The Berlin SPARQL Benchmark

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 as well as systems that rewrite SPARQL queries to SQL queries against non-RDF relational databases. This article introduces the Berlin SPARQL Benchmark (BSBM) for comparing the performance of native RDF stores with the performance of SPARQL-to-SQL rewriters 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 emulates the search and navigation pattern of a consumer looking for a product. The article discusses the design of the BSBM benchmark and presents the results of a benchmark experiment comparing the performance of four popular RDF stores (Sesame, Virtuoso, Jena TDB, and Jena SDB) with the performance of two SPARQL-to-SQL rewriters (D2R Server and Virtuoso RDF Views) as well as the performance of two relational database management systems (MySQL and Virtuoso RDBMS).

Keywords: Benchmark, scalability, Semantic Web, SPARQL, RDF, relational database to RDF mapping, SPARQL to SQL rewriting

1. Introduction

The SPARQL Query Language for RDF (Prud'hommeaux & Seaborne 2008) and the SPARQL Protocol for RDF (Kendall et al, 2008) are increasingly used as a standardized query API for providing access to datasets on the public Web[1] and within enterprise settings[2]. Today, most enterprise data is stored in relational databases. In order to prevent synchronization problems, it is preferable in many situations to have direct SPARQL access to this data without having to replicate it into RDF. Such direct access can be provided by SPARQL-to-SQL rewriters that

[1] http://esw.w3.org/topic/SparqlEndpoints
[2] http://www.w3.org/2001/sw/sweo/public/UseCases/

translate incoming SPARQL queries on the fly into SQL queries against an application-specific relational schema based on a mapping. The resulting SQL queries are then executed against the legacy database and the query results are transformed into a SPARQL result set. An overview of existing work in this space has been gathered by the W3C RDB2RDF Incubator Group[3] and is presented in (Sahoo et al., 2009).

This article introduces the Berlin SPARQL Benchmark (BSBM) for comparing the SPARQL query performance of native RDF stores with the performance of SPARQL-to-SQL rewriters. The benchmark aims to assist application developers in choosing the right architecture and the right storage system for their requirements. The benchmark might also be useful for the developers of RDF stores and SPARQL-to-SQL rewriters as it reveals the strengths and weaknesses of current systems and might help to improve them in the future.

The Berlin SPARQL Benchmark was designed in accordance with three goals:

1. The benchmark should allow the comparison of storage systems that expose SPARQL endpoints across architectures.
2. The benchmark should simulate an enterprise setting where multiple clients concurrently execute realistic workloads of use case motivated queries against the systems under test.
3. As the SPARQL query language and the SPARQL protocol are often used within scenarios that do not rely on heavyweight reasoning but focus on the integration and visualization of large amounts of data from multiple data sources, the BSBM benchmark should not be designed to require complex reasoning but to measure SPARQL query performance against large amounts of RDF data.

The BSBM 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 query mix emulates the search and navigation pattern of a consumer looking for a product.

The implementation of 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 stores with the performance of SPARQL-to-SQL rewriters, the data generator can output two representations of the benchmark data: An RDF representation and a purely relational representation.

The test driver executes sequences of SPARQL queries over the SPARQL protocol against the system under test (SUT). In order to emulate a realistic workload, the test driver can simulate multiple clients that concurrently execute query mixes against the SUT. The queries are parameterized with random values from the benchmark dataset, in order to make it more difficult for the SUT to apply caching techniques. The test driver executes a series of warm-up query mixes before the actual performance is measured in order to benchmark systems under normal working conditions.

The BSBM benchmark also defines a SQL representation of the query mix, which the test driver can execute via JDBC against relational databases. This allows the comparison of SPARQL results with the performance of traditional RDBMS.

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

This article makes the following contributions to the field of benchmarking Semantic Web technologies:

1. It complements the field with a use case driven benchmark for comparing the SPARQL query performance of native RDF stores with the performance of SPARQL-to-SQL rewriters.
2. It provides guidance to application developers by applying the benchmark to measure and compare the performance of four popular RDF stores, two SPARQL-to-SQL rewriters and two relational database management systems.

The remainder of the paper is structured as follows: Section 2 gives an overview of the benchmark dataset. Section 3 motivates the benchmark query mix and defines the benchmark queries. Section 4 compares the BSBM benchmark with other benchmarks for Semantic Web technologies. As a proof of concept, Sections 5 and 6 present the results of an experiment that applies the BSBM benchmark to compare the performance of RDF stores and SPARQL-to-SQL rewriters, and sets the results into relation to the performance of RDBMS.

2. The Benchmark Dataset

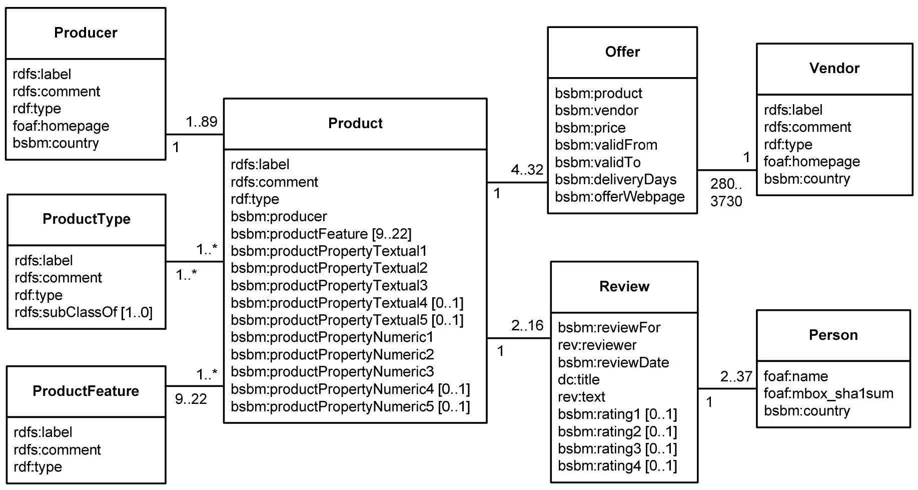
The BSBM benchmark is settled in an e-commerce use case in which a set of products is offered by different vendors and consumers have posted reviews about these products on various review sites. The benchmark defines an abstract data model for this use case together with data production rules that allow benchmark datasets to be scaled to arbitrary sizes using the number of products as scale factor. In order to compare RDF stores with SPARQL-to-SQL rewriters, the benchmark defines two concrete representations of the abstract model: An RDF representation and a relational representation.

The data model contains the following classes: *Product, ProductType, ProductFeature, Producer, Vendor, Offer, Review,* and *Person*. Figure 1 gives an overview of the properties of each class, the multiplicity of properties, and the multiplicity ranges into which 99% of the associations between classes fall. In the following we describe the data production rules that are used to generate datasets for a given scale factor *n*.

The data generator creates *n* product instances. Products are described by a rdfs:label and a rdfs:comment. Products have between 3 and 5 textual properties. The values of these properties consist of 5 to 15 words which are randomly chosen from a dictionary. Products have 3 to 5 numeric properties with property values ranging from 1 to 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. The depth of the hierarchy is calculated as $d = round(log10(n))/2 + 1$. The branching factor for the root level of the hierarchy is $bfr = 2*round(log10(n))$. The branching factor for all other levels is 8. Every product has one leaf-level product type. 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.

Figure 1. Overview of the abstract data model.



Products have a variable number of product features. Two products that share the same product type also share the same set of possible product features. This set is determined as follows: Each product type in the type hierarchy is assigned with a random number of product features. The range of these random numbers is calculated for product types on level *i* of the hierarchy as *lowerBound = 35 * i / (d * (d+1)/2 – 1)* and *upperBound = 75 * i / (d * (d+1)/2 – 1)*, with *d* being the depth of the hierarchy. The set of possible features for a specific product type is the union of the features of this type and all its super-types. For a specific product, each feature from this set is picked with a probability of 25%.

Products are produced by producers. The number of products per producer follows a normal distribution with a mean of μ = 50 and a standard deviation of σ = 16.6. New producers are created until all products are assigned to a producer.

Products are offered by vendors. Vendors are described by a label, a comment, a homepage URL and a country URI. Countries have the following distribution: US 40%, UK 10%, JP 10%, CN 10%, 5% DE, 5% FR, 5% ES, 5% RU, 5% KR, 5% AT.

There are 20 times *n* offers. Offers are valid for a specific period and contain a price ($5-$10000) and the number of days it takes to deliver the product (1-21). Offers are distributed over products using a normal distribution with the parameters μ = *n*/2 and σ = *n*/4. The number of offers per vendor follows a normal distribution with the parameters μ = 2000 and σ = 667. New vendors are created until all offers are assigned to a vendor.

Reviews consist of a title and a review text between 50 and 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 30%. There are 10 times the scale factor *n* reviews. The reviews are distributed over products using a normal distribution with the parameters $\mu = n/2$ and $\sigma = n/4$. The number of reviews per reviewer is randomly chosen from a normal distribution with the parameters $\mu = 20$ and $\sigma = 6.6$. New reviewers are generated until each review is assigned. Reviewers are described by their name, mailbox checksum and the country the reviewer lives in. The reviewer countries follow the same distribution as the vendor countries.

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	**250K**	**1M**	**25M**	**100M**
Number of products	666	2,785	70,812	284,826
Number of product features	2,860	4,745	23,833	47,884
Number of product types	55	151	731	2011
Number of producers	14	60	1422	5,618
Number of vendors	8	34	722	2,854
Number of offers	13,320	55,700	1,416,240	5,696,520
Number of reviewers	339	1432	36,249	146,054
Number of reviews	6,660	27,850	708,120	2,848,260
Total number of instances	**23,922**	**92,757**	**2,258,129**	**9,034,027**

The BSBM data generator can output an RDF representation and a relational representation of benchmark datasets. As the data production rules are deterministic, it is possible to create RDF and relational representations of exactly the same data.

3. The Query Mix

There are two principle options for the design of benchmark query mixes (Gray, 1993): 1. Design the queries to test specific features of the query language or to test specific data management approaches. 2. Base the query mix on the specific requirements of a real world use case. The second approach leads to sequences of more complex queries that test combinations of different language features. With SP^{2}Bench (Schmidt, et al., 2008a and 2008b), there exists already a benchmark for SPARQL stores that is designed for the comparison of different RDF data management approaches. We therefore decided to follow the second approach and designed the BSBM query mix as a sequence of use case motivated queries that simulate a realistic workload against the SUT.

The query mix emulates 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 price for this product offered by a vendor that is located in his country and is able to deliver within three days. After choosing a specific offer, he retrieves all information about the offer and then transforms the information into another schema in order to save it locally for future reference. Table 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 4: Find products matching two different sets of features.
8.	Query 2: Retrieve basic information about a specific product for display purposes.
9.	Query 2: Retrieve basic information about a specific product for display purposes.
10.	Query 5: Find products that are similar to a given product.
11.	Query 7: Retrieve in-depth information about a product including offers and reviews.
12.	Query 7: Retrieve in-depth information about a product including offers and reviews.
13.	Query 6: Find products having a label that contains a specific string.
14.	Query 7: Retrieve in-depth information about a product including offers and reviews.
15.	Query 7: Retrieve in-depth information about a product including offers and reviews.
16.	Query 8: Give me recent English language reviews for a specific product.
17.	Query 9: Get information about a reviewer.
18.	Query 9: Get information about a reviewer.
19.	Query 8: Give me recent English language reviews for a specific product.
20.	Query 9: Get information about a reviewer.
21.	Query 9: Get information about a reviewer.
22.	Query 10: Get cheap offers which fulfill the consumer's delivery requirements.
23.	Query 10: Get cheap offers which fulfill the consumer's delivery requirements.
24.	Query 11: Get all information about an offer.
25.	Query 12: Export information about an offer into another schema.

The BSBM benchmark defines two representations of the query mix: A SPARQL representation for benchmarking RDF stores and SPARQL-to-SQL rewriters, and a SQL representation for benchmarking RDBMS.

SPARQL Representation

Table 3 contains the SPARQL representation of the benchmark queries. The benchmark queries contain parameters which are enclosed with % chars in the table. During a test run, these parameters are replaced with random values from the benchmark dataset. Queries within two consecutive query mixes differ by the chosen parameters which makes it harder for SUTs to apply query caching. As the test driver uses a deterministic randomizer, the overall query sequence is the same for test runs against different SUTs.

Table 3. SPARQL representation of the BSBM 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 .
  FILTER (%ProductXYZ% != ?product)
    %ProductXYZ% bsbm:productFeature ?prodFeature .
    ?product bsbm:productFeature ?prodFeature .
    %ProductXYZ% bsbm:productPropertyNumeric1 ?origProperty1 .
    ?product bsbm:productPropertyNumeric1 ?simProperty1 .
  FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 >
    (?origProperty1 - 120))
    %ProductXYZ% bsbm:productPropertyNumeric2 ?origProperty2 .
    ?product bsbm:productPropertyNumeric2 ?simProperty2 .
  FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 >
    (?origProperty2 - 170)) }
ORDER BY ?productLabel
LIMIT 5
```

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

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

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 xsd:double(str(?price))
LIMIT 10
```

Query 11: Get all information about an offer.

```
SELECT ?property ?hasValue ?isValueOf
WHERE {
    { %OfferXYZ% ?property ?hasValue }
  UNION
    { ?isValueOf ?property %OfferXYZ% } }
```

Query 12: Export information about an offer into another schema.

```
CONSTRUCT {
    %OfferXYZ% bsbm-export:product ?productURI .
    %OfferXYZ% bsbm-export:productlabel ?productlabel .
    %OfferXYZ% bsbm-export:vendor ?vendorname .
    %OfferXYZ% bsbm-export:vendorhomepage ?vendorhomepage .
    %OfferXYZ% bsbm-export:offerURL ?offerURL .
    %OfferXYZ% bsbm-export:price ?price .
    %OfferXYZ% bsbm-export:deliveryDays ?deliveryDays .
    %OfferXYZ% bsbm-export:validuntil ?validTo }
WHERE {
    %OfferXYZ% bsbm:product ?productURI .
    ?productURI rdfs:label ?productlabel .
    %OfferXYZ% bsbm:vendor ?vendorURI .
    ?vendorURI rdfs:label ?vendorname .
    ?vendorURI foaf:homepage ?vendorhomepage .
    %OfferXYZ% bsbm:offerWebpage ?offerURL .
    %OfferXYZ% bsbm:price ?price .
    %OfferXYZ% bsbm:deliveryDays ?deliveryDays .
    %OfferXYZ% bsbm:validTo ?validTo }
```

Table 4 gives an overview of the characteristics of the BSBM benchmark queries and highlights specific SPARQL features that are used by the queries. As the queries are motivated by the use case of an e-commerce portal, various queries use LIMIT modifiers in order to restrict the number of query results. Query 3 requires negation. As the SPARQL standard does not directly provide for negation, the query uses a combination of an OPTIONAL pattern and a FILTER clause that tests whether the optional variable is unbound to express negation. Query 6 encodes a free text search. As the SPARQL standard does not support free text search and as the BSBM benchmark strictly follows the standard without making use of proprietary extension functions, query 6 uses the SPARQL regex() function. This function is likely to be much slower than proprietary SPARQL extension functions for free text search that are usually backed by a full text index. We hope that negation and free text search will be added to a future version of SPARQL and will then change the queries accordingly.

Table 4. Characteristics of the BSBM benchmark queries.

Characteristic	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
Simple filters	√		√	√			√	√	√	√		
Complex filters					√	√						
More than 9 patterns		√		√			√	√				
Unbound predicates											√	
Negation			√									
OPTIONAL operator		√	√				√	√				
LIMIT modifier	√		√	√	√			√		√		
ORDER BY modifier	√		√	√	√			√		√		
DISTINCT modifier	√				√					√		
REGEX operator						√						
UNION operator				√							√	
DESCRIBE operator									√			
CONSTRUCT operator												√

The benchmark queries do not test all features of the SPARQL query language as various features were not required by the use case. SPARQL and RDF(S) features that are not benchmarked include querying RDF datasets and named graphs, blank nodes, collections and containers, property hierarchies, reified triples, the REDUCED modifier, and the ASK query form.

SQL Representation

Table 5 contains the SQL representation of four benchmark queries. The complete SQL representation of the query mix is given in Section 3.4 of the BSBM specification (Bizer & Schultz, 2008a). It is nevertheless important to note that there are no exact counterparts for several SPARQL features in standard SQL. SPARQL features without exact counterparts are: 1. The regex() function in Query 6 which is emulated using the SQL LIKE operator in order to stay in the bounds of standard SQL. 2. The DESCRIBE operator used in Query 9 and emulated in SQL with a SELECT clause that lists attributes that are likely to be returned by a store. Thus, benchmark results obtained using the SQL query mix should only be used for general orientation.

Table 5. SQL representation of selected BSBM queries.

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

```
SELECT DISTINCT nr, label
FROM product p, producttypeproduct ptp
WHERE p.nr = ptp.product AND ptp.productType=@ProductType@
   AND propertyNum1 > @x@
   AND p.nr IN (SELECT distinct product FROM productfeatureproduct WHERE
                    productFeature=@ProductFeature1@)
   AND p.nr IN (SELECT distinct product FROM productfeatureproduct WHERE
                    productFeature=@ProductFeature2@)
ORDER BY label
LIMIT 10;
```

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

```
SELECT pt.label, pt.comment, pt.producer, productFeature, propertyTex1, propertyTex2,
            propertyTex3, propertyNum1, propertyNum2, propertyTex4, propertyTex5,
            propertyNum4
FROM product pt, producer pr, productfeatureproduct pfp
WHERE pt.nr=@ProductXYZ@ AND pt.nr=pfp.product AND pt.producer=pr.nr;
```

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

```
SELECT nr, label
FROM product
WHERE label like "%@word1@%";
```

Query 9: Get information about a reviewer.

```
SELECT p.nr, p.name, p.mbox_sha1sum, p.country, r2.nr, r2.product, r2.title
FROM review r, person p, review r2
WHERE r.nr=@ReviewXYZ@ AND r.person=p.nr AND r2.person=p.nr;
```

4. Performance Metrics

BSBM benchmark experiments should report the following performance metrics:

1. **Query Mixes per Hour (QMpH):** The central performance metric of the BSBM benchmark are query mixes per hour. The metric measures the number of complete BSBM query mixes that are answered by a SUT within one hour. QMpH numbers should always be reported together with the size of the dataset against which the queries were run, and the numbers of clients that concurrently worked against the SUT.
2. **Queries per Second (QpS):** In order to allow a more differentiated analysis, benchmark results should also be reported on a per query type basis. The QpS metric measures the number of queries of a specific type that were answered by the SUT within a second. The metric is calculated by dividing the number of queries of a specific type within a benchmark run by the cumulated execution time of these queries. The metric must be measured by running complete BSBM query mixes against the SUT and may not be measured by running only queries of the specific type. QpS numbers should always be reported together with the size of the dataset against which the queries were run, and the numbers of clients that concurrently worked against the SUT.
3. **Load time (LT):** Cumulative time to load an RDF or relational benchmark dataset from the source file into the SUT. This includes any time spend by the SUT to build initial index structures and generate statistics about the dataset for query optimization. LT numbers should always be reported together with the size of the dataset and the representation type (i.e. Turtle or SQL dump).

5. 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 (Gray, 1993; Yuanbo Guo et al, 2007). As Semantic Web technologies are used within a wide range of application scenarios, a variety of different 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) (Guo et al., 2005). In addition to the experiment in the original paper, (Rohloff et al., 2007) presents the results of benchmarking DAML DB, SwiftOWLIM, BigOWLIM and AllegroGraph using a LUMB(8000) dataset consisting of roughly one billion triples. The LUBM benchmark has been extended in (Ma et al., 2006) 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. Both benchmarks do not employ benchmarking techniques such as system warm-up, simulating concurrent clients, and executing mixes of parameterized queries in order to test the caching strategy of a SUT.

An early SPARQL-specific performance benchmark is the DBpedia Benchmark (Becker, 2008). The benchmark measures the execution time of 5 queries that are relevant in the context of DBpedia Mobile (Becker & Bizer, 2008) against parts of the DBpedia dataset. Compared to the BSBM benchmark, the DBpedia Benchmark has the drawbacks that its dataset cannot be scaled to different sizes and that the queries only test a relatively narrow set of SPARQL features.

A recent SPARQL benchmark is SP^{2}Bench (Schmidt, et al., 2008a and 2008b). 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. The SP^{2}Bench benchmark queries are not parameterized and are not ordered within a use case motivated sequence. As the primary interest of the authors is the "basic performance of the approaches (rather than caching or learning strategies of the systems)" (Schmidt, et al., 2008a), they decided for cold runs instead of executing queries against warmed-up systems. Because of these differences, the SP^{2}Bench benchmark is likely to be more useful to RDF store developers that want to test "the generality of RDF storage schemes" (Schmidt, et al., 2008a), while the BSBM benchmark aims to support application developers in choosing systems that are suitable for mixed query workloads.

A first benchmark for comparing the performance of relational database to RDF mapping tools with the performance of native RDF stores is presented in (Svihala & Jelinek, 2007). The benchmark focuses on the production of RDF graphs from relational databases and thus only tests SPARQL CONSTRUCT queries. In contrast, the BSBM query mix also contains various SELECT queries.

A benchmarking methodology for measuring the performance of Ontology Management APIs is presented in (García-Castro & Gómez-Pérez, 2005). Like BSBM, this methodology also employs parameterized queries and requires systems to be warmed up before their performance is measured.

Ongoing initiatives in the area of benchmarking Semantic Web technologies are the Ontology Alignment Evaluation Initiative (Caracciolo, et al, 2008) which compares ontology matching systems, and the Billion Triple track of the Semantic Web Challenge[4] which evaluates the ability of Semantic Web applications to process large quantities of RDF data that is represented using different schemata and has partly been crawled from the public Web. Further information about RDF benchmarks and current benchmark results are found on the ESW *RDF Store Benchmarking* wiki page[5].

[4] http://challenge.semanticweb.org/

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

5. Benchmark Experiment

As a proof of concept, we ran the Berlin SPARQL Benchmark against four popular RDF stores (Sesame[6], Virtuoso[7], Jena TDB[8], and Jena SDB[9]) and two SPARQL-to-SQL rewriters (D2R Server[10] and Virtuoso RDF Views[11]) for three dataset sizes: One million triples, 25 million triples, and 100 million triples. After describing the setup and the methodology of the experiment, this section presents the benchmark results for a single client as well as for up to 64 clients working concurrently against the SUTs. In order to set the benchmark results into context, we compare them with the performance of two RDBMS (MySQL and Virtuoso RDBMS).

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 (kernel version 2.6.24-23). All databases were placed on the 10,000 rpm hard disk. Java version 1.6.0_07 was used and all Java stores were run under the Java HotSpot(TM) 64-Bit Server VM (build 10.0-b23).

Systems under Test and their Configuration

The systems under test store RDF data either in underlying relational databases or rely on native RDF storage engines. Sesame allows the user to choose between three storage engines (in-memory, native, DBMS-backend). Jena SDB offers three different RDF storage layouts for the underlying RDBMS (layout2, layout2/index und layout2/hash). Virtuoso RDF Views is coupled with the Virtuoso RDBMS, while D2R Server can work on top of MySQL, PostgreSQL, Oracle and other SQL-92 compatible databases. The systems employ cost-based query planning (Erling & Mikhailov, 2007; Owens et al., 2009). The dataset statistics that are used for evaluating the cost of different query execution plans are either generated once after the dataset is loaded (Jena TDB, Jena SDB, D2R Server) or are created on the fly by sampling data (Virtuoso TS). The systems dynamically cache parts of the dataset, indices as well as (intermediate) query results in main memory.

The impact of the storage layout, query plan optimization and caching on the overall query performance highly depends on the concrete configuration of the system as well as on the number and types of queries that contributed to filling the caches. In order to be able to report meaningful benchmark results we therefore optimized the configuration of the systems in cooperation with the developers of the systems and

[6] http://www.openrdf.org/about.jsp
[7] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/
[8] http://jena.hpl.hp.com/wiki/TDB
[9] http://jena.hpl.hp.com/wiki/SDB
[10] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/
[11] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSQL2RDF

warmed up the caches of the systems by executing query mixes until the average runtime per query mix stabilized (see Methodology of the Experiment below).

In the following, we provide the version numbers of the SUTs and give an overview about their configuration. The exact configuration of each system including all settings that were changed from defaults and all indices that were set is given in (Bizer & Schultz, 2008b).

1. **Sesame** Version 2.2.4 with Tomcat Version 5.5.25.5 as HTTP interface. We used the *native storage* layout, set the *spoc, posc, opsc* indices in the native storage configuration, and increased Java heap size to 6144MB.
2. **Jena TDB** Version 0.7.2 and Joseki Version 3.2 (CVS 2009-02-15) as HTTP interface. The TDB optimizer was configured to use the statistics based optimization strategy.
3. **Jena SDB** Version 1.2.0 and Joseki Version 3.2 (CVS 2009-02-15) as HTTP interface and MySQL Version 5.1.26 as underlying RDBMS. We configured SDB to use layout2/hash. The MySQL configuration is given below at 7.
4. **Virtuoso Triple Store** Open-Source Edition v5.0.10, abbreviated later in this article as *Virtuoso TS*. We changed the following performance related parameters: NumberOfBuffers = 520000; MaxCheckpointRemap = 1000000; StopCompilerWhenXOverRunTime = 1.
5. **Virtuoso RDF Views** with Virtuoso Open-Source Edition v5.0.10 as underlying RDBMS. Abbreviated later in this article as *Virtuoso RV*. The configuration parameters were the same as for Virtuoso TS. We used a RDBMS-to-RDF mapping provided by the Openlink developers and set 24 indices according to their suggestion. The complete mapping is given in (Bizer & Schultz, 2008b).
6. **D2R Server** Version 0.6 with MySQL Version 5.1.26 as underlying RDBMS. We increased Java heap size to 6144MB and configured MySQL as described within point 7. The complete D2RQ mapping is given in (Bizer & Schultz, 2008b).
7. **MySQL** Version 5.1.26. We set the key_buffer size to 5600M, set indices on every foreign key column as well as on producttypeproduct (productType, product), review(product,person), offer(product, deliveryDays, validTo), and productfeatureproduct(productFeature, product). The *analyze table* command was executed for all tables in the database.
8. **Virtuoso RDBMS** Open-Source Edition v5.0.10, abbreviated later in this article as *Virtuoso SQL*. The configuration was the same as for Virtuoso TS and we set the 24 indices that were proposed by the OpenLink developers.

Methodology of the Experiment

Before we started to measure the performance of the systems, we ran a qualification test against all systems in order to check whether they return correct results for the BSBM queries. Within this test, the one million triple BSBM dataset was loaded into the stores and 15 query mixes (altogether 375 queries, fixed randomizer seed) were run against the stores. The query results returned by the stores were compared with the expected results using the BSBM qualification tool. For the

DESCRIBE query (Query 9), the qualification tool only checked whether the result contained any RDF triples as the results of DESCRIBE queries may vary from store to store. All SUTs passed the BSBM qualification test.

We then applied the following test procedure to each store for each dataset size:

1. **Load the benchmark dataset.** The load performance of the systems was measured by loading the Turtle representation of the BSBM datasets into the triple stores and by loading the relational representation in the form of SQL dumps into MySQL and the Virtuoso RDBMS. The loaded datasets were forward chained and contained all *rdf:type* statements for product types. Thus the systems did not have to do any inferencing.
2. **Shutdown store, clear caches, restart store.** After the dataset is loaded and all indices are build, the store and all associated software components were shut down. The operating system caches were freed and the store was restarted.
3. **Execute ramp-up until steady-state is reached.** In order to benchmark the systems under normal working conditions, we warmed them up by executing query mixes until the average runtime per query mix stabilized. We determined this steady-state by using the results of the last 50 query mixes as evaluation window. For this evaluation window, the average execution time of the first 25 query mixes aqt_{1-25} was calculated. Afterwards, we totaled the positive and negative aberration ta_{26-50} of the execution times of the last 25 query mixes from aqt_{1-25}. We consider a SUT in steady-state when ta_{16-30} deviates less than 3% from aqt_{1-15} and continued to run query mixes until this condition was met.
4. **Execute single-client test run.** The query performance of the systems for a single client was measured by running 500 BSBM query mixes (altogether 12500 queries) against the systems over the SPARQL protocol.
5. **Execute multiple-client test runs.** Afterwards, we executed measurement runs with 2 clients, 4 clients, 8 clients, and 64 clients concurrently working against the SUT. Each run consisted of 500 query mixes.
6. **Execute test run with reduced query mix.** Previous experiments showed that the SUTs spent between 25% and 95% of the overall runtime on only two out of the 25 queries: Query 5 which contains complex filter expressions for identifying similar products and Query 6 which uses a regex() function to emulate free text search. In order to allow the SUTs to optimize their caches for the less resource intensive queries, we excluded Query 5 and 6 from the query mix and repeat steps 2 to 5 with this reduced query mix.

The different test runs use different randomizer seeds to choose query parameters. This ensures that the test driver produces distinctly parameterized queries over all runs and makes it more complicated for the SUTs to apply caching techniques. The test driver and the SUT were running on the same machine in order to reduce the influence of network latency.

In order to make it possible for interested parties to reproduce the results of the experiment, we provide the RDF and the relational representation of the benchmark datasets as well as the qualification tool and the test driver for download on the BSBM website.

6. Results of the Experiment

In the following we present and discuss the results of the experiment.

Load Times

Table 6 summarizes the time it took to load the Turtle files and the SQL dumps into the different stores. Loading the SQL dump was significantly faster than loading the Turtle representation. Comparing the load performance of the triple stores, it turns out that Virtuoso is fast for small datasets while Jena TDB is fast for large datasets. Sesame Native and Jena SDB altogether showed a poor load performance which culminated in a load time of more than 3 days for loading the 100M dataset into Sesame.

Table 6. Load times for different stores and dataset sizes (in [day:]hh:min:sec).

	1M	**25M**	**100M**
Sesame	00:02:59	12:17:05	3:06:27:35
Jena TDB	00:00:49	00:16:53	01:34:14
Jena SDB	00:02:09	04:04:38	1:14:53:08
Virtuoso TS	00:00:23	00:39:24	07:56:47
Virtuoso RV	00:00:34	00:17:15	01:03:53
D2R Server	00:00:06	00:02:03	00:11:45
MySQL	00:00:06	00:02:03	00:11:45
Virtuoso SQL	00:00:34	00: 17:15	01:03:53

The disk space footprint of the loaded and index datasets varied widely between the triple stores and the RDBMS. The 100M triple dataset consumed 17 GB disk space after being loaded into Sesame, Jena SDB and Virtuoso TS. Loading the dataset into Jena TDB consumed 19 GB disk space. In contrast, the relational representation of the dataset consumed only 5.6 GB after being loaded into MySQL and 8.4 GB in the Virtuoso RDBMS.

Ramp-Up

The runtime of the first warm-up query mix that was executed against the newly started stores proved to be significantly longer than the runtimes of all other query mixes as the stores were slowed down by connecting to underlying RDBMS or building index structures in memory. In order to give an overview about the effect of system warm-up, Table 7 list the speed-up factors between the runtime of the second warm-up query mix and the average runtime of a query mix in steady-state.

Table 7. Speed-up factors between the runtime of the second query mix and the average runtime of a query mix in steady-state.

	1M	**25M**	**100M**
Sesame	15,61	3,98	0,75
Jena TDB	3,03	0,52	0,00
Jena SDB	0,97	2,68	0,64
Virtuoso TS	0,47	26,14	46,65
Virtuoso RV	0,15	1,98	100,09
D2R Server	0,67	0,03	0,04
MySQL	26,30	17,37	8,49
Virtuoso SQL	1,03	13,58	247,20

Results for Single Clients

This section presents the results of the single client experiments. The complete run logs of the experiment can be found in (Bizer & Schultz, 2008b).

Overall Performance

Table 8 gives an overview of the overall query performance of the SUTs. The performance was measured by running 500 query mixes against the stores and is reported in query mixes per hour (QMpH). The best performance figure for each dataset size is set bold in the tables. The results of running the SQL representation of the query mix against MySQL and Virtuoso RDBMS are also included in the tables for comparison but are not considered for determining the best SPARQL performance figure.

Comparing the four triple stores with each other, the Virtuoso triple store shows the best overall performance for the 25M and the 100M datasets. For the 1M dataset, Sesame outperforms Virtuoso TS. In the category of SPARQL-to-SQL rewriters, Virtuoso RV clearly outperforms D2R Server.

Table 8. Overall performance: Complete query mix (in QMpH).

	1M	**25M**	**100M**
Sesame	**18,094**	1,343	254
Jena TDB	4,450	353	81
Jena SDB	10,421	968	211
Virtuoso TS	12,360	4,123	954
Virtuoso RV	17,424	**12,972**	**4,407**
D2R Server	2,828	140	35
MySQL	*235,066*	*18,578*	*4,991*
Virtuoso SQL	*192,013*	*69,585*	*9,102*

The comparison of the fastest triple store with the fastest SPARQL-to-SQL rewriter shows that the performance of both architectures is similar for the 1M triple dataset. For bigger datasets, Virtuoso RV outperforms the triple stores by at least factor 3.

Setting the performance of triple stores and SPARQL-to-SQL rewriter in relation to the result from running the SQL query mix directly against the relational databases reveals an unedifying picture: MySQL outperforms Sesame for the 1M dataset by the factor 13. For the 100M dataset, the Virtuoso triple store is outperformed by Virtuoso SQL by the factor 9.5. In relation to Virtuoso SQL, the fastest SPARQL-to-SQL rewriter (Virtuoso RV) is outperformed by the factor 2.0 (100M dataset).

Examining the times spent on individual queries (see section below), it appears for the 25M and the 100M datasets, that the stores spent up to 95% of the overall runtime on only two out of the 25 queries: Query 5 which contains complex filter expressions for identifying similar products and Query 6 which uses the regex() function to emulate free text search. The time spend on Query 6 also explains the good performance of Virtuoso SQL and Virtuoso RV in relation to MySQL. Virtuoso SQL and Virtuoso RV both recognize that the SPARQL regex() and the SQL LIKE expression in Query 6 are workarounds for expressing free text search and thus use a full text index on the field product.label to evaluate the query. MySQL does not use full text indices to evaluate SQL LIKE expressions and therefore shows a significantly slower overall performance in relation to Virtuoso SQL for the 25M and 100M datasets.

Queries 5 and 6 distort the significance of the overall figures for assessing the performance of the stores on basic tasks such as query pattern matching and simple filtering. We therefore excluded query 5 and 6 from the query mix and rerun the experiment with this reduced mix. The overall performance figures obtained by this second run are given in Table 9.

Table 9. Overall performance: Reduced query mix without queries 5 and 6 (in QMpH).

	1M	**25M**	**100M**
Sesame	**38,727**	**39,059**	3,116
Jena TDB	15,842	1,856	459
Jena SDB	15,692	4,877	584
Virtuoso TS	13,759	10,718	2,166
Virtuoso RV	18,516	17,529	**6,293**
D2R Server	11,520	3,780	1,261
MySQL	*516,271*	*280,993*	*84,797*
Virtuoso SQL	*219,616*	*195,647*	*14,400*

For the 100M dataset, excluding the two queries leads to a factor 11 speed-up for Sesame, a factor 4.7 speed-up for Jena TDB, factor 1.7 for Jena SDB, and a factor 1.2 speed-up for Virtuoso TS. For the SPARQL-to-SQL rewriters, excluding the two queries speeds up D2R Server by the factor 35, while Virtuoso RV, which uses a full text index to evaluate query 6, only gains 43%.

Comparing the performance of the three triple stores without queries 5 and 6 Sesame proofs to be the fastest store for all dataset sizes. In the category of SPARQL-to-SQL rewriters, Virtuoso RV again outperforms D2R Server. Comparing the performance of triple stores and SPARQL-to-SQL rewriters across architectures, Sesame outperforms Virtuoso RV for smaller datasets, while Virtuoso RV is twice as fast as Sesame for the 100M dataset. In the RDBMS category, Virtuoso SQL is outperformed by MySQL as it lost the advantage of using the full text index to evaluate Query 6.

Performance by Individual Query

In order to give a more detailed picture of the strengths and weaknesses of the different systems, we present the benchmark results for each individual query in Table 10. The results were measured by running the complete BSBM query mix. For the query-by-query results obtained by running the reduced query mix please refer to (Bizer & Schultz, 2008b). The results in the table are given as queries per second (QpS). The values are calculated based on average runtime of all queries of a specific type within the 500 query mixes. The best performance figure for each dataset size is again set bold.

Table 10. Performance by individual query (in QpS).

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

	1M	**25M**	**100M**
Sesame	**662**	200	15
Jena TDB	494	165	35
Jena SDB	374	198	12
Virtuoso TS	202	192	**132**
Virtuoso RV	199	173	122
D2R Server	328	**236**	79
MySQL	*3,021*	*955*	*476*
Virtuoso SQL	*1,195*	*833*	*470*

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

	1M	**25M**	**100M**
Sesame	**251**	**168**	32
Jena TDB	61	51	38
Jena SDB	50	47	35
Virtuoso TS	47	46	39
Virtuoso RV	78	75	**64**
D2R Server	41	36	40
MySQL	*4,525*	*3,333*	*3,268*
Virtuoso SQL	*1,592*	*1,456*	*991*

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

	1M	25M	100M
Sesame	**505**	140	13
Jena TDB	451	141	28
Jena SDB	283	151	8
Virtuoso TS	176	165	**136**
Virtuoso RV	182	**167**	129
D2R Server	226	115	56
MySQL	*2,833*	*919*	*459*
Virtuoso SQL	*1,079*	*838*	*456*

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

	1M	25M	100M
Sesame	**452**	128	10
Jena TDB	429	116	25
Jena SDB	240	132	7
Virtuoso TS	92	86	54
Virtuoso RV	106	96	**84**
D2R Server	224	**167**	72
MySQL	*2,653*	*919*	*428*
Virtuoso SQL	*1,098*	*759*	*443*

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

	1M	25M	100M
Sesame	30	1.69	0.52
Jena TDB	1.80	0.13	0.04
Jena SDB	18	1.05	0.46
Virtuoso TS	76	14	5.86
Virtuoso RV	**118**	**30**	**14**
D2R Server	1.08	0.04	0.01
MySQL	*396*	*25*	*8*
Virtuoso SQL	*410*	*43*	*12*

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

	1M	25M	100M
Sesame	14	0.53	0.13
Jena TDB	59	2.40	0.09
Jena SDB	16	0.55	0.12
Virtuoso TS	55	2.12	0.50
Virtuoso RV	**275**	**25**	**6.03**
D2R Server	26	1	0.23

MySQL	*163*	*7*	*1*
Virtuoso SQL	*1,605*	*97*	*21*

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

	1M	**25M**	**100M**
Sesame	87	57	2
Jena TDB	**189**	28	6
Jena SDB	112	27	2
Virtuoso TS	72	36	5
Virtuoso RV	81	76	**15**
D2R Server	123	**97**	12
MySQL	*1,912*	*1,370*	*407*
Virtuoso SQL	*831*	*733*	*26*

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

	1M	**25M**	**100M**
Sesame	**297**	90	4
Jena TDB	159	27	8
Jena SDB	134	30	3
Virtuoso TS	116	113	12
Virtuoso RV	132	**129**	**22**
D2R Server	72	62	12
MySQL	*3,497*	*601*	*63*
Virtuoso SQL	*1,715*	*1,603*	*31*

Query 9: Get information about a reviewer.

	1M	**25M**	**100M**
Sesame	**924**	128	19
Jena TDB	57	3	1
Jena SDB	129	9	2
Virtuoso TS	541	**533**	53
Virtuoso RV	506	482	**164**
D2R Server	81	73	33
MySQL	*4,255*	*2,849*	*1,370*
Virtuoso SQL	*2,639*	*2,639*	*145*

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

	1M	25M	100M
Sesame	**429**	93	2
Jena TDB	**429**	62	19
Jena SDB	289	40	2
Virtuoso TS	95	75	8
Virtuoso RV	224	**220**	67
D2R Server	218	200	**77**
MySQL	*4,444*	*3,356*	*1,883*
Virtuoso SQL	*2,004*	*1,587*	*267*

Query 11: Get all information about an offer.

	1M	25M	100M
Sesame	**652**	98	13
Jena TDB	376	45	24
Jena SDB	351	97	23
Virtuoso TS	361	**342**	**44**
Virtuoso RV	102	100	41
D2R Server	33	2	0.4
MySQL	*9,174*	*4,367*	*456*
Virtuoso SQL	*2,494*	*3,195*	*1,248*

Query 12: Export information about an offer into another schema.

	1M	25M	100M
Sesame	**797**	**350**	18
Jena TDB	53	3	1
Jena SDB	119	9	2
Virtuoso TS	133	129	39
Virtuoso RV	151	148	91
D2R Server	203	162	**170**
MySQL	*7,246*	*2,571*	*539*
Virtuoso SQL	*2,801*	*2,985*	*1,524*

The picture drawn by the overall runtimes is partly changed when the runtimes of specific queries are analyzed. In the following, we will interpret the results on a query-by-query basis.

Query 1 is very simple and specific and all stores show a good performance on this query. Query 2 consists of 15 triple patterns out of which 3 are optional. For the 1M and 25M datasets, Sesame clearly outperforms all other stores on this query. For the 100M dataset, both SPARQL-to-SQL rewriters show a better performance than the RDF stores. As the query can be evaluated against the relational representation using two simple join, the RDBMS outperform the RDF stores by factors between 15 and 100. Queries 3 and 4 are unproblematic for all stores, with again, Sesame being the fastest store for the 1M dataset and Virtuoso RV and TS performing well for the larger data sets. Jena SDB shows a steep performance slump between the 25M and the 100M dataset for both queries.

Query 5 contains the most demanding FILTER expressions of all queries in the mix. Evaluating this query is time consuming for all RDF stores as well as for the RDBMS. Not taking into account the RDBMS, Virtuoso RV shows the best performance on this query for all dataset sizes and largely outperforms D2R Server which is very slow on this query. The time spent on Query 5 dominates the overall runtime of the complete query mix for several stores: D2R Server spends 90% of the overall runtime on this query, Jena TDB 70%, Jena SDB and Sesame around 20%, the Virtuoso triple store only 6%. The performance on Query 6 is mostly determined by whether a store relies on a full text index or not to answer the query. Stores without text index, like Jena TDB, need up to 11 seconds to evaluate the query against the 100M dataset. Virtuoso RV and Virtuoso SQL recognize that the SPARQL regex() and the SQL LIKE expressions in Query 6 are workarounds for expressing free text search and thus clearly outperform the other stores by using a full text index to answer the query. Query 7 and 8 again contain a large number of triple patters out of which several are optional. They thus proof to be expansive for all stores. For both queries, triple stores (Sesame and Jena TDB) perform best on the 1M dataset, while the SPARQL-to-SQL rewriters show the best performance for the large datasets. The Query 9 uses the DESCRIBE operator. Jena SDB and TDB both have problems with this operator while the other stores show a good performance.

Query 10 can be evaluated against the relation representation using a single join between the vendor and the offer tables. The SPARQL-to-SQL rewriters thus clearly outperform the triple stores for larger datasets. For the 1M dataset Sesame and Jena TDB show the best performance. Query 11 contains two triple patterns with unbound predicates. As unbound predicates require SPARQL-to-SQL rewriters to examine various columns in the relational database, the rewriters are outperformed on this query by the triple stores. Query 12 uses the CONSTRUCT operator. Sesame performs best on this query for the 1M and 25M dataset, while D2R Server shows the best performance for the 100M dataset. The CONSTRUCT operator is problematic for Jena TDB and SDB which are both very slow for the 25M and 100M datasets.

Results for Multiple Clients

In real-world settings there are often multiple clients working against a SPARQL endpoint. We thus had the test driver simulate up to 64 clients concurrently working against the SUTs. Tables 11 and 12 summarize the results of the multi-client runs against the 1M and 25M datasets. The performance of the systems is measured by the number of query mixes that were answered by the SUT for all clients within one hour. Note that the query mixes per hour values are extrapolated from the time it took the clients to execute 500 query mixes each.

Table 11. Performance for multiple clients, 1M dataset (in QMpH).

Dataset size 1M	Number of clients				
	1	**2**	**4**	**8**	**64**
Sesame	**18,094**	19,057	16,460	18,295	16,517
Jena TDB	4,450	6,752	9,429	8,453	8,664
Jena SDB	10,421	17,280	23,433	24,959	23,478
Virtuoso TS	12,360	21,356	32,513	29,448	29,483
Virtuoso RV	17,424	**28,985**	**34,836**	**32,668**	**33,339**
D2R Server	2,828	3,861	3,140	2,960	2,938
MySQL	*235,066*	*318,071*	*472,502*	*442,282*	*454,563*
Virtuoso SQL	*192,013*	*199,205*	*274,796*	*357,316*	*306,172*

Table 12. Performance for multiple clients, 25M dataset (in QMpH).

Dataset size 25M	Number of clients				
	1	**2**	**4**	**8**	**64**
Sesame	1,343	1,485	1,204	1,300	1,271
Jena TDB	353	513	694	536	555
Jena SDB	968	1,346	1,021	883	927
Virtuoso TS	4,123	7,610	9,491	5,901	5,400
Virtuoso RV	**12,972**	**22,552**	**30,387**	**28,261**	**28,748**
D2R Server	140	187	160	146	143
MySQL	*18,578*	*31,093*	*39,647*	*40,599*	*40,470*
Virtuoso SQL	*69,585*	*85,146*	*135,097*	*173,665*	*148,813*

The experiment showed for all systems except Sesame and D2R Server that the number of query mixes per hour doubles on average between the single client and the 4 client runs. Then the performance more or less stabilizes in a 20% variation corridor for the 8 and 64 client runs. For Sesame and D2R Server, the number of query mixes per hour stays mostly constant, which indicates that the systems do not take advantage of the multi-core processor provided by the benchmark machine.

The absolute numbers show that Virtuoso TS outperforms Jena TDB, Jena SDB and Sesame in all multi-client tests. Virtuoso RV also clearly outperforms D2R Server. Virtuoso RV's ability to handle multiple clients combined with the generally good performance of SPARQL-to-SQL rewriting architecture for larger datasets leads to a far-off overall win of Virtuoso RV against all other stores for the 25M dataset: For the 8 client case Virtuoso RV is factor 3.7 faster than Virtuoso TS, factor 20 faster than Sesame, and factor 31 faster than Jena SDB.

7. Conclusion

This article introduced the Berlin SPARQL Benchmark for comparing the query performance of native RDF stores with the performance of SPARQL-to-SQL rewriters across architectures. The benchmark dataset, the benchmark queries as well as the query sequence are grounded within an e-commerce use case. The benchmark focuses on measuring query performance under enterprise conditions. The BSBM benchmark thus employs benchmarking techniques from the database and transaction processing field (Gray, 1993; TPC, 2008), such as executing query mixes, query parameterization, simulation of multiple clients, and system ramp-up.

The BSBM benchmark complements the field of benchmarks for Semantic Web technologies with a benchmark for measuring SPARQL query performance for mixed query workloads as well as for comparing the performance of RDF stores and SPARQL-to-SQL rewriters. The benchmark fills a gap between SP2Bench which is designed for the comparison of different RDF store layouts, LUBM which focuses on reasoning, and the Ontology Alignment Evaluation Initiative which focuses on schema matching and mapping.

As a proof of concept, the article presented the results of a benchmark experiment in which the BSBM benchmark is used to compare the performance of four RDF stores with two SPARQL-to-SQL rewriters and to set the results into relation to the performance of RDBMS. The experiment unveiled several interesting findings:

Within all categories, none of the benchmarked systems was superior within the single client use case for all queries and all dataset sizes. Comparing the RDF stores, Sesame showed a good performance for small datasets while Virtuoso TS was faster for larger datasets. For larger datasets, Jena TDB and SDB could not compete in terms of overall performance. In the category of SPARQL-to-SQL rewriters Virtuoso RV clearly outperformed D2R Server.

Comparing the fastest RDF store with the fastest SPARQL-to-SQL rewriter shows that the rewriting approach outperforms native RDF storage with increasing dataset size. This is shown by the overall runtimes as well as by the results for 9 out of 12 individual queries (100M triple, single client).

Setting the results of the RDF stores and the SPARQL-to-SQL rewriters in relation to the performance of classical RDBMS unveiled an unedifying picture. Comparing the overall performance (100M triple, single client, all queries) of the fastest rewriter with the fastest relational database shows an overhead for query rewriting of 106%. This is an indicator that there is still room for improving the rewriting algorithms. Comparing the overall performance (100M triple, single client, all queries) of the fastest RDF store with the fastest RDBMS shows that the RDF store is outperformed by the factor 8.5. There are two potential explanations for this finding: First, as SPARQL is still a very new query language it is likely that the RDF stores have not yet implemented similarly sophisticated optimization techniques as SQL query engines which are under development for decades. Thus, there should be potential for RDF stores to catch up in the future. The second reason is more fundamental and lies in the combination of the RDF data model and the structure of benchmark dataset. The RDF data model has been designed for the open Web use case and thus provides for representing semi-structured data that mixes different schemata. As the BSBM benchmark dataset is relatively homogeneous, the flexibility of the RDF data model turns out to be a disadvantage compared to the relational model which is designed for clearly structured data.

The complete BSBM benchmark specification, current benchmark results, the 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 for making this work possible with a research grant. We also would like to thank Orri Erling (OpenLink Software), Arjohn Kampman (Aduna), Andy Seaborne (Hewlett Packard), Michael Schmidt (Freiburg University), Richard Cyganiak (DERI Galway), Patrick van Kleef (OpenLink Software), and Ivan Mikhailov (OpenLink Software) for their feedback on the benchmark design and their help with configuring the stores for the experiment.

References

Becker, C. (2008): RDF Store Benchmarks with DBpedia comparing Virtuoso, SDB and Sesame. Retrieved March 2, 2009, http://www4.wiwiss.fu-berlin.de/benchmarks-200801/

Becker, C., Bizer, C. (2008): DBpedia Mobile: A Location-Enabled Linked Data Browser. In: Proceedings of the 1st Workshop about Linked Data on the Web (LDOW2008).

Bizer, C., Schultz, A. (2008a): Berlin SPARQL Benchmark (BSBM) Specification - V2.0. Retrieved March 2, 2009, http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/

Bizer, C., Schultz, A. (2008b): Berlin SPARQL Benchmark Results. Retrieved March 2, 2009, http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/

Caracciolo, C., et al (2008): Results of the Ontology Alignment Evaluation Initiative 2008. In: Proceedings of the 3rd International Workshop on Ontology Matching (OM-2008), CEUR-WS Vol-431.

Erling, O., Mikhailov, I.: RDF Support in the Virtuoso DBMS. In: Proceedings of the 1st Conference on Social Semantic Web (CSSW), pp. 59-68.

García-Castro, R., Gómez-Pérez, A.: Guidelines for Benchmarking the Performance of Ontology Management APIs. In: The Semantic Web – ISWC 2005, LNCS vol. 3729/2005, pp 277-292

Gray, J. (1993): The Benchmark Handbook for Database and Transaction Systems, (2nd Edition), Morgan Kaufmann, ISBN 1-55860-292-5.

Guo, Y., Pan, Z., Heflin J. (2005): LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics, issue 3(2), pp 158-182.

Kendall, G. C., Feigenbaum, L., Torres, E. (2008): SPARQL Protocol for RDF. W3C Recommendation. Retrieved March 2, 2009, http://www.w3.org/TR/rdf-sparql-protocol/

Ma, L., et al. (2006): Towards a Complete OWL Ontology Benchmark (UOBM). In: The Semantic Web: Research and Applications, LNCS vol. 4011/2006, pp 125-139.

Owens, A., et al. (2009): Clustered TDB: A Clustered Triple Store for Jena. Retrieved March 2, 2009, http://eprints.ecs.soton.ac.uk/16974/

Prud'hommeaux, E., Seaborne, A. (2008): SPARQL Query Language for RDF. W3C Recommendation. Retrieved March 2, 2009, http://www.w3.org/TR/rdf-sparql-query/

Rohloff, K., et al. (2007): An Evaluation of Triple-Store Technologies for Large Data Stores. In: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, LNCS vol. 4806/2007, pp 1105-1114.

Sahoo, S., et al. (2009): A Survey of Current Approaches for Mapping of Relational Databases to RDF. Retrieved February 25, 2009, http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf

Schmidt, M., Hornung, T., Küchlin, N., Lausen, G., Pinkel, C. (2008a): An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In: Proceedings of the International Semantic Web Conference (ISWC 2008).

Schmidt, M., Hornung, T., Lausen, G., Pinkel, C. (2008b): SP2Bench: A SPARQL Performance Benchmark. Technical Report, arXiv:0806.4627V1 cs.DB.

Svihala, M., Jelinek, I. (2007): Benchmarking RDF Production Tools. In: Proceedings of the 18th International Conference on Database and Expert Systems Applications (DEXA 2007).

Transaction Processing Performance Council (2008): TPC Benchmark H, Standard Specification Revision 2.7.0. Retrieved March 2, 2009, http://www.tpc.org/tpch/spec/tpch2.7.0.pdf

Yuanbo, G., et al. (2007): A Requirements Driven Framework for Benchmarking Semantic Web Knowledge Base Systems. IEEE Transactions on Knowledge and Data Engineering, issue 19(2), pp 297-309.