# Une classification expérimentale multi-critères des évaluateurs SPARQL répartis

Damien Graux, Louis Jachiet,
Pierre Genevès & Nabil Layaïda

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 }`
-

$$
\begin{array}{lcl}
\texttt{Pattern}\ (\mathsf{P}) & := & \mathsf{P}\ .\ \mathsf{P} \\
 & | & (\text{subject}|\text{var})\ (\text{predicate}|\text{var})\ (\text{object}|\text{var}) \\
 & | & \{\mathsf{P}\}\ \texttt{UNION}\ \{\mathsf{P}\} \\
 & | & \mathsf{P}\ \texttt{OPTIONAL}\ \{\mathsf{P}\} \\
 & | & \texttt{FILTER}\ \text{Constraint}
\end{array}
$$

Section 2

# SPARQL Evaluators

# Jumble of Evaluators

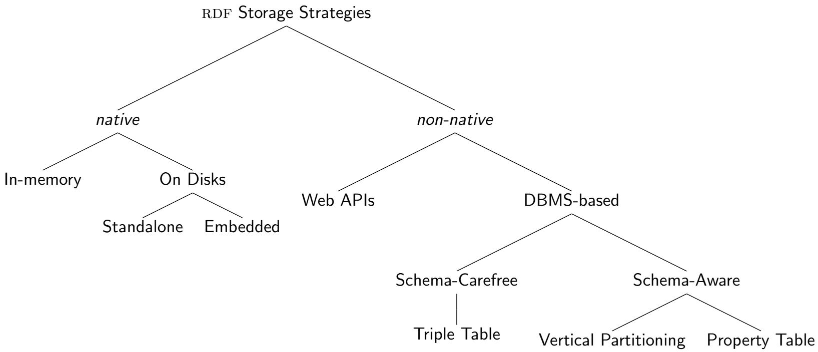
4store
CouchBaseRDF
BitMat
YARS
Hexastore
CliqueSquare
RYA
Parliament
Virtuoso
RDF-3X
...

# Jumble of Evaluators

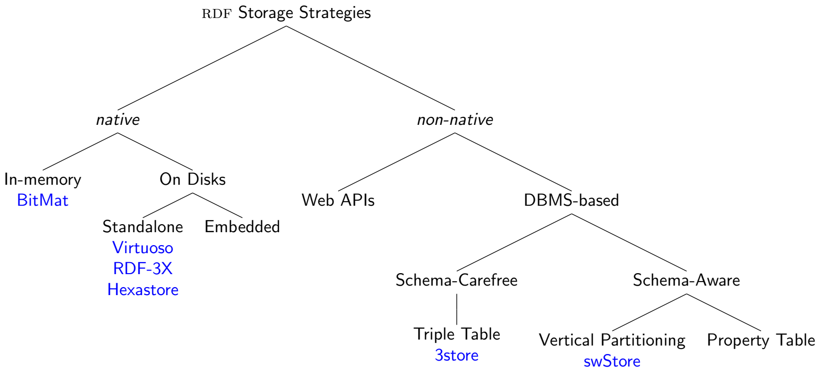
## ... Some Previous Surveys

| 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 |

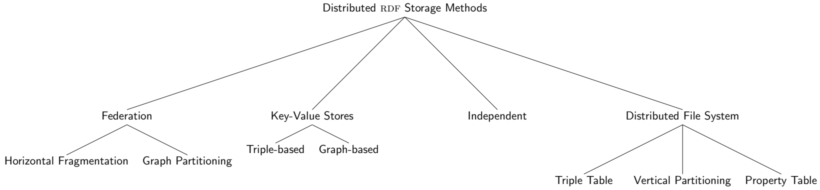
# RDF Storage Strategies [FCB12]



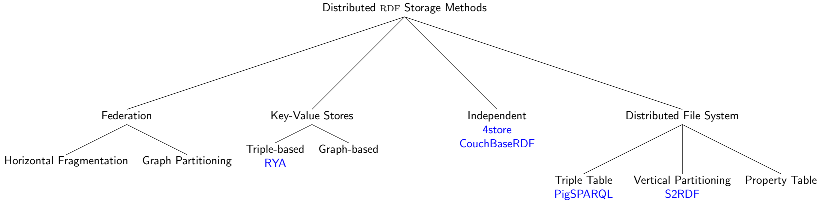
# RDF Storage Strategies [FCB12]



# Distributed Evaluation Methods [KM15]



# Distributed Evaluation Methods [KM15]



# 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 3

# Running an Experimental Study

# 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-Mauroux [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 |
| $SP^2$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 |

# 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 |

# 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

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

# 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:

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

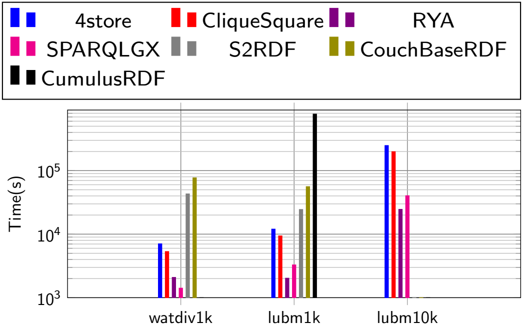



Figure : Preprocessing Time.

# 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

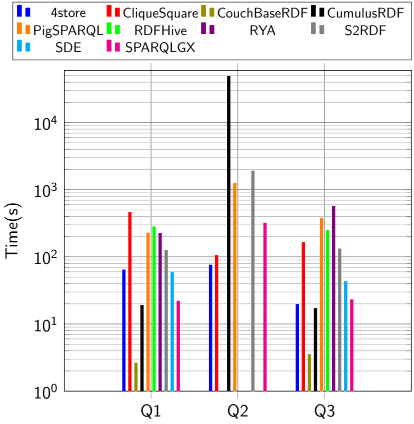



Figure : Query Response Time with Lubm1k (134 million triples).

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

}
```

**Q3**

```
SELECT ?X WHERE {
 ?X rdf:type ub:Publication .
 ?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 incurr long running times with some implementations still remain in the "comfort zone" for other implementations

# Obtained Results

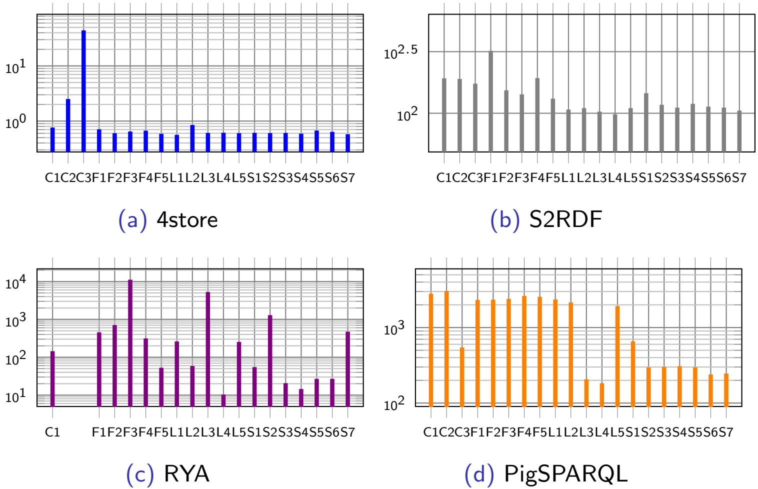



Figure : Obtained results with WatDiv1k.

# 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 incurr 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*

# 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*

# 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

  *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, . . .*

# 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, offseting the time spent in preprocessing.

# Immediacy

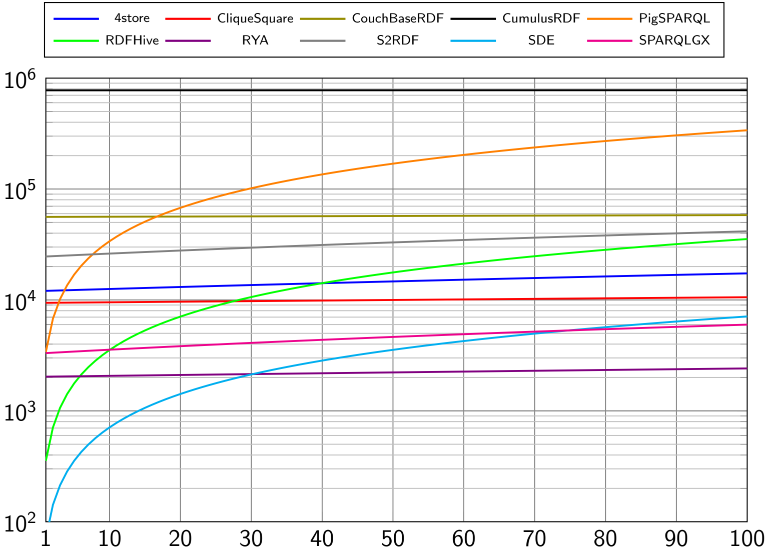
## Trade-off between querying and preprocessing

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, offseting 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, offseting 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

| Systems | Lubm1k (GB) | WatDiv1k (GB) |
|---|---|---|
| S2RDF | **13.057** | 15.150 |
| RYA | 16.275 | **11.027** |
| CumulusRDF | 20.325 | – |
| 4store | 20.551 | 14.390 |
| CouchBaseRDF | 37.941 | 20.559 |
| SPARQLGX | 39.057 | 23.629 |
| CliqueSquare | 55.753 | 90.608 |
| PigSPARQL | 72.044 | 46.797 |
| RDFHive | 72.044 | 46.797 |
| SDE | 72.044 | 46.797 |

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

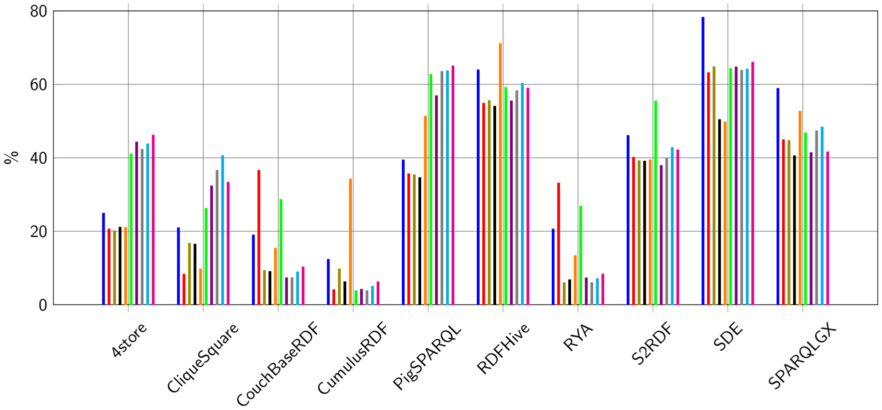
- *Data Resiliency*: HDFS-based evaluators
- *Computation Resiliency*: 3 observed behaviors
  1. Failure: 4store, CumulusRDF
  2. Waiting:
     - A predefined number of minutes: CouchBaseRDF
     - For ever: PigSPARQL
  3. Succeeding: RDFHive, SPARQLGX, RYA, SDE, CliqueSquare, S2RDF

# Parsimony

## Some observations:

1. CPU-consumption observation allows to see storage model.

# Parsimony



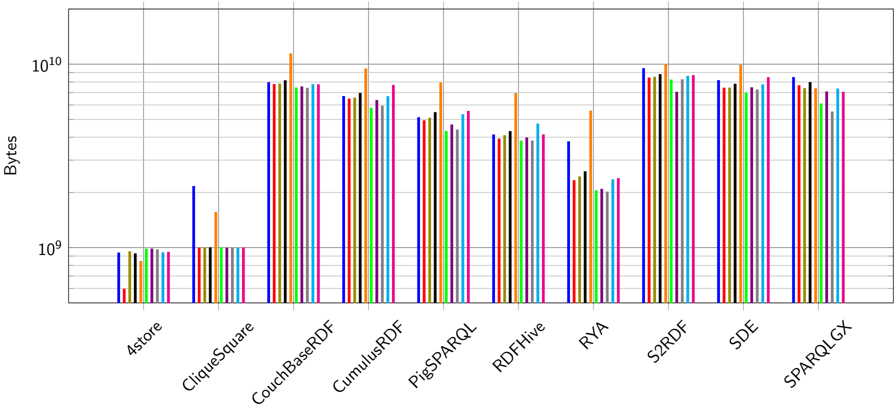

Average CPU during Lubm1k query phase.

# Parsimony

**Some observations:**

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

# Parsimony



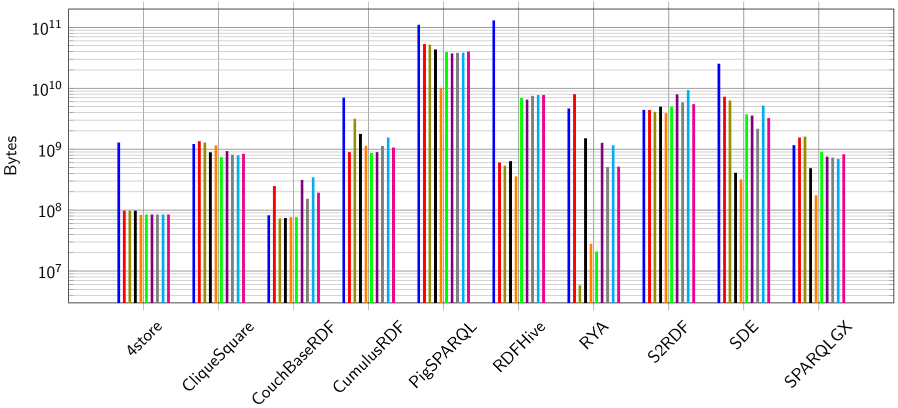

Maximum allocated RAM during Lubm1k query phase.

# Parsimony

**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.

# Parsimony




Total bytes sent 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.
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
- 10 $\times$ price $\times$ benchTime, where benchTime is the sum of preprocessing and successfully answered query times.

# Estimated Cost

| Systems | WatDiv1k | Lubm1k |
|---|---|---|
| 4store | \$9.48 | \$16.81 |
| CliqueSquare | \$7.81 | \$14.17 |
| CouchBaseRDF | \$106.29 | \$74.80 |
| CumulusRDF | – | \$1125.15 |
| PigSPARQL | \$36.44 | \$23.55 |
| RDFHive | \$7.72 | \$4.44 |
| RYA | \$29.40 | \$3.98 |
| S2RDF | \$61.17 | \$37.53 |
| SDE | **\$1.60** | **\$0.89** |
| SPARQLGX | \$2.69 | \$5.14 |

# 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 $\times$ price $\times$ 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

# Conclusion

**Summary:**

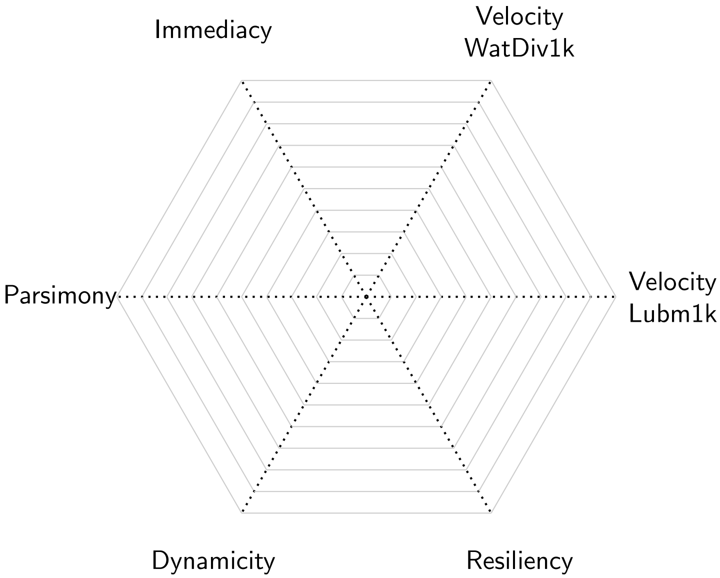
1. Update comparative Cudré-Mauroux *et al.* survey
2. Record a larger set of metrics

# Conclusion

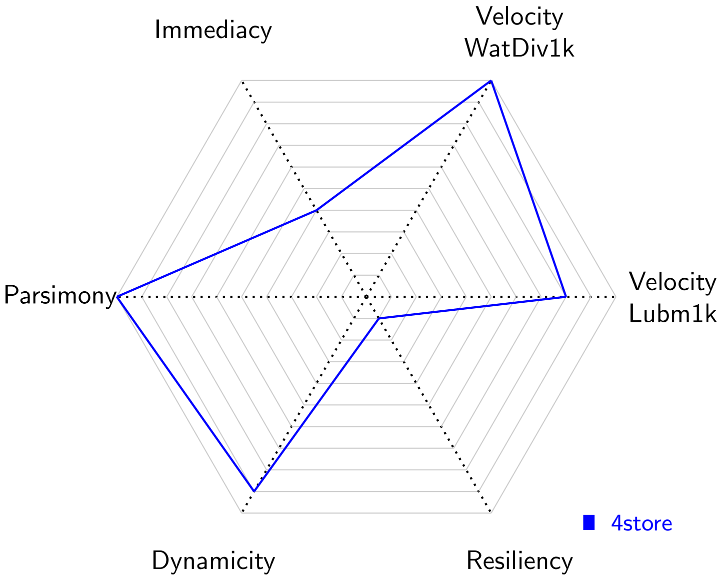
**Summary:**

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

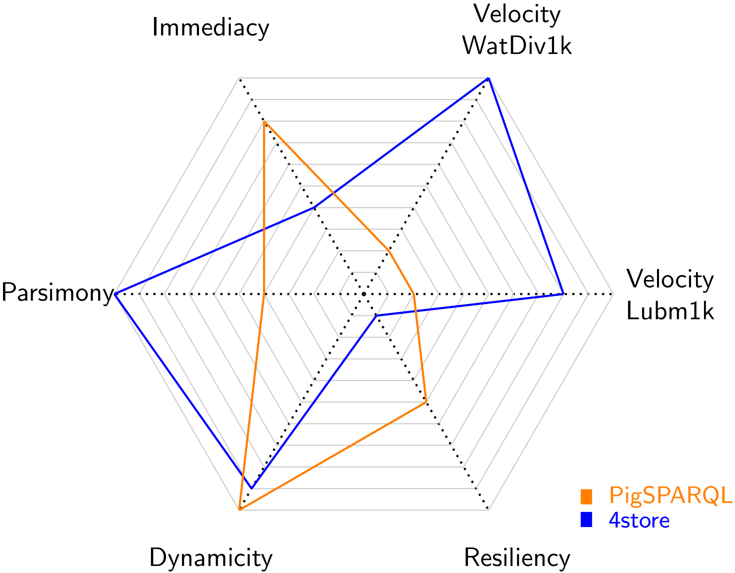
# Final Kiviat Chart



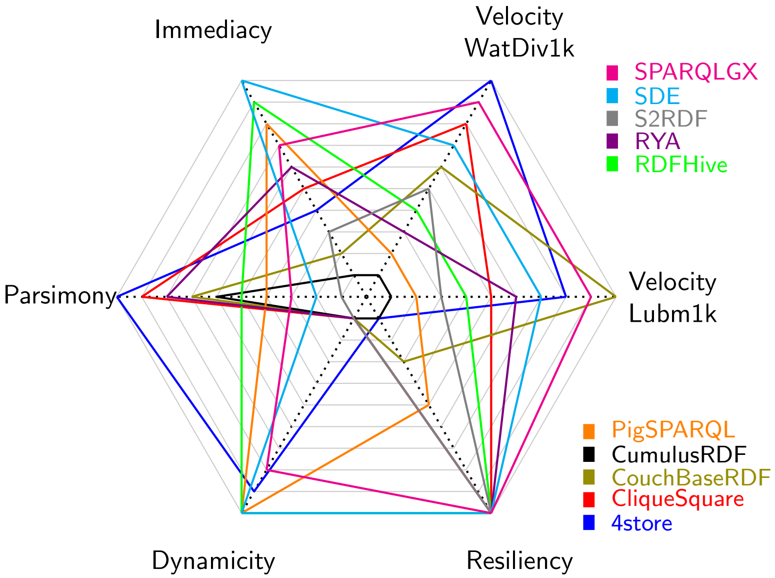
# Final Kiviat Chart



# Final Kiviat Chart



# Final Kiviat Chart



# 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

# 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.

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.

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

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.

# References


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.

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