Une classification expérimentale multi-critères des évaluateurs SPARQL répartis

Damien Graux, Louis Jachiet, Pierre Genevès & Nabil Layaida

Université Grenoble Alpes, Inria, LIG

Tyrex Team
<http://tyrex.inria.fr/>

BDA, Nancy, 14 Novembre 2017
General Context & Goal

Context

- Large amounts of available RDF data
- Several SPARQL evaluators
- Distributed context
General Context & Goal

Context
- Large amounts of available RDF data
- Several SPARQL evaluators
- Distributed context

Goal
How to choose an efficient SPARQL evaluator in a distributed context?
Section 1

Background
Resource Description Framework [HM04]

RDF standard has been designed to:

- Present a simple way of representing data
- Have a formal semantics to offer basis for reasoning
- Have an extensible vocabulary
- Provide, share and exchange datasets between applications.
- Allow anyone to make statements about any resource.
Resource Description Framework [HM04]

RDF standard has been designed to:

- Present a simple way of representing data
- Have a formal semantics to offer basis for reasoning
- Have an extensible vocabulary
- Provide, share and exchange datasets between applications.
- Allow anyone to make statements about any resource.

RDF essentials

- RDF is a w3c standard
- An RDF graph is a set of RDF triples
- An RDF triple has three components:
  - a subject (s)
  - a predicate (p)
  - a object (o)
SPARQL Protocol and RDF Query Language [G⁺13]

Syntax in a nutshell

- SPARQL is an RDF query language
- SPARQL is a w3c standard
- 4 types: SELECT ASK CONSTRUCT DESCRIBE
- SQL-like syntax: SELECT Var List WHERE { Pattern }

Pattern (P) := P . P
| (subject|var) (predicate|var) (object|var)
| {P} UNION {P}
| P OPTIONAL {P}
| FILTER Constraint
Section 2

SPARQL Evaluators
Jumble of Evaluators

4store
CouchBaseRDF
BitMat
YARS
Hexastore
CliqueSquare
RYA
Parliament
Virtuoso
RDF-3X
...
### Some Previous Surveys

<table>
<thead>
<tr>
<th>When?</th>
<th>Who?</th>
<th>What?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Barstow [Bar01]</td>
<td>Focuses on open-source solutions; and looks at some of their specificities</td>
</tr>
<tr>
<td>2002</td>
<td>Beckett [Bec02]</td>
<td>Updates</td>
</tr>
<tr>
<td>2003</td>
<td>Beckett [BG03]</td>
<td>Focuses on the use of relational database management systems to store RDF datasets</td>
</tr>
<tr>
<td>2004</td>
<td>Lee [Lee04]</td>
<td>Updates</td>
</tr>
<tr>
<td>2012</td>
<td>Faye [FCB12]</td>
<td>Lists the various RDF storage approaches mainly used by single-node systems</td>
</tr>
<tr>
<td>2015</td>
<td>Kaoudi [KM15]</td>
<td>Presents a survey focusing only on RDF in the clouds</td>
</tr>
</tbody>
</table>
RDF Storage Strategies [FCB12]
RDF Storage Strategies [FCB12]

native
- In-memory
  - BitMat
- On Disks
  - Standalone
    - Virtuoso
    - RDF-3X
  - Embedded

non-native
- Web APIs
- DBMS-based
  - Schema-Carefree
    - Triple Table
      - 3store
  - Schema-Aware
    - Vertical Partitioning
      - swStore
    - Property Table
Distributed Evaluation Methods [KM15]

- Distributed RDF Storage Methods
  - Federation
    - Horizontal Fragmentation
    - Graph Partitioning
  - Key-Value Stores
    - Triple-based
    - Graph-based
  - Independent
  - Distributed File System
    - Triple Table
    - Vertical Partitioning
    - Property Table
Distributed Evaluation Methods [KM15]

- Distributed RDF Storage Methods
  - Federation
    - Horizontal Fragmentation
    - Graph Partitioning
  - Key-Value Stores
    - Triple-based: RYA
    - Graph-based
  - Independent
    - 4store
    - CouchBase
    - RDF
  - Distributed File System
    - Triple Table: Pig, SPARQL
    - Vertical Partitioning: S2RDF
    - Property Table
Distributed SPARQL Evaluator State-of-the-art Summary

Observations

1. Multiple RDF storage strategies
2. Various methods to distribute data and to compute queries
Observations

1. Multiple RDF storage strategies
2. Various methods to distribute data and to compute queries

How to pick an efficient evaluator?
Distributed SPARQL Evaluator State-of-the-art Summary

Observations

1. Multiple RDF storage strategies
2. Various methods to distribute data and to compute queries

How to pick an efficient evaluator?
Experimental Evaluation!
Section 3

Running an Experimental Study
# Experimental Studies

<table>
<thead>
<tr>
<th>When?</th>
<th>Who?</th>
<th>What?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td>Magkanaraki [MKA⁺02]</td>
<td>Reviews solutions dealing with ontologies</td>
</tr>
<tr>
<td>2009</td>
<td>Stegmaier [SGD⁺09]</td>
<td>Reviews solutions according to several parameters such as their licenses, their architectures and compares them using a scalable test dataset</td>
</tr>
<tr>
<td>2013</td>
<td>Cudré-Mauroux [CMEF⁺13]</td>
<td>Realizes an empirical study of distributed SPARQL evaluators (native RDF stores and several NoSQL solutions they adapted)</td>
</tr>
</tbody>
</table>
## Popular Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>SPARQL Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM [GPH05]</td>
<td>BGP</td>
</tr>
<tr>
<td>WatDiv [AHÖD14]</td>
<td>BGP</td>
</tr>
<tr>
<td>SP²Bench [SHLP09]</td>
<td>BGP + FILTER UNION OPTIONAL + Solution Modifiers + ASK</td>
</tr>
<tr>
<td>BolowgnaB [DEW+11]</td>
<td>BGP + aggregator (e.g. COUNT)</td>
</tr>
<tr>
<td>BSBM [BS09]</td>
<td>BGP + FILTER UNION OPTIONAL + Solution Modifiers + Logical negation + CONSTRUCT</td>
</tr>
<tr>
<td>DBPSB [MLAN11]</td>
<td>Use actually posed queries against dbpedia</td>
</tr>
<tr>
<td>RBench [QÖ15]</td>
<td>Generate queries according to considered datasets</td>
</tr>
</tbody>
</table>
## Popular Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>SPARQL Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM [GPH05]</td>
<td>BGP</td>
</tr>
<tr>
<td>WatDiv [AHÖD14]</td>
<td>BGP</td>
</tr>
<tr>
<td>SP²Bench [SHLP09]</td>
<td>BGP + FILTER UNION OPTIONAL + Solution Modifiers + ASK</td>
</tr>
<tr>
<td>BolowgnaB [DEW⁺11]</td>
<td>BGP + aggregator (e.g. COUNT)</td>
</tr>
<tr>
<td>BSBM [BS09]</td>
<td>BGP + FILTER UNION OPTIONAL + Solution Modifiers + Logical negation + CONSTRUCT</td>
</tr>
<tr>
<td>DBPSB [MLAN11]</td>
<td>Use actually posed queries against dbpedia</td>
</tr>
<tr>
<td>RBench [QÖ15]</td>
<td>Generate queries according to considered datasets</td>
</tr>
</tbody>
</table>
Considered Benchmarks

LUBM
- deterministically generated datasets
- 14 predefined queries (Q1-Q14)

WatDiv
- deterministically generated datasets (richer than the LUBM one)
- 20 generated queries according to templates
Selected SPARQL Evaluators

Criteria

- OpenSource
- Popular
- Recent
- Distributed
- At least BGP SPARQL Fragment
## Selected SPARQL Evaluators

<table>
<thead>
<tr>
<th>10 Systems</th>
<th>Framework</th>
<th>Back-End</th>
<th>Layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>4store</td>
<td>—</td>
<td>Data Fragments</td>
<td>Indexes</td>
</tr>
<tr>
<td>CumulusRDF</td>
<td>Cassandra</td>
<td>Key-Value</td>
<td>Hash-Sorted Indexes</td>
</tr>
<tr>
<td>CouchBaseRDF</td>
<td>CouchBase</td>
<td>Buckets</td>
<td>3 views</td>
</tr>
<tr>
<td>RYA</td>
<td>Accumulo</td>
<td>Key-Value on HDFS</td>
<td>3 Sorted indexes</td>
</tr>
<tr>
<td>SPARQLGX</td>
<td>Spark</td>
<td>Files on HDFS</td>
<td>Vertical Partition</td>
</tr>
<tr>
<td>S2RDF</td>
<td>SparkSQL</td>
<td>Tables on HDFS</td>
<td>Ext. Vertical Partition</td>
</tr>
<tr>
<td>CliqueSquare</td>
<td>Hadoop</td>
<td>Files on HDFS</td>
<td>Indexes</td>
</tr>
<tr>
<td>PigSPARQL</td>
<td>PigLatin</td>
<td>Files on HDFS</td>
<td>N-Triples Files</td>
</tr>
<tr>
<td>RDFHive</td>
<td>Hive</td>
<td>Tables on HDFS</td>
<td>Three-column Table</td>
</tr>
<tr>
<td>SDE</td>
<td>Spark</td>
<td>Files on HDFS</td>
<td>N-Triples Files</td>
</tr>
</tbody>
</table>
Cluster Specifications

- Cluster composed of 10 Virtual Machines hosted on two servers.
- Each VM has:
  - dedicated 2 physical cores (thus 4 logical cores)
  - 17 GB of RAM
  - 6 TeraBytes (TB) of disk.
- The network allows two VMs to communicate at 125MB/s
Obtained Results

We learned:

1. Considering the same dataset, loading times are spread over several magnitude orders.
Obtained Results

With the following RDF datasets:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of Triples</th>
<th>Original File Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>WatDiv1k</td>
<td>109 million</td>
<td>15 GB</td>
</tr>
<tr>
<td>Lubm1k</td>
<td>134 million</td>
<td>23 GB</td>
</tr>
<tr>
<td>Lubm10k</td>
<td>1.38 billion</td>
<td>232 GB</td>
</tr>
</tbody>
</table>

Figure: Preprocessing Time.
Obtained Results

We learned:

1. Considering the same dataset, loading times are spread over several magnitude orders.
2. For the same query on the same dataset, elapsed times can differ very significantly.
Obtained Results

**Figure**: Query Response Time with Lubm1k (134 million triples).

```
Q1
SELECT ?X WHERE {
  ?X rdf:type ub:GraduateStudent .
  ?X ub:takesCourse GraduateCourse0
}

Q2
SELECT ?X ?Y ?Z WHERE {
  ?X rdf:type ub:GraduateStudent .
  ?Y rdf:type ub:University .
  ?Z rdf:type ub:Department .
  ?X ub:undergraduateDegreeFrom ?Y
}

Q3
SELECT ?X WHERE {
  ?X ub:publicationAuthor AssistantProfessor0
}
```
Obtained Results

We learned:

1. Considering the same dataset, loading times are spread over several magnitude orders.
2. For the same query on the same dataset, elapsed times can differ very significantly.
3. Even with large datasets, most queries are not harmful per se, i.e. queries that incur long running times with some implementations still remain in the “comfort zone” for other implementations.
Obtained Results

(a) 4store

(b) S2RDF

(c) RYA

(d) PigSPARQL

Figure: Obtained results with WatDiv1k.
Obtained Results

We learned:

1. Considering the same dataset, loading times are spread over several magnitude orders
2. For the same query on the same dataset, elapsed times can differ very significantly
3. Even with large datasets, most queries are not harmful *per se, i.e.* queries that incur long running times with some implementations still remain in the “comfort zone” for other implementations

Ok, but...

...how to rank evaluators? 😞
Section 4

Multi-Criteria Experimental Ranking
An extended set of metrics

Usual metrics:
- Time  
  always
- Disk Footprint  
  only sometimes
An extended set of metrics

Usual metrics:
- Time: always
- Disk Footprint: only sometimes

Our additions:
- Disk Activity: new
An extended set of metrics

**Usual metrics:**
- Time: *always*
- Disk Footprint: *only sometimes*

**Our additions:**
- Disk Activity: *new*
- Network Traffic: *new*
An extended set of metrics

Usual metrics:
- Time: always
- Disk Footprint: only sometimes

Our additions:
- Disk Activity: new
- Network Traffic: new
- Resources: CPU, RAM, SWAP: new
Multi-Criteria Reading Grid

Criteria List

- **Velocity**: the fastest possible answers

Query Time
## Multi-Criteria Reading Grid

### Criteria List

- **Velocity**: the fastest possible answers
  - *Query Time*

- **Resiliency**: trying to avoid as much as possible to recompute everything when a machine fails
  - *Footprint*
Criteria List

- **Velocity**: the fastest possible answers

- **Resiliency**: trying to avoid as much as possible to recompute everything when a machine fails

- **Immediacy**: evaluating some SPARQL queries only once
Multi-Criteria Reading Grid

Criteria List

- **Velocity**: the fastest possible answers
  
  *Query Time*

- **Resiliency**: trying to avoid as much as possible to recompute everything when a machine fails
  
  *Footprint*

- **Immediacy**: evaluating some SPARQL queries only once
  
  *Preprocessing Time*

- **Dynamicity**: dealing with dynamic data
  
  *Preprocessing Time & Disk Activity*
Multi-Criteria Reading Grid

Criteria List

- **Velocity**: the fastest possible answers
  
- **Resiliency**: trying to avoid as much as possible to recompute everything when a machine fails

- **Immediacy**: evaluating some SPARQL queries only once

- **Dynamicity**: dealing with dynamic data

- **Parsimony**: minimizing some of the resources

---

- **Query Time**
- **Footprint**
- **Preprocessing Time**
- **Preprocessing Time & Disk Activity**
- **CPU, RAM, ...**
Immediacy

Trade-off between querying and preprocessing

The preprocessing time required before querying can be seen as an investment \textit{i.e.} taking time to preprocess data (load/index) should imply faster query response time, offseting the time spent in preprocessing.
Immediacy

Trade-off between querying and preprocessing

The preprocessing time required before querying can be seen as an investment i.e. taking time to preprocess data (load/index) should imply faster query response time, offsetting the time spent in preprocessing.

To highlight the trade-off of a query Q, we draw

- with logarithmic scale
- the affine line \( y = ax + b \)
- where:
  - \( a \) is the time to evaluate a chosen query
  - \( b \) is the preprocessing time
Immediacy

Trade-off for Q8 with Lubm1k.
Immediacy

Trade-off between querying and preprocessing

The preprocessing time required before querying can be seen as an investment \( i.e. \) taking time to preprocess data (load/index) should imply faster query response time, offsetting the time spent in preprocessing.

To highlight the trade-off of a query \( Q \), we draw

- with logarithmic scale
- the affine line \( y = ax + b \)
- where:
  - \( a \) is the time to evaluate a chosen query
  - \( b \) is the preprocessing time

To quickly answer to a single query once:

SDE should be selected.
Dynamicity & Resiliency

### Dynamicity
- SPARQL extension dedicated to updates
- 4store implements `Insert` and `Delete`
- Direct evaluators can deal with file modifications
Dynamicity & Resiliency

Dynamicity
- SPARQL extension dedicated to updates
- 4store implements Insert and Delete
- Direct evaluators can deal with file modifications

Resiliency
- Data Resiliency
## Dynamicity & Resiliency

<table>
<thead>
<tr>
<th>Systems</th>
<th>Lubm1k (GB)</th>
<th>WatDiv1k (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2RDF</td>
<td>13.057</td>
<td>15.150</td>
</tr>
<tr>
<td>RYA</td>
<td>16.275</td>
<td>11.027</td>
</tr>
<tr>
<td>CumulusRDF</td>
<td>20.325</td>
<td>–</td>
</tr>
<tr>
<td>4store</td>
<td>20.551</td>
<td>14.390</td>
</tr>
<tr>
<td>CouchBaseRDF</td>
<td>37.941</td>
<td>20.559</td>
</tr>
<tr>
<td>SPARQLGX</td>
<td>39.057</td>
<td>23.629</td>
</tr>
<tr>
<td>CliqueSquare</td>
<td>55.753</td>
<td>90.608</td>
</tr>
<tr>
<td>PigSPARQL</td>
<td>72.044</td>
<td>46.797</td>
</tr>
<tr>
<td>RDFHive</td>
<td>72.044</td>
<td>46.797</td>
</tr>
<tr>
<td>SDE</td>
<td>72.044</td>
<td>46.797</td>
</tr>
</tbody>
</table>

Without replication: Lubm1k 23GB and WatDiv1k 15GB
Dynamicity & Resiliency

**Dynamicity**
- SPARQL extension dedicated to updates
- 4store implements `Insert` and `Delete`
- Direct evaluators can deal with file modifications

**Resiliency**
- *Data Resiliency*: HDFS-based evaluators
Dynamicity & Resiliency

**Dynamicity**
- SPARQL extension dedicated to updates
- 4store implements Insert and Delete
- Direct evaluators can deal with file modifications

**Resiliency**
- *Data Resiliency*: HDFS-based evaluators
- *Computation Resiliency*
Dynamicity & Resiliency

**Dynamicity**
- SPARQL extension dedicated to updates
- 4store implements `Insert` and `Delete`
- Direct evaluators can deal with file modifications

**Resiliency**
- *Data Resiliency*: HDFS-based evaluators
- *Computation Resiliency*: 3 observed behaviors
  1. Failure: 4store, CumulusRDF
Dynamicity & Resiliency

**Dynamicity**
- SPARQL extension dedicated to updates
- 4store implements Insert and Delete
- Direct evaluators can deal with file modifications

**Resiliency**
- *Data Resiliency*: HDFS-based evaluators
- *Computation Resiliency*: 3 observed behaviors
  1. Failure: 4store, CumulusRDF
  2. Waiting:
     - A predefined number of minutes: CouchBaseRDF
     - For ever: PigSPARQL
Dynamicity & Resiliency

**Dynamicity**
- SPARQL extension dedicated to updates
- 4store implements `Insert` and `Delete`
- Direct evaluators can deal with file modifications

**Resiliency**
- *Data Resiliency*: HDFS-based evaluators
- *Computation Resiliency*: 3 observed behaviors
  1. Failure: 4store, CumulusRDF
  2. Waiting:
     - A predefined number of minutes: CouchBaseRDF
     - For ever: PigSPARQL
  3. Succeeding: RDFHive, SPARQLGX, RYA, SDE, CliqueSquare, S2RDF
Some observations:

1. CPU-consumption observation allows to see storage model.
Average CPU during Lubm1k query phase.
Parsimony

Some observations:

1. CPU-consumption observation allows to see storage model.
2. Spark-based evaluators are RAM-selfish unlike 4store or CliqueSquare.
Parsimony

Maximum allocated RAM during Lubm1k query phase.
Some observations:

1. CPU-consumption observation allows to see storage model.
2. Spark-based evaluators are RAM-selfish unlike 4store or CliqueSquare.
3. Master nodes show their role looking at network traffic.
Total bytes sent during Lubm1k query phase.
Some observations:

1. CPU-consumption observation allows to see storage model.
2. Spark-based evaluators are RAM-selfish unlike 4store or CliqueSquare.
3. Master nodes show their role looking at network traffic.

When concurrent services are running...

...4store and CliqueSquare should be selected.
Estimated Cost

**Estimation Scenario**

- MS-Azure cloud platform
- 10 “A4” instances (close to our vms in terms of ram)
- Each one costs $0.480 an hour
- $0.480 \times \text{benchTime}$, where benchTime is the sum of preprocessing and successfully answered query times.
## Estimated Cost

<table>
<thead>
<tr>
<th>Systems</th>
<th>WatDiv1k</th>
<th>Lubm1k</th>
</tr>
</thead>
<tbody>
<tr>
<td>4store</td>
<td>$9.48</td>
<td>$16.81</td>
</tr>
<tr>
<td>CliqueSquare</td>
<td>$7.81</td>
<td>$14.17</td>
</tr>
<tr>
<td>CouchBaseRDF</td>
<td>$106.29</td>
<td>$74.80</td>
</tr>
<tr>
<td>CumulusRDF</td>
<td>–</td>
<td>$1125.15</td>
</tr>
<tr>
<td>PigSPARQL</td>
<td>$36.44</td>
<td>$23.55</td>
</tr>
<tr>
<td>RDFHive</td>
<td>$7.72</td>
<td>$4.44</td>
</tr>
<tr>
<td>RYA</td>
<td>$29.40</td>
<td>$3.98</td>
</tr>
<tr>
<td>S2RDF</td>
<td>$61.17</td>
<td>$37.53</td>
</tr>
<tr>
<td>SDE</td>
<td><strong>$1.60</strong></td>
<td><strong>$0.89</strong></td>
</tr>
<tr>
<td>SPARQLGX</td>
<td>$2.69</td>
<td>$5.14</td>
</tr>
</tbody>
</table>
Estimated Cost

**Estimation Scenario**

- MS-Azure cloud platform
- 10 “A4” instances (close to our vms in terms of ram)
- Each one costs $0.480 an hour
- $10 \times \text{price} \times \text{benchTime}$, where benchTime is the sum of preprocessing and successfully answered query times.

For a small number of queries...

...SDE seems to be the less expensive.
Section 5

Conclusion & Perspectives
Conclusion

Summary:
1. Update comparative Cudré-Mauroux et al. survey
Summary:

1. Update comparative Cudré-Mauroux *et al.* survey
2. Record a larger set of metrics
Conclusion

Summary:

1. Update comparative Cudré-Mauroux et al. survey
2. Record a larger set of metrics
3. Provide a new reading grid
Final Kiviat Chart

- Immediacy
- Parsimony
- Dynamicity
- Resiliency
- Velocity: WatDiv1k
- Velocity: Lubm1k
Final Kiviat Chart
Final Kiviat Chart

- Immediacy
- Parsimony
- Dynamicity
- Resiliency
- Velocity: WatDiv1k, Lubm1k

Graph showing performance metrics for PigSPARQL and 4store in SPARQL evaluators context.
Final Kiviat Chart

- **Immediacy**
- **Parsimony**
- **Velocity**
- **WatDiv1k**
- **Lubm1k**
- **Dynamicity**
- **Resiliency**

### Evaluator Performance

#### Velocity
- **4store**
- **CliqueSquare**
- **CouchBaseRDF**
- **CumulusRDF**
- **PigSPARQL**
- **RYA**
- **SDE**
- **S2RDF**
- **SPARQLGX**
- **RDFHive**
Perspectives

Evaluators
Staying up-to-date in terms of SPARQL evaluators

Infrastructures
Testing with other clusters

Comparison Methods
Consider new benchmarks

New criteria
Extend to other criteria e.g. SPARQL covered fragment
Thanks for your attention! 😊
References
Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee.  
Diversified stress testing of RDF data management systems.  

A Barstow.  
Survey of rdf/triple data stores.  

Dave Beckett.  
Scalability and storage: Survey of free software/open source rdf storage systems.  

Dave Beckett and Jan Grant.  
Mapping semantic web data with RDBMSes.  

Christian Bizer and Andreas Schultz.  
The berlin SPARQL benchmark.  
IJSWIS, 2009.

Philippe Cudré-Mauroux, Iliya Enchev, Sever Fundatureanu, Paul Groth, Albert Haque, Andreas Harth, Felix Leif Keppmann, Daniel Miranker, Juan F Sequeda, and Marcin Wylot.  
NoSQL databases for RDF: An empirical evaluation.  

Bowllognabench – Benchmarking RDF Analytics.  

David C Faye, Olivier Curé, and Guillaume Blin.  
A survey of RDF storage approaches.  

W3C SPARQL Working Group et al.  
SPARQL 1.1 overview, 2013.  
http://www.w3.org/TR/sparql11-overview/.

Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin.  
LUBM: A benchmark for OWL knowledge base systems.  
Patrick Hayes and Brian McBride.
RDF semantics.
www.w3.org/TR/rdf-concepts/.

Zoi Kaoudi and Ioana Manolescu.
RDF in the clouds: a survey.

Ryan Lee.
Scalability report on triple store applications.

Aimilia Magkanaraki, Grigoris Karvounarakis, Ta Tuan Anh, Vassilis Christophides, and Dimitris Plexousakis.
Ontology storage and querying.

Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo.
DBpedia SPARQL Benchmark – Performance assessment with real queries on real data.

Shi Qiao and Z Meral Özsoyoğlu.
Rbench: Application-specific RDF benchmarking.

Florian Stegmaier, Udo Gröbner, Mario Döller, Harald Kosch, and Gero Baese.
Evaluation of current rdf database solutions.

Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel.
SP²Bench: a SPARQL performance benchmark.