sources such as XML stores or object-oriented databases.

As SPARQL is taken up by the community and as the amount of data that is accessible via SPARQL endpoints increases, there is a growing need for benchmarks to compare the performance of storage systems that expose SPARQL endpoints.

The Berlin SPARQL Benchmark (BSBM) is a benchmark for comparing the performance of systems that expose SPARQL endpoints across architectures. The benchmark is built around an e-commerce use case, where a set of products is offered by different vendors and consumers have posted reviews about products.

The benchmark consists of a data generator and a test driver. The data generator supports the creation of arbitrarily large datasets using the number of products as scale factor. In order to be able to compare the performance of RDF triple stores with the performance of RDF-mapped relational databases or XML stores, the data generator can output RDF, XML and relational representations of the benchmark data.

The test driver executes sequences of parameterized SPARQL queries over the SPARQL protocol against the system under test. In order to simulate a realistic workload, the benchmark query mix emulates the search and navigation pattern of a consumer looking for a product.

The rest of the paper is structured as follows: Section 2 gives an overview about existing benchmarks for Semantic Web technologies. Section 3 describes the design of the Berlin SPARQL benchmark. Section 3.1 gives an overview about the benchmark dataset. Section 3.2 motivates the benchmark query mix and defines the benchmark queries. Section 4 presents the results of an experiment comparing the performance of D2R Server, a relational database to RDF wrapper, with the performance of Sesame, Virtuoso, and Jena SDB, three popular RDF stores. Section 5 discusses the contributions of the paper and outlines our next steps.

2 Related Work

A benchmark is only a good tool for evaluating a system if the benchmark dataset and the workload are similar to the ones expected in the target use case [4]. Thus a variety of benchmarks for Semantic Web technologies have been developed.

A widely used benchmark for comparing the performance, completeness and soundness of OWL reasoning engines is the Lehigh University Benchmark (LUBM) [6]. The LUBM benchmark has been extended in [7] to the University Ontology Benchmark (UOBM) by adding axioms that make use of all OWL Lite and OWL DL constructs. As both benchmarks predate the SPARQL query language, they do not support benchmarking specific SPARQL features such as OPTIONAL filters or DESCRIBE and UNION operators.

An early performance benchmark for SPARQL is the DBpedia Benchmark [8]. The benchmark measures the execution time of 5 queries that are relevant in the context of DBpedia Mobile [9] against parts of the DBpedia dataset.

[4] http://esw.w3.org/topic/Rdb2RdfXG/StateOfTheArt/

A recent SPARQL benchmark is SP^{2}Bench [10]. SP^{2}Bench uses a scalable dataset that reflects the structure of the DBLP Computer Science Bibliography. The benchmark queries are designed for the comparison of different RDF store layouts and RDF data management approaches.

A first benchmark for comparing the performance of relational database to RDF mapping tools with the performance of native RDF stores is presented in [11]. The benchmark focuses on the production of RDF graphs from relational databases and thus only tests SPARQL CONSTRUCT queries.

Further information about RDF benchmarks and current benchmark results are found on the ESW *RDF Store Benchmarking* wiki page [5].

3. Design of the Berlin Benchmark

This section introduces the design goals of the Berlin SPARQL Benchmark and gives an overview about the benchmark dataset, the query mix and the individual benchmark queries.

The Berlin SPARQL Benchmark was designed along three goals: First, the benchmark should allow the comparison of different storage systems that expose SPARQL endpoints across architectures. Testing storage systems with realistic workloads of use case motivated queries is a well established benchmarking technique in the database field and is for instance implemented by the TPC H benchmark [12]. The Berlin SPARQL Benchmark should apply this technique to systems that expose SPARQL endpoints. As an increasing number of Semantic Web applications do not rely on heavyweight reasoning but focus on the integration and visualization of large amounts of data from autonomous data sources on the Web, the Berlin SPARQL Benchmark should not be designed to require complex reasoning but to measure the performance of queries against large amounts of RDF data.

3.1 The Benchmark Dataset

The BSBM dataset is settled in an e-commerce use case in which a set of products is offered by different vendors and consumers have posted reviews about these products on various review sites.

The benchmark dataset can be scaled to arbitrary sizes by using the number of products as scale factor. The data generation is deterministic in order to make it possible to create different representations of exactly the same dataset. Section 2 of the BSBM specification[6] contains the complete schema of the benchmark dataset, all data production rules and the probability distributions that are used for data generation. Due to the restricted space of this paper, we will only outline the schema and the production rules below.

[5] http://esw.w3.org/topic/RdfStoreBenchmarking

[6] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/

The benchmark dataset consists of instances of the following classes: Product, ProductType, ProductFeature, Producer, Vendor, Offer, Review, Reviewer and ReviewingSite.

Products are described by an *rdfs:label* and an *rdfs:comment*. Products have between 3 and 5 textual properties. The values of these properties consist of 5 to 15 words that are randomly chosen from a dictionary. Products have 3 to 5 numeric properties with property values ranging from 1-2000 with a normal distribution. Products have a type that is part of a type hierarchy. The depth and width of this subsumption hierarchy depends on the chosen scale factor. In order to run the benchmark against stores that do not support RDFS inference, the data generator can forward chain the product hierarchy and add all resulting *rdf:type* statements to the dataset. Products have a variable number of product features which depend on the position of a product in the product hierarchy. Each product type and product feature is described by an *rdfs:label* and an *rdfs:comment*.

Products are offered by vendors. Vendors are described by a label, a comment, a homepage URL and a country URI. Offers are valid for a specific period and contain the price and the number of days it takes to deliver the product. The number of offers for a product and the number of offers by each vendor follow a normal distribution.

Reviews are published by reviewers. Reviewers are described by their name, mailbox checksum and the country the reviewer live in. Reviews consist of a title and a review text between 50-300 words. Reviews have up to four ratings with a random integer value between 1 and 10. Each rating is missing with a probability of 0.3. The number of reviews for a product and the number of reviews published by each reviewer follow a normal distribution. Table 1 summarizes the number of instances of each class in BSMB datasets of different sizes.

Table 1. Number of instances in BSBM datasets of different sizes.

Total number of triples	**50K**	**250K**	**1M**	**5M**	**25M**
Number of products	91	477	1,915	9,609	48,172
Number of product features	580	580	1,390	3,307	3,307
Number of product types	13	13	31	73	73
Number of producers	2	10	41	199	974
Number of vendors	2	10	39	196	961
Number of offers	1,820	9,540	38,300	192,180	963,440
Number of reviewers	116	622	2,452	12,351	61,862
Number of reviews	2,275	11,925	47,875	240,225	1,204,300
Total number of instances	**4,899**	**23,177**	**92,043**	**458,140**	**2,283,089**

3.2 The Benchmark Query Mix

In order to simulate a realistic workload on the system under test, the Berlin SPARQL Benchmark executes sequences of parameterized queries that are motivated by the use case. The benchmark queries are designed to emulate the search and

navigation pattern of a consumer looking for a product. In a real world setting, such a query sequence could for instance be executed by a shopping portal which is used by consumers to find products and sales offers.

First, the consumer searches for products that have a specific type and match a generic set of product features. After looking at basic information about some matching products, the consumer gets a better idea of what he actually wants and searches again with a more specific set of features. After going for a second time through the search results, he searches for products matching two alternative sets of features and products that are similar to a product that he likes. After narrowing down the set of potential candidates, the consumer starts to look at offers and recent reviews for the products that fulfill his requirements. In order to check the trustworthiness of the reviews, he retrieves background information about the reviewers. He then decides which product to buy and starts to search for the best prize for this product offered by a vendor that is located in his country and is able to deliver within three days. Tables 2 shows the BSBM query mix resulting from this search and navigation path.

Table 2. The BSBM query mix.

1.	Query 1: Find products for a given set of generic features.
2.	Query 2: Retrieve basic information about a specific product for display purposes.
3.	Query 2: Retrieve basic information about a specific product for display purposes.
4.	Query 3: Find products having some specific features and not having one feature.
5.	Query 2: Retrieve basic information about a specific product for display purposes.
6.	Query 2: Retrieve basic information about a specific product for display purposes.
7.	Query 2: Retrieve basic information about a specific product for display purposes.
8.	Query 4: Find products matching two different sets of features.
9.	Query 2: Retrieve basic information about a specific product for display purposes.
10.	Query 2: Retrieve basic information about a specific product for display purposes.
11.	Query 5: Find products that are similar to a given product.
12.	Query 7: Retrieve in-depth information about a product including offers and reviews.
13.	Query 7: Retrieve in-depth information about a product including offers and reviews.
14.	Query 6: Find products having a label that contains specific words.
15.	Query 7: Retrieve in-depth information about a product including offers and reviews.
16.	Query 7: Retrieve in-depth information about a product including offers and reviews.
17.	Query 8: Give me recent English language reviews for a specific product.
18.	Query 9: Get information about a reviewer.
19.	Query 9: Get information about a reviewer.
20.	Query 8: Give me recent English language reviews for a specific product.
21.	Query 9: Get information about a reviewer.
22.	Query 9: Get information about a reviewer.
23.	Query 10: Get cheap offers which fulfill the consumer's delivery requirements.
24.	Query 10: Get cheap offers which fulfill the consumer's delivery requirements.
25.	Query 10: Get cheap offers which fulfill the consumer's delivery requirements.

Table 3 contains the SPARQL representation of the benchmark queries. Parameters within the queries are enclosed with % chars. During a test run, these parameters are replaced with values from the benchmark dataset.

Table 3. The BSBM benchmark queries

Query 1: Find products for a given set of generic features

```
SELECT DISTINCT ?product ?label
WHERE {
    ?product rdfs:label ?label .
    ?product rdf:type %ProductType% .
    ?product bsbm:productFeature %ProductFeature1% .
    ?product bsbm:productFeature %ProductFeature2% .
    ?product bsbm:productPropertyNumeric1 ?value1 .
  FILTER (?value1 > %x%)}
ORDER BY ?label
LIMIT 10
```

Query 2: Retrieve basic information about a specific product for display purposes

```
SELECT ?label ?comment ?producer ?productFeature ?propertyTextual1
    ?propertyTextual2 ?propertyTextual3 ?propertyNumeric1
    ?propertyNumeric2 ?propertyTextual4 ?propertyTextual5
    ?propertyNumeric4
WHERE {
    %ProductXYZ% rdfs:label ?label .
    %ProductXYZ% rdfs:comment ?comment .
    %ProductXYZ% bsbm:producer ?p .
    ?p rdfs:label ?producer .
    %ProductXYZ% dc:publisher ?p .
    %ProductXYZ% bsbm:productFeature ?f .
    ?f rdfs:label ?productFeature .
    %ProductXYZ% bsbm:productPropertyTextual1 ?propertyTextual1 .
    %ProductXYZ% bsbm:productPropertyTextual2 ?propertyTextual2 .
    %ProductXYZ% bsbm:productPropertyTextual3 ?propertyTextual3 .
    %ProductXYZ% bsbm:productPropertyNumeric1 ?propertyNumeric1 .
    %ProductXYZ% bsbm:productPropertyNumeric2 ?propertyNumeric2 .
  OPTIONAL { %ProductXYZ% bsbm:productPropertyTextual4 ?propertyTextual4 }
  OPTIONAL { %ProductXYZ% bsbm:productPropertyTextual5 ?propertyTextual5 }
  OPTIONAL { %ProductXYZ% bsbm:productPropertyNumeric4 ?propertyNumeric4 }}
```

Query 3: Find products having some specific features and not having one feature

```
SELECT ?product ?label
WHERE {
    ?product rdfs:label ?label .
    ?product rdf:type %ProductType% .
    ?product bsbm:productFeature %ProductFeature1% .
    ?product bsbm:productPropertyNumeric1 ?p1 .
  FILTER ( ?p1 > %x% )
    ?product bsbm:productPropertyNumeric3 ?p3 .
  FILTER (?p3 < %y% )
  OPTIONAL {
      ?product bsbm:productFeature %ProductFeature2% .
      ?product rdfs:label ?testVar }
    FILTER (!bound(?testVar)) }
ORDER BY ?label
LIMIT 10
```

Query 4: Find products matching two different sets of features

```
SELECT ?product ?label
WHERE {
    { ?product rdfs:label ?label .
      ?product rdf:type %ProductType% .
      ?product bsbm:productFeature %ProductFeature1% .
      ?product bsbm:productFeature %ProductFeature2% .
      ?product bsbm:productPropertyNumeric1 ?p1 .
    FILTER ( ?p1 > %x% )
} UNION {
      ?product rdfs:label ?label .
      ?product rdf:type %ProductType% .
```

```
      ?product bsbm:productFeature %ProductFeature1% .
      ?product bsbm:productFeature %ProductFeature3% .
      ?product bsbm:productPropertyNumeric2 ?p2 .
    FILTER ( ?p2> %y% ) }}
ORDER BY ?label
LIMIT 10 OFFSET 10
```

Query 5: Find products that are similar to a given product

```
SELECT DISTINCT ?product ?productLabel
WHERE {
    ?product rdfs:label ?productLabel .
    %ProductXYZ% rdf:type ?prodtype.
    ?product rdf:type ?prodtype .
  FILTER (%ProductXYZ% != ?product)
    %ProductXYZ% bsbm:productFeature ?prodFeature .
    ?product bsbm:productFeature ?prodFeature .
    %ProductXYZ% bsbm:productPropertyNumeric1 ?origProperty1 .
    ?product bsbm:productPropertyNumeric1 ?simProperty1 .
  FILTER (?simProperty1 < (?origProperty1 + 150) && ?simProperty1 >
    (?origProperty1 - 150))
    %ProductXYZ% bsbm:productPropertyNumeric2 ?origProperty2 .
    ?product bsbm:productPropertyNumeric2 ?simProperty2 .
  FILTER (?simProperty2 < (?origProperty2 + 220) && ?simProperty2 >
    (?origProperty2 - 220)) }
ORDER BY ?productLabel
LIMIT 5
```

Query 6: Find products having a label that contains specific words

```
SELECT ?product ?label
WHERE {
    ?product rdfs:label ?label .
    ?product rdf:type bsbm:Product .
  FILTER regex(?label, "%word1%|%word2%|%word3%")}
```

Query 7: Retrieve in-depth information about a product including offers and reviews

```
SELECT ?productLabel ?offer ?price ?vendor ?vendorTitle ?review
    ?revTitle ?reviewer ?revName ?rating1 ?rating2
WHERE {
    %ProductXYZ% rdfs:label ?productLabel .
  OPTIONAL {
      ?offer bsbm:product %ProductXYZ% .
      ?offer bsbm:price ?price .
      ?offer bsbm:vendor ?vendor .
      ?vendor rdfs:label ?vendorTitle .
      ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#DE>.
      ?offer dc:publisher ?vendor .
      ?offer bsbm:validTo ?date .
    FILTER (?date > %currentDate% ) }
  OPTIONAL {
      ?review bsbm:reviewFor %ProductXYZ% .
      ?review rev:reviewer ?reviewer .
      ?reviewer foaf:name ?revName .
      ?review dc:title ?revTitle .
    OPTIONAL { ?review bsbm:rating1 ?rating1 . }
    OPTIONAL { ?review bsbm:rating2 ?rating2 . } } }
```

Query 8: Give me recent English language reviews for a specific product

```
SELECT ?title ?text ?reviewDate ?reviewer ?reviewerName ?rating1
    ?rating2 ?rating3 ?rating4
WHERE {
    ?review bsbm:reviewFor %ProductXYZ% .
    ?review dc:title ?title .
    ?review rev:text ?text .
  FILTER langMatches( lang(?text), "EN" )
    ?review bsbm:reviewDate ?reviewDate .
    ?review rev:reviewer ?reviewer .
```

```
    ?reviewer foaf:name ?reviewerName .
  OPTIONAL { ?review bsbm:rating1 ?rating1 . }
  OPTIONAL { ?review bsbm:rating2 ?rating2 . }
  OPTIONAL { ?review bsbm:rating3 ?rating3 . }
  OPTIONAL { ?review bsbm:rating4 ?rating4 . } }
ORDER BY DESC(?reviewDate)
LIMIT 20
```

Query 9: Get information about a reviewer.

```
DESCRIBE ?x
WHERE {
    %ReviewXYZ% rev:reviewer ?x }
```

Query 10: Get cheap offers which fulfill the consumer's delivery requirements.

```
SELECT DISTINCT ?offer ?price
WHERE {
    ?offer bsbm:product %ProductXYZ% .
    ?offer bsbm:vendor ?vendor .
    ?offer dc:publisher ?vendor .
    ?vendor bsbm:country %CountryXYZ% .
    ?offer bsbm:deliveryDays ?deliveryDays .
  FILTER (?deliveryDays <= 3)
    ?offer bsbm:price ?price .
    ?offer bsbm:validTo ?date .
  FILTER (?date > %currentDate% ) }
ORDER BY ?price
LIMIT 10
```

Table 4. Characteristics of the BSBM Benchmark Queries.

Characteristic	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Simple Filters	√		√	√		√	√	√	√	√
Complex Filters					√					
More than 9 triple patterns		√				√	√	√		
OPTIONAL operator		√	√			√	√	√		
LIMIT modifier			√	√	√		√	√		√
ORDER BY modifier			√	√	√		√	√		√
DISTINCT modifier	√				√					√
REGEX operator						√				
UNION operator				√						
DESCRIBE operator									√	

4 Benchmark Experiment

Today, most enterprise data is stored in relational databases. In order to prevent synchonization problems, it is preferable in many situations to have direct SPARQL access to this data without having to replicate it into RDF. We were thus interested in comparing the performance of relational database to RDF wrappers with the performance of native RDF stores. In order to get an idea which architecture performs better for which types of queries and for which dataset sizes, we ran the Berlin SPARQL Benchmark against D2R Server[7], a database to RDF wrapper which

[7] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/

rewrites SPARQL queries into SQL queries against an application-specific relational schemata based on a mapping, as well as against Sesame[8], Virtuoso[9], and Jena SDB[10], three popular RDF stores.

4.1 Experimental Setup

The experiment was conducted on a DELL workstation (processor: Intel Core 2 Quad Q9450 2.66GHz; memory: 8GB DDR2 667; hard disks: 160GB (10,000 rpm) SATA2, 750GB (7,200 rpm) SATA2) running Ubuntu 8.04 64-bit as operating system. We used the following releases of the systems under test:

1) **Jena SDB** Version 1.1 (Hash Storage Layout) with MySQL Version 5.0.51a-3 as underlying RDMS and Joseki Version 3.2 (CVS) as HTTP interface.
2) **Sesame** Version 2.2-Beta2 (Native Storage) with Tomcat Version 5.5.25.5 as HTTP interface.
3) **Virtuoso** Open-Source Edition v5.0.6 (Native Storage)
4) **D2R Server** Version 0.4 (Standalone Setup) with MySQL Version 5.0.51a-3 as underlying RDMS.

The load performance of the systems was measured by loading the N-Triple representation of the BSBM datasets into the triple stores and by loading the relational representation in the form of MySQL dumps into the RDMS behind D2R Server. The loaded datasets were forward chained and contained all *rdf:type* statements for product types. Thus the systems under test did not have to do any inferencing.

The query performance of the systems was measured by running 50 BSBM query mixes (altogether 1250 queries) against the systems over the SPARQL protocol. The test driver and the system under test (SUT) were running on the same machine in order to reduce the influence of network latency. In order to enable the SUTs to load parts of the working set into main memory and to take advantage of query result and query execution plan caching, 10 BSBM query mixes (altogether 250 queries) were executed for warm-up before the actual times were measured.

4.2 Results and Interpretation

This section summarizes the results of the experiment and gives a first interpretation. The complete run logs of the experiment and the exact configuration of the SUTs are found on the BSBM website.

4.2.1 Load Times

Table 5 summarizes the time it took to load the N-Triple files and the MySQL dumps into the different stores. For all dataset sizes, loading the MySQL dump was significantly faster than loading the N-Triple representation of the benchmark data.

[8] http://www.openrdf.org/about.jsp
[9] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/
[10] http://jena.hpl.hp.com/wiki/SDB

When comparing the load performance of the triple stores, it turns out that Sesame and Virtuoso are faster for small datasets while Jena SDB is faster for larger datasets.

Table 5. Load times for different stores and dataset sizes (in seconds).

	50K	250K	1M	5M	25M
Jena SDB	5	24	116	1053	13306
Sesame	3	18	132	1988	27674
Virtuoso	2	33	87	609	49096
D2R Server	0.4	2	6	38	202

4.2.4 Overall Run Time

Table 6 contains the overall times for running 50 query mixes against the stores. It turns out that Sesame is the fastest RDF store for small dataset sizes. Virtuoso is the fastest RDF store for big datasets, but is rather slow for small ones. According to feedback from the Virtuoso team, this is due to the time spent on compiling the query and deciding on the query execution plan being independent of the dataset size.

Compared to the RDF stores, D2R Server showed an inferior overall performance. When running the query mix against the 25M dataset, query 5 did hid the 30 seconds time out and was excluded from the run. Thus there is no overall run time figure for D2R Server at this dataset size.

Table 6. Overall runtime for executing 50 query mixes (in seconds).

	Sesame	Virtuoso	Jena SDB	D2R Server
50 K	9.4	162.0	23.4	66.4
250 K	28.5	162.8	72.9	153.4
1 M	115.2	201.4	268.0	484.4
5 M	569.1	476.8	1406.6	2188.1
25 M	3038.205	2089.118	7623.958	not applicable

4.2.3 Average Run Time Per Query

The picture drawn by the overall run times radically changed when we examined the run times of specific queries. It turned out that no store is superior for all queries. Sesame is the fastest store for queries 1 to 4 at all dataset sizes. Sesame shows a bad performance for queries 5 and 9 against large datasets which prevents the store from having the best overall performance for all dataset sizes. D2R Server, which showed an inferior overall runtime, is the fastest store for queries 6 to 10 against the 25M dataset. Jena SDB turns out to be the fastest store for query 9, while Virtuoso is good at queries 5 and 6. The tables below contain the arithmetic means of the query execution times over all 50 runs per query (in seconds). The best performance figure for each dataset size is set bold in the tables.

Query 1: Find products for a given set of generic features.

	50K	250K	1M	5M	25M
Sesame	**0.002106**	**0.002913**	**0.003904**	**0.005051**	**0.035575**
Virtuoso	0.017650	0.017195	0.015160	0.020225	0.082935
SDB	0.002559	0.004955	0.012592	0.057347	0.328271
D2R	0.004810	0.011643	0.038524	0.195950	0.968688

Query 2: Retrieve basic information about a specific product for display purposes.

	50K	250K	1M	5M	25M
Sesame	**0.005274**	**0.004904**	**0.005523**	**0.006115**	**0.017859**
Virtuoso	0.071556	0.069347	0.069178	0.077868	0.085244
SDB	0.023058	0.024708	0.042374	0.071731	0.446813
D2R	0.038768	0.041448	0.047844	0.056018	0.041420

Query 3: Find products having some specific features and not having one feature.

	50K	250K	1M	5M	25M
Sesame	**0.002364**	**0.003793**	**0.004681**	**0.010618**	**0.049690**
Virtuoso	0.009790	0.008394	0.008904	0.011443	0.055770
SDB	0.005324	0.006671	0.014379	0.059678	0.317215
D2R	0.007125	0.021041	0.054744	0.226272	1.080856

Query 4: Find products matching two different sets of features.

	50K	250K	1M	5M	25M
Sesame	**0.002595**	**0.004023**	**0.005491**	**0.009803**	**0.071493**
Virtuoso	0.016189	0.012962	0.013824	0.020450	0.100784
SDB	0.005694	0.008284	0.023286	0.112666	0.679248
D2R	0.008209	0.021871	0.075406	0.390003	1.935566

Query 5: Find products that are similar to a given product.

	50K	250K	1M	5M	25M
Sesame	**0.057673**	**0.310220**	1.313046	6.688869	32.895117
Virtuoso	0.221498	0.380958	**0.846915**	**3.344963**	**1.623720**
SDB	0.129683	0.509083	1.741927	8.199694	43.878071
D2R	0.718670	2.373025	8.639514	41.823326	time out

Query 6: Find products having a label that contains specific words.

	50K	250K	1M	5M	25M
Sesame	0.004993	0.014792	0.064516	0.290293	1.483277
Virtuoso	0.003639	0.007368	0.020584	**0.109199**	**0.536659**
SDB	**0.003332**	**0.005291**	**0.014744**	0.254799	1.197798
D2R	0.009314	0.013181	0.038695	0.144180	0.383284

Query 7: Retrieve in-depth information about a product including offers and reviews.

	50K	250K	1M	5M	25M
Sesame	**0.005265**	**0.005510**	**0.010948**	**0.007205**	0.505330
Virtuoso	0.125126	0.107109	0.125771	0.310843	2.290610
SDB	0.020116	0.078490	0.402448	2.193056	13.129511
D2R	0.028352	0.030421	0.068561	0.069431	**0.055678**

Query 8: Give me recent English language reviews for a specific product.

	50K	250K	1M	5M	25M
Sesame	**0.005111**	**0.005286**	**0.005770**	**0.006150**	0.315689
Virtuoso	0.026703	0.023340	0.024814	0.028546	4.535478
SDB	0.046099	0.159153	0.660130	3.535463	20.466382
D2R	0.058440	0.069917	0.078460	0.094175	**0.066114**

Query 9: Get information about a reviewer.

	50K	250K	1M	5M	25M
Sesame	0.010526	0.040371	0.202310	1.071040	5.728123
Virtuoso	0.039994	0.039972	0.039744	0.042302	0.106834
SDB	**0.006450**	**0.004545**	**0.004461**	**0.004560**	0.018882
D2R	0.016164	0.015216	0.015215	0.026702	**0.017857**

Query 10: Get cheap offers which fulfill the consumer's delivery requirements.

	50K	250K	1M	5M	25M
Sesame	**0.002722**	**0.002589**	**0.003027**	**0.002814**	0.179580
Virtuoso	0.585728	0.569584	0.642199	1.338361	6.708351
SDB	0.009257	0.033867	0.102852	1.028677	3.141508
D2R	0.005088	0.005209	0.004926	0.005565	**0.017070**

5 Conclusion and Future Work

This paper introduced the Berlin SPARQL Benchmark for comparing the performance of different types of storage systems that expose SPARQL endpoints and presented the results of an initial benchmark experiment. The paper provides the following contributions to the field of SPARQL benchmarking:

1. The W3C Data Access Working Group has developed the SPARQL query language together with the SPARQL protocol in order to enable data access over the Web. The Berlin SPARQL Benchmark is the first benchmark that measures the performance of systems that use both technologies together to facilitate data access over the Web.
2. The Berlin Benchmark provides an RDF triple, an XML and a relational representation of the benchmark dataset. The Benchmark can thus be used to

compare the performance of storage systems that use different internal data models.

3. Testing storage systems with realistic workloads of use case motivated queries is a well established technique in the database field and is for instance used by the widely accepted TPC H benchmark [12]. The Berlin SPARQL Benchmark is the first benchmark that applies this technique to systems that expose SPARQL endpoints.

The benchmark experiment has shown that no store is dominant for all queries. Concerning the performance of relational database to RDF wrappers, D2R Server has shown a very good performance for specific queries against large datasets but was very slow at others. For us as the development team behind D2R Server it will be very interesting to further analyze these results and to improve D2R Server's SPARQL to SQL rewriting algorithm accordingly. As next steps for the benchmark, we plan to:

1. Run the benchmark with 100M, 250M and 1B triple datasets against all stores in order to find out at which dataset size the query times become unacceptable.
2. Run the benchmark against other relational database to RDF wrappers in order to make a contribution to the current work of the W3C RDB2RDF Incubator Group[11].
3. Run the benchmark against further RDF stores and test different Sesame and Jena SDB storage layouts and setups.
4. Extend the test driver with multi-threading features in order to simulate multiple clients simultaneously working against a server.
5. Extend the benchmark to the Named Graphs data model in order to compare RDF triple stores and Named Graph stores.

The complete benchmark specification, current benchmark results, benchmark datasets, detailed run logs as well as the source code of the data generator and test driver (GPL license) are available from the Berlin SPARQL Benchmark website at http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/

Acknowledgments. We would like to thank Lilly and Company and especially Susie Stephens from Lilly for making this work possible with a research grant. We also would like to thank Orri Erling, Andy Seaborne, Michael Schmidt, Richard Cyganiak, and Ivan Mikhailov for their feedback on the benchmark design and the experiment.

References

1. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation, http://www.w3.org/TR/rdf-sparql-query/, 2008.
2. Kendall, G. C., Feigenbaum, L., Torres, E.: SPARQL Protocol for RDF. W3C Recommendation, http://www.w3.org/TR/rdf-sparql-protocol/, 2008.

[11] http://www.w3.org/2005/Incubator/rdb2rdf/

3. Boag, S., et al: XQuery 1.0: An XML Query Language. W3C Recommendation, http://www.w3.org/TR/xquery/, 2007.
4. Yuanbo Guo et al: A Requirements Driven Framework for Benchmarking Semantic Web Knowledge Base Systems. IEEE Transactions on Knowledge and Data Engineering, issue 19(2), pp 297-309, 2007.
5. Weithöner, T., et al.: What's Wrong with OWL Benchmarks? In Proceedings of the 2nd International Workshop on Scalable Semantic Web Knowledge Base Systems, pp 101-114, 2006.
6. Guo, Y., Pan, Z., Heflin J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics, issue 3(2), pp 158-182, 2005.
7. Li Ma et al.: Towards a Complete OWL Ontology Benchmark (UOBM). In: The Semantic Web: Research and Applications , LNCS vol. 4011/2006, pp 125-139, 2006.
8. Becker, C.: RDF Store Benchmarks with DBpedia comparing Virtuoso, SDB and Sesame. http://www4.wiwiss.fu-berlin.de/benchmarks-200801/, 2008.
9. Becker, C., Bizer, C.: DBpedia Mobile: A Location-Enabled Linked Data Browser. In: Proceedings of the 1st Workshop about Linked Data on the Web (LDOW2008), 2008.
10. Schmidt, M., Hornung, T., Küchlin, N., Lausen, G., Pinkel, C.: An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In: Proceedings of the International Semantic Web Conference (ISWC 2008), 2008
11. Svihala, M., Jelinek, I.: Benchmarking RDF Production Tools. In: Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA 2007), 2007
12. Transaction Processing Performance Council: TPC Benchmark H, Standard Specification Revision 2.7.0, http://www.tpc.org/tpch/spec/tpch2.7.0.pdf, 2008.