On the Efficient Distributed Evaluation of SPARQL Queries

Damien Graux

Supervisor: Nabil Layaïda
Co-Supervisor: Pierre Genevès
Funded by: Datalyse Project

Université Grenoble Alpes, Inria, LIG

Tyrex Team
<tyrex.inria.fr>

December 15th, 2016

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: “Is it possible to sightsee at stopovers?”

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: “Is it possible to sightsee at stopovers?”

Planes

Subways

POIs

Reviews

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: “Is it possible to sightsee at stopovers?”

Planes

- Relational

Subways

- GTFS

POIs

- RDF

Reviews

- *Various*

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: "Is it possible to sightsee at stopovers?"

Planes
- Relational
- Thousand

Subways
- GTFS
- Million

POIs
- RDF
- Billion

Reviews
- *Various*
- Billion

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: "Is it possible to sightsee at stopovers?"

Planes
- Relational
- Thousand
- Static

Subways
- GTFS
- Million
- Static

POIs
- RDF
- Billion
- Dynamic

Reviews
- *Various*
- Billion
- Dynamic

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: "Is it possible to sightsee at stopovers?"

Planes

- Relational
- Thousand
- Static

Subways

- GTFS
- Million
- Static

POIs

- RDF
- Billion
- Dynamic

Reviews

- *Various*
- Billion
- Dynamic

Finally, . . .

. . . Linking the blocks!

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: "Is it possible to sightsee at stopovers?"

Complex Problem

Planes

- Relational
- Thousand
- Static

Subways

- GTFS
- Million
- Static

POIs

- RDF
- Billion
- Dynamic

Reviews

- *Various*
- Billion
- Dynamic

Finally,. . .

. . . Linking the blocks!

Context & Objectives driven by an example

Context:

- Large datasets available
- Heterogeneous data

Context & Objectives driven by an example

Context:

- Large datasets available
- Heterogeneous data

Objectives:

- Efficiently request these datasets
- Aggregate results to build complex applications

Context & Objectives driven by an example

A practical usecase:

What did you miss (touristically) last time you travelled (by plane)?

More specifically: "Is it possible to sightsee at stopovers?"

Complex Problem

Planes
- Relational
- Thousand
- Static

Subways
- GTFS
- Million
- Static

POIs
- RDF
- Billion
- Dynamic

Reviews
- *Various*
- Billion
- Dynamic

Finally, . . .

. . . Linking the blocks!

My PhD topic

Focuses

1. Focusing on evaluating SPARQL queries,
2. On large amounts of RDF data,
3. In a distributed context.

My PhD topic

Focuses

1. Focusing on evaluating SPARQL queries,
2. On large amounts of RDF data,
3. In a distributed context.

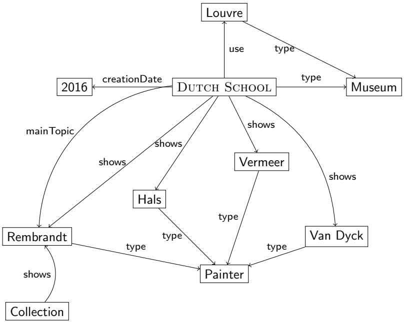
Problem

How to design efficient distributed SPARQL evaluators?

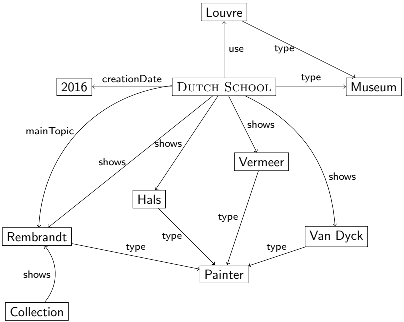
Section 1

RDF & SPARQL

Resource Description Framework [HM04]



Resource Description Framework [HM04]



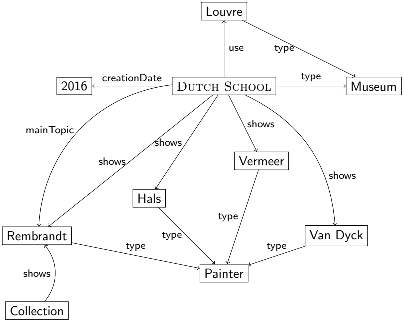
subject	predicate	object
Dutch School	type	Museum
Dutch School	creationDate	2016
Dutch School	use	Louvre
Louvre	type	Museum
Rembrandt	type	Painter
Hals	type	Painter
Vermeer	type	Painter
Van Dyck	type	Painter
Dutch School	mainTopic	Rembrandt
Collection	shows	Rembrandt
Dutch School	shows	Rembrandt
Dutch School	shows	Hals
Dutch School	shows	Vermeer
Dutch School	shows	Van Dyck

Resource Description Framework [HM04]

RDF essentials

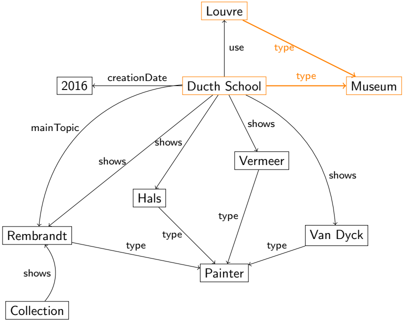
- RDF is a W3C standard
- RDF designed to provide, share and exchange datasets
- An RDF graph is a set of RDF triples
- An RDF triple has three components:
  - a subject (s)
  - a predicate (p)
  - a object (o)

SPARQL Protocol and RDF Query Language [G+13]




```
SELECT ?s ?g WHERE {
  ?s type Museum
  ?g type Painter
  ?s shows ?g
}
```

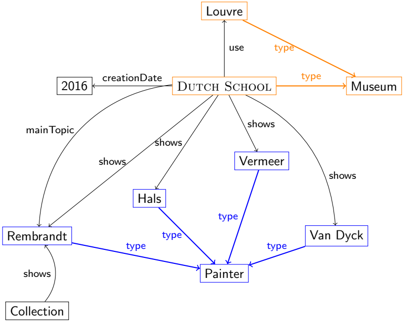
SPARQL Protocol and RDF Query Language [G+13]



?s type Museum
?g type Painter
?s shows ?g

?s: Ducth School, Louvre

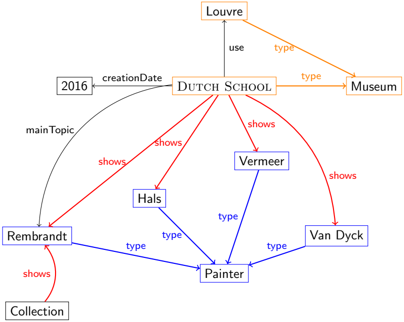
SPARQL Protocol and RDF Query Language [G+13]



?s type Museum
?g type Painter
?s shows ?g

?s: Ducth School, Louvre
?g: Rembrandt, Hals, Vermeer, Van Dyck

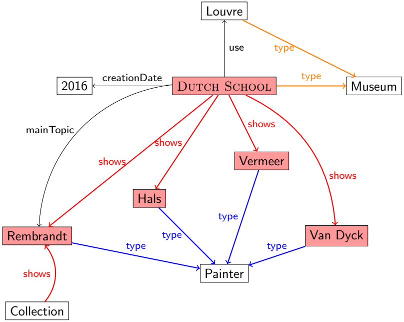
SPARQL Protocol and RDF Query Language [G+13]



?s type Museum
?g type Painter
?s shows ?g

?s: Ducth School, Louvre
?g: Rembrandt, Hals, Vermeer, Van Dyck
(?s,?g): (Ducth School,Rembrandt), (Ducth School,Hals), (Ducth School,Vermeer), (Ducth School,Van Dyck),(Collection,Rembrandt)

SPARQL Protocol and RDF Query Language [G+13]



?s type Museum
?g type Painter
?s shows ?g

?s: Ducth School, Louvre
?g: Rembrandt, Hals, Vermeer, Van Dyck
(?s,?g): (Ducth School,Rembrandt), (Ducth School,Hals), (Ducth School,Vermeer), (Ducth School,Van Dyck),(Collection,Rembrandt)

Solution (?s,?g): (Ducth School,Rembrandt), (Ducth School,Hals), (Ducth School,Vermeer), (Ducth School,Van Dyck)

SPARQL Protocol and RDF Query Language [G+13]

Considered SPARQL Fragment

- *Basic Graph Pattern* (BGP) fragment composed of conjunctions of Triple Patterns (TPs).
- *Triple Pattern* (TP)

```
SELECT ?s ?g WHERE {
  ?s type Museum
  ?g type Painter
  ?s shows ?g
}
```

- One BGP
- Composed of 3 TPs

SPARQL Protocol and RDF Query Language [G+13]

Considered SPARQL Fragment

- *Basic Graph Pattern* (BGP) fragment composed of conjunctions of Triple Patterns (TPs).
- *Triple Pattern* (TP)

```
SELECT ?s ?g WHERE {
  ?s type Museum
  ?g type Painter
  ?s shows ?g
}
```

- One BGP
- Composed of 3 TPs

Solutions

- A *candidate solution* satisfies a TP when the replacement of the variables of the TP with their value corresponds to a triple that appears in the RDF data.
- A *query solution* is a candidate solution that satisfies all the TPs of the query.

Section 2

Distributed Frameworks

MapReduce Strategy

The paradigm

- Parallel processing of massive datasets [DG08]
- A *job* has two separate phases:
  1. *Map* phase which takes k/v pairs, performs computations and returns k/v pairs
  2. *Reduce* phase where k/v pairs from the Map are ingested to return a single set of results.
- Intermediate results sometimes need to be shuffled – exchanged and/or merge-sorted – across the network to be reduced.

In brief, MapReduce

proposes to not only consider dataset as distributed and fragmented on each machine but also to develop the computation as small blocks (the Map part) which are finally grouped together (the Reduce part).

Distributed Frameworks

Hadoop

- Framework for distributed systems based on MapReduce
- It is twofold:
  - a distributed file system (including replication)
  - a MapReduce library

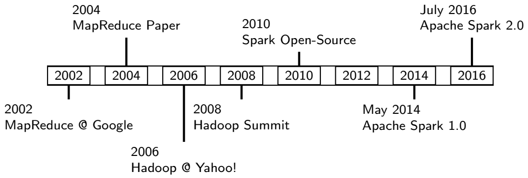
Cluster Computing Frameworks

- Provide an interface with implicit data parallelism and fault-tolerance
- Offer a set of low-level functions *e.g.* map, join, collect. . .
- For instance: PigLatin, Flink, Spark . . .

Apache Spark[ZCD+12]

Spark in a nutshell

- Master/Worker(s) Architecture
- Various file system sources supported *e.g.* HDFS
- One of the most active Apache project *e.g.* 1000+ contributors



Apache Spark[ZCD+12]

Spark in a nutshell

- Master/Worker(s) Architecture
- Various file system sources supported *e.g.* HDFS
- One of the most active Apache project *e.g.* 1000+ contributors

Resilient Distributed Datasets

- Distributed object collections
- Split into *partitions* stored in RAM or disks
- Created through deterministic operations
- Fault-tolerant: automatically re-built

Section 3

SPARQL Evaluators

Jumble of Evaluators

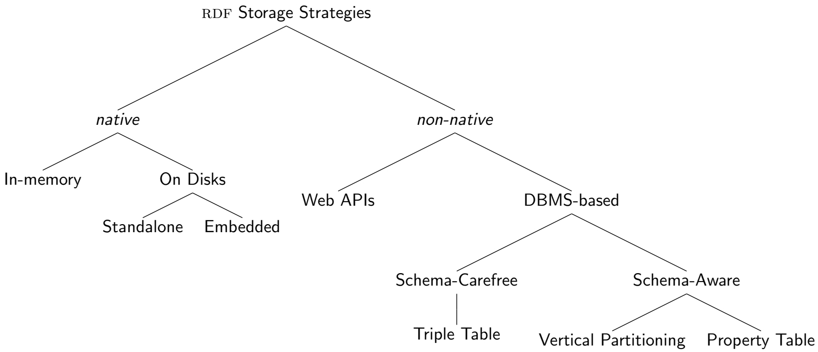
4store
CouchBaseRDF
BitMat
YARS
Hexastore
CliqueSquare
RYA
Parliament
Virtuoso
RDF-3X
. . .

Jumble of Evaluators

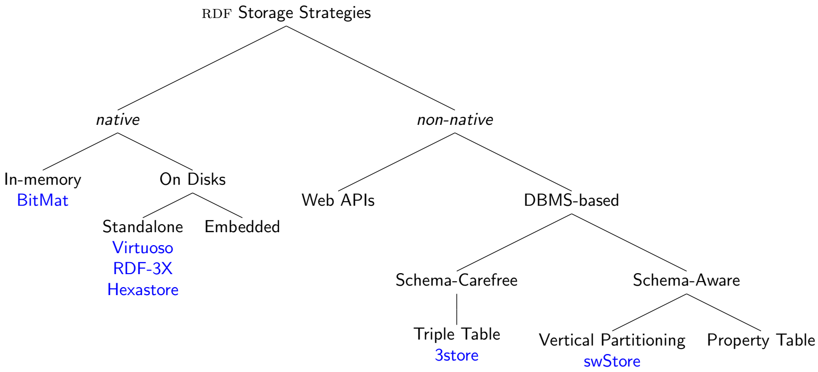
. . . Some Previous Surveys

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

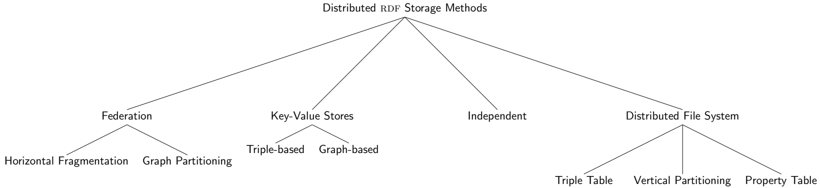
RDF Storage Strategies



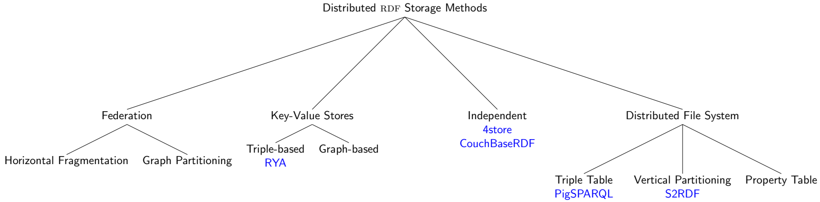
RDF Storage Strategies



Distributed Evaluation Methods



Distributed Evaluation Methods



Distributed SPARQL Evaluator State-of-the-art Summary

Observations

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

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?

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 4

Multi-Criteria Experimental Ranking

Experimental Studies

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

Popular Benchmarks

Name	**SPARQL Fragment**
LUBM [GPH05]	BGP
WatDiv [AHÖD14]	BGP
SP^2Bench [SHLP09]	BGP + `FILTER` `UNION` `OPTIONAL` + Solution Modifiers + `ASK`
BolowgnaB [DEW+11]	BGP + aggregator (*e.g.* `COUNT`)
BSBM [BS09]	BGP + `FILTER` `UNION` `OPTIONAL` + Solution Modifiers + Logical negation + `CONSTRUCT`
DBPSB [MLAN11]	Use actually posed queries against dbpedia
RBench [QÖ15]	Generate queries according to considered datasets

Popular Benchmarks

Name	SPARQL Fragment
LUBM [GPH05]	BGP
WatDiv [AHÖD14]	BGP
SP^{2}Bench [SHLP09]	BGP + `FILTER UNION OPTIONAL` + Solution Modifiers + `ASK`
BolowgnaB [DEW+11]	BGP + aggregator (*e.g.* `COUNT`)
BSBM [BS09]	BGP + `FILTER UNION OPTIONAL` + Solution Modifiers + Logical negation + `CONSTRUCT`
DBPSB [MLAN11]	Use actually posed queries against dbpedia
RBench [QÖ15]	Generate queries according to considered datasets

Contrib. 1 – Experimental Comparative Analysis

Considered Benchmarks

- LUBM: generated datasets and 14 queries (Q1-Q14)
- WatDiv: generated datasets and 20 queries

Competitors

- Selection criteria: OpenSource, Popular or Recent
- Two types of evaluators:
  - Conventional (with preprocessing): 4store, CumulusRDF, CouchBaseRDF, RYA, CliqueSquare and S2RDF
  - Direct: PigSPARQL

Contrib. 1 – Obtained Results

We learned:

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

Contrib. 1 – Obtained Results

With the following RDF datasets:

Dataset	Number of Triples	Original File Size
WatDiv1k	109 million	15 GB
Lubm1k	134 million	23 GB
Lubm10k	1.38 billion	232 GB

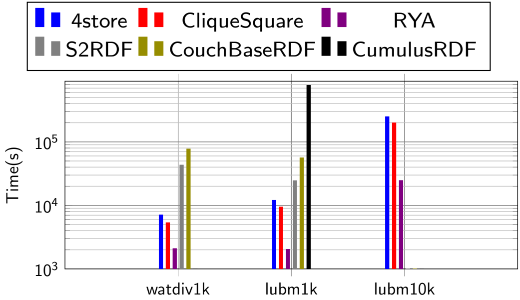


Figure : Preprocessing Time.

Contrib. 1 – 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

Contrib. 1 – Obtained Results

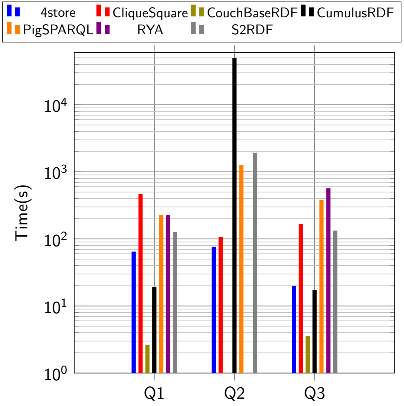


Figure : Query Response Time with Lubm1k (134 million triples).

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:memberOf ?Z .
 ?Z ub:subOrganizationOf ?Y .
 ?X ub:undergraduateDegreeFrom ?Y

}
```

Q3

```
SELECT ?X WHERE {
 ?X rdf:type ub:Publication .
 ?X ub:publicationAuthor AssistantProfessor0

}
```

Contrib. 1 – 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 incurr long running times with some implementations still remain in the "comfort zone" for other implementations

Contrib. 1 – Obtained Results

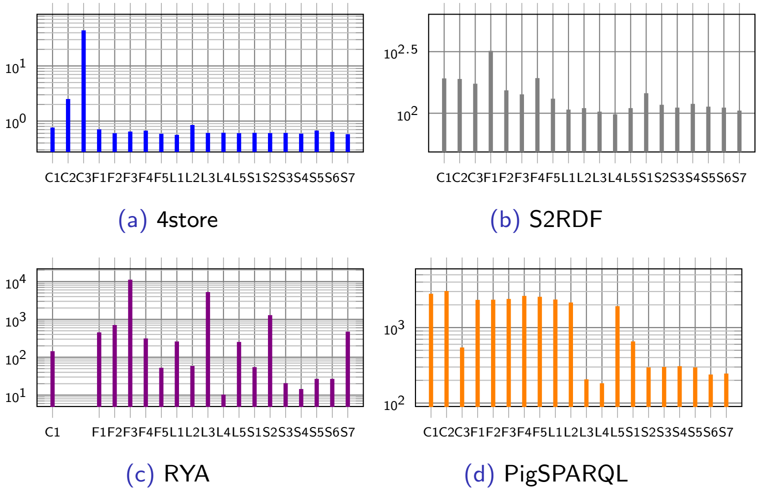


Figure : Obtained results with WatDiv1k.

Contrib. 1 – 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 incurr long running times with some implementations still remain in the "comfort zone" for other implementations

Ok, but...

... how to rank evaluators? ☹

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*

Contrib. 2 – Multi-Criteria Reading Grid

Criteria List

- **Velocity**: the fastest possible answers

Query Time

Contrib. 2 – 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

Contrib. 2 – 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

Contrib. 2 – 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

Contrib. 2 – 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

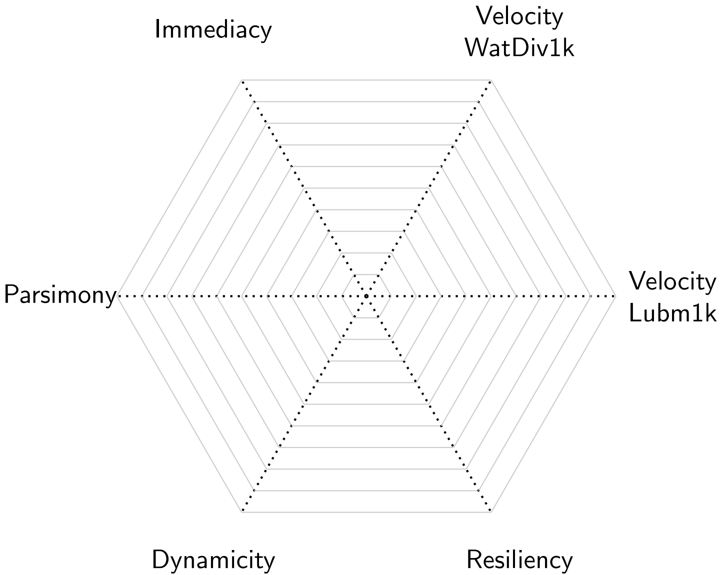
- **Dynamicity**: dealing with dynamic data

  Preprocessing Time & Disk Activity

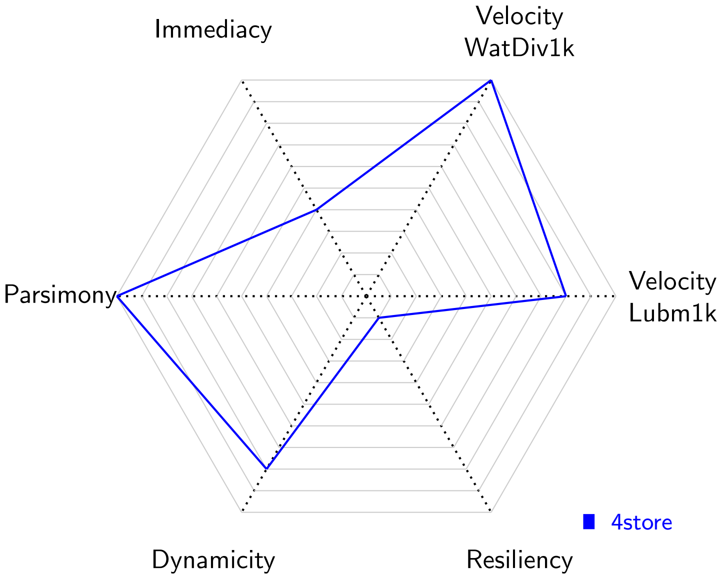
- **Parsimony**: minimizing some of the resources

  CPU, RAM, . . .

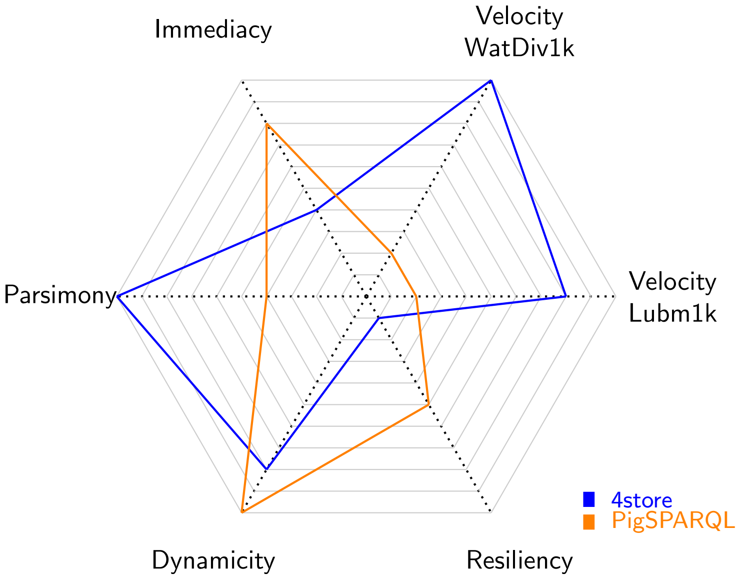
Contrib. 2 – Ranking



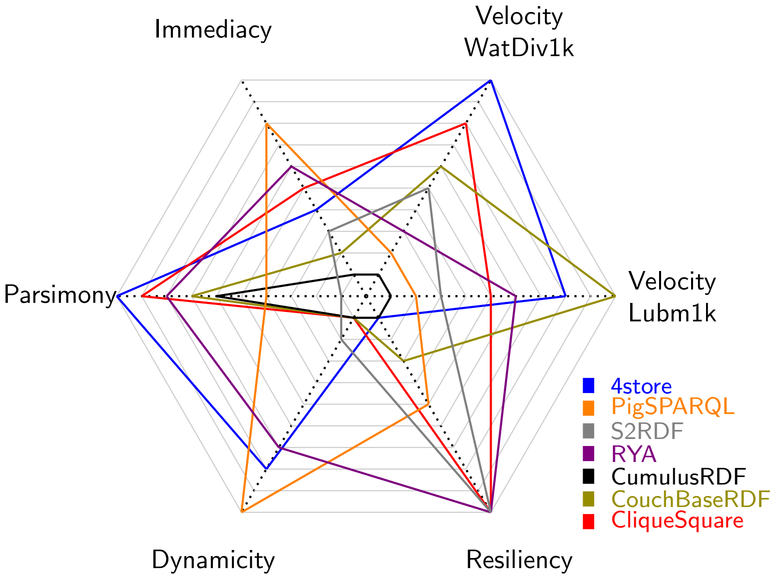
Contrib. 2 – Ranking



Contrib. 2 – Ranking



Contrib. 2 – Ranking



Section 5

Efficient Distributed SPARQL Evaluation

Contrib. 3 – Efficient Distributed SPARQL evaluation

We designed:

- **SPARQLGX**
- **SDE**
- **RDFHive**

Available from: <https://github.com/tyrex-team>

Contrib. 3 – Efficient Distributed SPARQL evaluation

These evaluators in nutshells:

- **SPARQLGX** a distributed SPARQL evaluator with Apache Spark
- **SDE** a direct SPARQL evaluator with Apache Spark
- **RDFHive** a direct evaluation of SPARQL with Apache Hive

Available from: <https://github.com/tyrex-team>

Contrib. 3 – Efficient Distributed SPARQL evaluation

Considering the reading grid, we have:

- **SPARQLGX** *velocity, resiliency*
- **SDE** *immediacy, dynamicity, resiliency*
- **RDFHive** *immediacy, dynamicity, resiliency, parsimony*

Available from: <https://github.com/tyrex-team>

Details of SPARQLGX

1. Selected storage model
2. SPARQL translation process
3. Optimization strategies

Vertical Partitioning [Abadi *et al.* 2007]

SPARQLGX Storage Model

RDF *predicates* carry the semantic information, thereby:

- Limited number of distinct predicates *e.g.* few hundreds [Gallego *et al.* 2011]
- Predicates rarely variable in queries [Gallego *et al.* 2011]

Vertical Partitioning [Abadi *et al.* 2007]

SPARQLGX Storage Model

RDF *predicates* carry the semantic information, thereby:

- Limited number of distinct predicates *e.g.* few hundreds [Gallego *et al.* 2011]
- Predicates rarely variable in queries [Gallego *et al.* 2011]

Vertical Partitioning

Splitting by predicate and saving two-column files

Vertical Partitioning [Abadi *et al.* 2007]

SPARQLGX Storage Model

RDF *predicates* carry the semantic information, thereby:

- Limited number of distinct predicates *e.g.* few hundreds [Gallego *et al.* 2011]
- Predicates rarely variable in queries [Gallego *et al.* 2011]

Vertical Partitioning

Splitting by predicate and saving two-column files

Advantages

- Natural compression and indexing
- Straightforward implementation

Vertical Partitioning [Abadi *et al.* 2007]

SPARQLGX Storage Model

dataset

Dutch School	type	Museum
Dutch School	creationDate	2016
Dutch School	use	Louvre
Louvre	type	Museum
Rembrandt	type	Painter
Hals	type	Painter
Vermeer	type	Painter
Van Dyck	type	Painter
Collection	shows	Rembrandt
Dutch School	mainTopic	Rembrandt
Dutch School	shows	Rembrandt
Dutch School	shows	Hals
Dutch School	shows	Vermeer
Dutch School	shows	Van Dyck

type.txt

Dutch School	Museum
Louvre	Museum
Rembrandt	Painter
Hals	Painter
Vermeer	Painter
Van Dyck	Painter

creationDate.txt

Dutch School	2016

use.txt

Dutch School	Louvre

shows.txt

Collection	Rembrandt
Dutch School	Rembrandt
Dutch School	Hals
Dutch School	Vermeer
Dutch School	Van Dyck

mainTopic.txt

Dutch School	Rembrar

SPARQL Translation Process

SPARQL→Scala

Dealing with one TP ...

- `textFile` to access relevant files
- `filter` to keep matching triples

SPARQL Translation Process

SPARQL→Scala

Dealing with one TP . . .

- `textFile` to access relevant files
- `filter` to keep matching triples

?s type Museum .

```
textFile("type.txt")
 .filter{case(s,o)=>o.equals("Museum")}
 .map{case(s,o)=>s}
```

SPARQL Translation Process

SPARQL→Scala

Dealing with one TP ...

- `textFile` to access relevant files
- `filter` to keep matching triples

?s type Museum .

textFile("type.txt")
.filter{case(s,o)=>o.equals("Museum")}
.map{case(s,o)=>s}

... with a conjunction of TPs

- Translate each TP
- Join them one by one

SPARQL Translation Process

SPARQL→Scala

?s type Museum .
?g type Painter .
?s shows ?g

SPARQL Translation Process

SPARQL→Scala

?s type Museum .
?g type Painter .
?s shows ?g

```
tp1=sc.textFile(''type.txt'')
  .filter{case(s,o)=>o.equals(''Museum'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
```

SPARQL Translation Process

SPARQL→Scala

?s type Museum .
?g type Painter .
?s shows ?g

```
tp1=sc.textFile(‘‘type.txt’’)
  .filter{case(s,o)=>o.equals(‘‘Museum’’)}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp2=sc.textFile(‘‘type.txt’’)
  .filter{case(g,o)=>o.equals(‘‘Painter’’)}
  .map{(g,o)=>g}
  .keyBy{case(g)=>g}
```

SPARQL Translation Process

SPARQL→Scala

```
?s type Museum .
?g type Painter .
?s shows ?g
```

```
tp1=sc.textFile(''type.txt'')
  .filter{case(s,o)=>o.equals(''Museum'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp2=sc.textFile(''type.txt'')
  .filter{case(g,o)=>o.equals(''Painter'')}
  .map{(g,o)=>g}
  .keyBy{case(g)=>g}
tp3=sc.textFile(''shows.txt'')
  .keyBy{case(s,g)=>(s,g)}
```

SPARQL Translation Process

SPARQL→Scala

```
?s type Museum .
?g type Painter .
?s shows ?g
```

```
tp1=sc.textFile(``type.txt'')
  .filter{case(s,o)=>o.equals(``Museum'')}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp2=sc.textFile(``type.txt'')
  .filter{case(g,o)=>o.equals(``Painter'')}
  .map{(g,o)=>g}
  .keyBy{case(g)=>g}
tp3=sc.textFile(``shows.txt'')
  .keyBy{case(s,g)=>(s,g)}

bgp=tp1.cartesian(tp2).values
  .keyBy{case(s,g)=>(s,g)}
  .join(tp3).value
```

Join Order

SPARQL→Scala

To minimize size of intermediate results, we try:

1. Avoiding cartesian product
2. Exploiting statistics on data

Join Order

SPARQL→Scala

To minimize size of intermediate results, we try:

1. Avoiding cartesian product
2. Exploiting statistics on data

Selectivity

- Selectivity of an element located at pos is: either its occurrence number at pos if it is a constant or the total number of triples if it is a variable.
- Selectivity of a TP is the min of its element selectivities.

We just sort the TPs of a BGP in ascending order of their selectivities.

Join Order
SPARQL→Scala

Initial BGP:

?s type Museum .
?g type Painter .
?s shows ?g

Join Order

SPARQL→Scala

Initial BGP:

?s type Museum .
?g type Painter .
?s shows ?g

New BGP:

?s shows ?g
?s type Museum .
?g type Painter

Join Order

SPARQL→Scala

Initial BGP:

?s type Museum .
?g type Painter .
?s shows ?g

New BGP:

?s shows ?g
?s type Museum .
?g type Painter

Associated Scala code:

```
tp1=sc.textFile(‘‘shows.txt’’)
  .keyBy{case(s,g)=>s}
tp2=sc.textFile(‘‘type.txt’’)
  .filter{case(s,o)=>o.equals(‘‘Museum’’)}
  .map{case(s,o)=>s}
  .keyBy{case(s)=>s}
tp3=sc.textFile(‘‘type.txt’’)
  .filter{case(s,o)=>o.equals(‘‘Painter’’)}
  .map{case(g,o)=>g}
  .keyBy{case(g)=>g}

bgp=tp1.join(tp2).values
  .keyBy{case(s,g)=>(g)}
  .join(tp3).value
```

Direct SPARQL Evaluation

Direct SPARQL Evaluation

SDE (SPARQLGX as Direct Evaluator)

- Directly considering the initial RDF dataset
- Designed to evaluate on single query

Direct SPARQL Evaluation

SDE (SPARQLGX as Direct Evaluator)

- Directly considering the initial RDF dataset
- Designed to evaluate on single query

RDFHive

- Based on Apache Hive (relational solution on the HDFS)
- Translation of queries into Hive-QL
- Offers the possibility of merging relational and RDF datasets

Direct SPARQL Evaluation

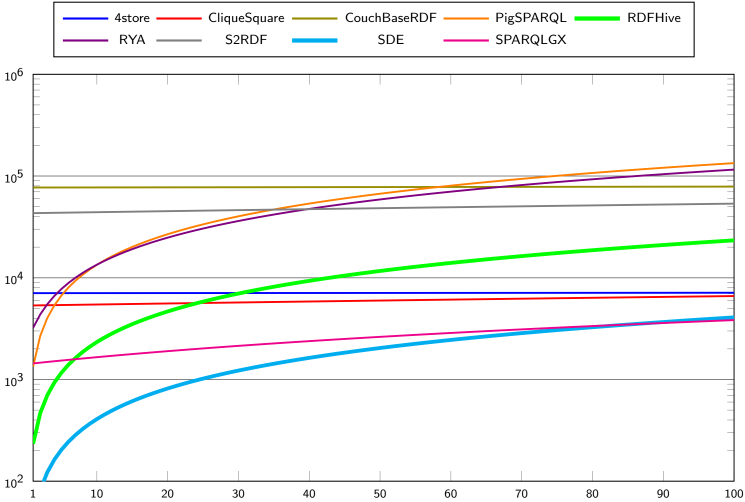


Figure : Tradeoff between preprocessing and query evaluation times (seconds) linear WatDiv.

Section 6

Conclusion & Perspectives

Conclusion

Summary of Contributions

1. Update comparative Cudré *et al.* survey — Submitted

Conclusion

Summary of Contributions

1. Update comparative Cudré *et al.* survey — Submitted
2. Provide a new reading grid (new set of metrics) — Submitted

Conclusion

Summary of Contributions

1. Update comparative Cudré *et al.* survey — Submitted
2. Provide a new reading grid (new set of metrics) — Submitted
3. Develop several distributed SPARQL evaluators:

Reusability

Openly available under the CeCILL license from:
<https://github.com/tyrex-team>

Conclusion

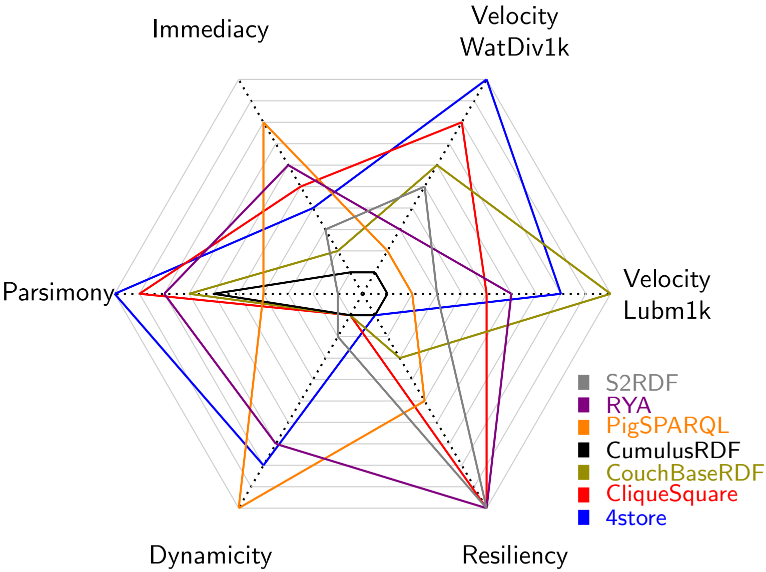
Summary of Contributions

1. Update comparative Cudré *et al.* survey — Submitted
2. Provide a new reading grid (new set of metrics) — Submitted
3. Develop several distributed SPARQL evaluators:
   - SPARQLGX — ISWC 2016
   - SDE — ISWC 2016
   - RDFHive

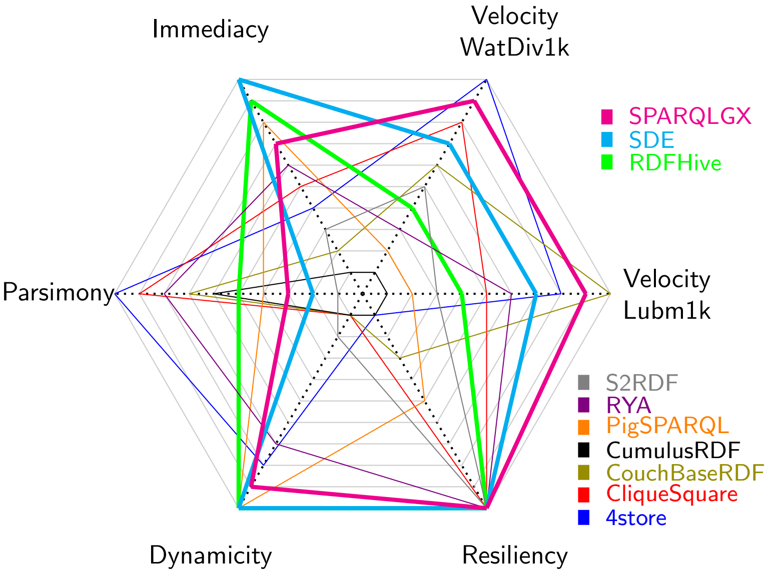
Reusability

Openly available under the CeCILL license from:
<https://github.com/tyrex-team>

Conclusion


Immediacy
Velocity
WatDiv1k
Velocity
Lubm1k
Parsimony
Dynamicity
Resiliency
S2RDF
RYA
PigSPARQL
CumulusRDF
CouchBaseRDF
CliqueSquare
4store

Conclusion


Immediacy
Velocity WatDiv1k
Velocity Lubm1k
Resiliency
Dynamicity
Parsimony
SPARQLGX
SDE
RDFHive
S2RDF
RYA
PigSPARQL
CumulusRDF
CouchBaseRDF
CliqueSquare
4store

I – Perspectives: SPARQL Benchmarking

Uniform test-suite for dynamicity — Short-Term

Designing a benchmark for the SPARQL UPDATE fragment

Staying up to date — Continuous

- Adding new evaluators
- Considering other test suites
- Benchmarking on other clusters

Varying the number of nodes — Mid-Term

- Validating our results on larger clusters
- New kind of limitation?

II – Perspectives: SPARQL Evaluators

Improving our evaluators — On going

- Extending the supported SPARQL fragment
- Improving the storage models

Designing criteria-specific evaluators — Mid-Term

- Implementing a parsimonious and resilient evaluator
- Developing evaluators in highly dynamic context

Storage-adaptative distributed evaluators — Long-Term

Adapting the idea of Aluç *et al.* [AÖD14] in a distributed context
Considering SPARQL query shapes
$\Longrightarrow$ Choosing its storage model dynamically!

III – Perspectives: Integration in ETL systems

Designing SPARQL pipeline — Mid-Term

- Using `CONSTRUCT` to refine existing RDF datasets
- Aggregating several sources into a single one

Creating heterogeneous data pipeline — Mid/Long-Term

- We provide a prototype for trip planning — ISWC 2016
- Development of a dedicated language

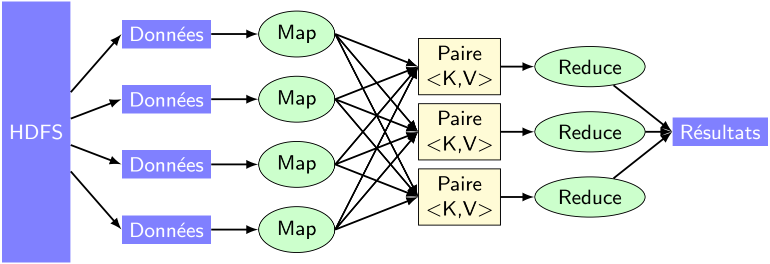
Thanks for your attention! ☺

Appendices

Appendices

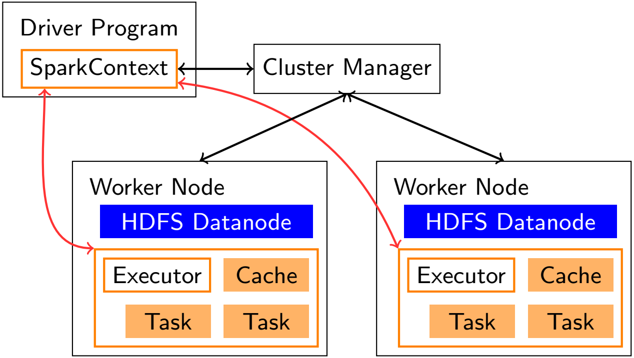
- Appendices
  - Hadoop
  - Spark
  - Cluster

Concept
Map Reduce



Architecture

Spark



1. Resource allocation *via* `cluster manager` through *master*
2. *Executors* acquisition on the cluster nodes
3. Code transfert from the application to the *executors*
4. Task transfert to the *executors*

Technical Details

Cluster of 10 nodes with 17GB of RAM each

Dataset	Number of Triples	Original File Size
WatDiv1k	109 million	15 GB
Lubm1k	134 million	23 GB
Lubm10k	1.38 billion	232 GB

References

References

Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee.
Diversified stress testing of RDF data management systems.
In *ISWC*, pages 197–212. Springer, 2014.

Güneş Aluç, M Tamer Özsu, and Khuzaima Daudjee.
Workload matters: Why rdf databases need a new design.
Proceedings of the VLDB Endowment, 7(10):837–840, 2014.

A Barstow.
Survey of rdf/triple data stores.
World Wide Web Consortium. Retrieved April, 10:2003, 2001.

Dave Beckett.
Scalability and storage: Survey of free software/open source rdf storage systems.
Latest version is available at http://www. w3. org/2001/sw/Europe/reports/rdf_ scalable_storage_report, 2002.

Dave Beckett and Jan Grant.
Mapping semantic web data with RDBMSes.
W3C Semantic Web Advanced Development for Europe (SWAD-Europe), 2003.

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.
ISWC, pages 310–325, 2013.

Gianluca Demartini, Iliya Enchev, Marcin Wylot, Joël Gapany, and Philippe Cudré-Mauroux.
Bowlognabench – Benchmarking RDF Analytics.
In *International Symposium on Data-Driven Process Discovery and Analysis*, pages 82–102. Springer, 2011.

Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: simplified data processing on large clusters.
Communications of the ACM, 51(1):107–113, 2008.

David C Faye, Olivier Curé, and Guillaume Blin.
A survey of RDF storage approaches.
Arima Journal, 15:11–35, 2012.

References

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.
Web Semantics, 2005.

Patrick Hayes and Brian McBride.
RDF semantics.
W3C recommendation, 10, 2004.
www.w3.org/TR/rdf-concepts/.

Zoi Kaoudi and Ioana Manolescu.
RDF in the clouds: a survey.
The VLDB Journal, 24(1):67–91, 2015.

Ryan Lee.
Scalability report on triple store applications.
Massachusetts institute of technology, 2004.

Aimilia Magkanaraki, Grigoris Karvounarakis, Ta Tuan Anh, Vassilis Christophides, and Dimitris Plexousakis.
Ontology storage and querying.
Ics-forth Technical Report, 308, 2002.

Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo.
DBpedia SPARQL Benchmark – Performance assessment with real queries on real data.
ISWC, pages 454–469, 2011.

Shi Qiao and Z Meral Özsoyoğlu.
Rbench: Application-specific RDF benchmarking.
In *SIGMOD*, pages 1825–1838. ACM, 2015.

Florian Stegmaier, Udo Gröbner, Mario Döller, Harald Kosch, and Gero Baese.
Evaluation of current rdf database solutions.
In *Proceedings of the 10th International Workshop on Semantic Multimedia Database Technologies (SeMuDaTe), 4th International Conference on Semantics And Digital Media Technologies (SAMT)*, pages 39–55. Citeseer, 2009.

References

Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel.
SP^{2}Bench: a SPARQL performance benchmark.
ICDE, pages 222–233, 2009.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica.
Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.
NSDI, 2012.