# Une classification expérimentale multi-critère des évaluateurs SPARQL répartis

Damien Graux, Louis Jachiet, Pierre Genevès, Nabil Layaïda

# Une classification expérimentale multi-critère des évaluateurs SPARQL répartis

Damien Graux, Louis Jachiet, Pierre Genevès and Nabil Layaïda
Inria, Cnrs, Univ. Grenoble Alpes {*firstname.lastname*}@inria.fr

## ABSTRACT

SPARQL est le langage standard pour interroger des données au format RDF. Il exite une grande variété d'évaluateurs SPARQL mettant en place différentes architectures tant pour la répartition des données que pour le déroulement des calculs. Ces différences couplées à des optimisations spécifiques pour chaque évaluateur rendent la comparaison entre ces systèmes impossible d'un point de vue théorique. Nous proposons un nouvel angle de comparaison des évaluateurs SPARQL répartis basé sur un classement multi-critère. Nous suggérons d'utiliser un ensemble de cinq fonctionnalités afin d'obtenir une description plus fine des comportements des évaluateurs répartis plutôt que de considérer l'analyse plus traditionnelle des performances temporelles. Afin d'illustrer cette méthode, nous avons mené des expérimentations mettant en compétition dix systèmes existants que nous avons ensuite classés en utilisant une grille de lecture aidant à la visualisation des avantages et des limitations des techniques dans le domaine de l'évaluation répartie de requêtes SPARQL.

## 1 INTRODUCTION

We provide a new perspective on distributed SPARQL evaluators, based on a multi-criteria ranking obtained through extensive experiments. Specifically, we propose a set of five principal features which we use to rank evaluators. Each system exhibits a particular combination of these features. Similarly, the various requirements of practical use cases can also be decomposed in terms of these features. Our suggested set of features provides a more comprehensive description of the behavior of a distributed evaluator when compared to traditional performance metrics. We show how it helps in more accurately evaluating to which extent a given system is appropriate for a given use case. For this purpose, we systematically benchmarked a panel of 10 state-of-the-art implementations. We ranked them using this reading grid to pinpoint the advantages and limitations of current SPARQL evaluation systems.

## 2 DATASTORES & METHODOLOGY

We used several criteria in the selection of the SPARQL evaluators tested. First, we choose to focus on distributed evaluators. Furthermore, we retained systems that support at least a minimal fragment of SPARQL composed of conjunctive queries and called the BGP fragment. We focused on open-source systems. We wanted to include some widely used systems to have a well-known basis of comparison, as well as more recent research implementations. We also wanted our candidates to represent the variery and the richness of underlying frameworks, storage layouts, and techniques found – see *e.g.* taxonomies of [10] and [5] –, so that we can compare them on a common ground. We finally selected a panel of 10 candidate implementations:

(1) 4store is a native RDF solution introduced in [9].
(2) CumulusRDF [11] relies on Apache Cassandra.
(3) CouchBaseRDF [3] uses CouchBase.
(4) RYA [13] is a solution leveraging Apache Accumulo.
(5) SPARQLGX [7] is based on Apache Spark.
(6) S2RDF [16] uses SparkSQL.
(7) CliqueSquare [6] is a native RDF solution.
(8) PigSPARQL [15] compiles SPARQL to PigLatin.
(9) RDFHive [7] uses relational tables with Apache Hive.
(10) SDE [7] is a modification and extension of SPARQLGX.

Also for a fair comparison of the systems, we decided to rely on third-party benchmarks. The literature about benchmarks is also abundant (see *e.g.* [14] for a recent survey). For the purpose of this study, we selected benchmarks according to two conditions: (1) queries should focus on testing the BGP fragment and (2) the benchmark must be popular enough in order to allow for further comparisons with other related studies and empirical evaluations (such as [3] for instance). In this spirit, we retained the LUBM [8] and the WatDiv [1] benchmarks. During our tests we monitored each task by measuring not only time spent but a broader set of indicators: Time (Seconds), Disk footprint (Bytes), Disk activity (Bytes/second), Network traffic (Bytes/second), CPU usage (percentage), RAM usage (Bytes), and SWAP usage (Bytes).

## 3 NEW READING-GRID

We report on the overall behavior of each tested systems for these datasets: WatDiv1k (15GB), Lubm1k (23GB) and Lubm10k (232GB). These datasets constitute appropriate yardsticks for studying how the tested systems behave when the dataset size grows, with the characteristics of the cluster used. Specifically, the WatDiv1k dataset can still be held in memory of one single node, while the Lubm1k dataset becomes too large. Lubm10k is even larger than the whole available RAM of the cluster.

A first lesson learned is that, for the same query on the same dataset, elapsed times can differ very significantly from one system to another. Interestingly, we also observe that, 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, and sometimes even representing a case of demonstration of efficiency for others. For example, the response times for Q12 with Lubm1k span more than 3 orders of magnitude. Interestingly and more generally, for each query, there is at least a difference of one order of magnitude between the times spent by the fastest and the slowest evaluators.

The variety of RDF application workloads makes it hard to capture how well a particular system is suited compared to the others in a way based exclusively on time measurements. Thus, we consider five features that have different needs and where the main emerging requirement is not the same:

- *Velocity*: applications might favour the fastest answers.
- *Immediacy*: applications might need to evaluate some SPARQL queries only once. This is typically the case of some pipeline extraction applications.
- *Dynamicity*: applications might need to deal with dynamic data, requiring to react to frequent data updates.
- *Parsimony*: applications might need to execute queries while minimizing some of the resources, even at the cost of slower answers. This is for example the case of background batch jobs executed on cloud services where the main factors for the pricing of the service are network, CPU and RAM usage.
- *Resiliency*: applications that process very large data sets with complex queries might favour forms of resiliency for trying to avoid as much as possible to recompute everything when a machine fails because it is likely to happen.

Figure 1 presents a Kiviat chart in which the tested systems are ranked, based on Lubm1k and WatDiv1k according to all the features already discussed. This representation gives at a glance clues to select an evaluator. For instance it appears that 4store is especially relevant when velocity and parsimony are important and less importance is given to resiliency. SDE also appears as a reasonnable choice when all criteria (including its potential cost on a cloud platform) but parsimony matter.

## 4 RELATED WORK & CONCLUSION

This study benefited from the extensive earlier works on benchmarks for RDF systems. There are many benchmarks designed for evaluating RDF systems [1, 2, 4, 8, 12, 14, 17]. Compared to all these works, we focus on testing distribution techniques by considering a set of 10 state-of-the-art implementations; see *e.g.* [5, 10] for recent surveys about distributed RDF datastores and their storage approaches. Compared to studies included in the aforementioned benchmarks, we measure a broader set of indicators encompassing *e.g.* network usage. This allows to refine the comparative analysis according to features and requirements from a slightly higher perspective by identifying the bottlenecks of each system when they are pushed to the limits.

We conducted an empirical evaluation of 10 state-of-the-art distributed SPARQL evaluators on a common basis. By considering a full set of metrics, we improve on traditional empirical studies which usually focus exclusively on temporal considerations. We proposed five new dimensions of comparison that help in clarifying the limitations and advantages of SPARQL evaluators according to use cases with different requirements.

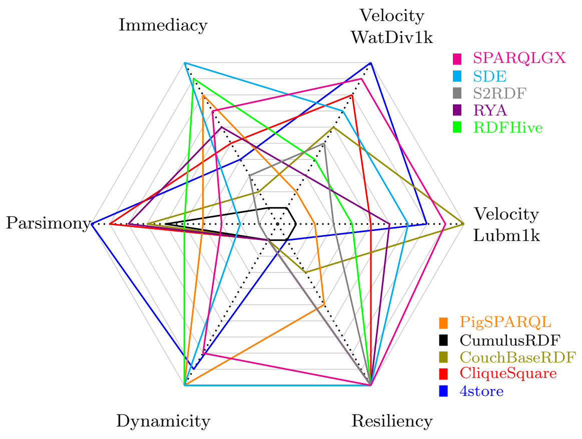

**Figure 1: System Ranking (farthest is better).**

## REFERENCES

[1] Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee. 2014. Diversified stress testing of RDF data management systems. In *ISWC*. Springer, 197–212.
[2] Christian Bizer and Andreas Schultz. 2009. The berlin SPARQL benchmark. *IJSWIS*.
[3] Philippe Cudré-Mauroux, Iliya Enchev, Sever Fundatureanu, Paul Groth, Albert Haque, Andreas Harth, Felix Leif Keppmann, Daniel Miranker, Juan F Sequeda, and Marcin Wylot. 2013. NoSQL databases for RDF: An empirical evaluation. *ISWC*, 310–325.
[4] Gianluca Demartini, Iliya Enchev, Marcin Wylot, Joël Gapany, and Philippe Cudré-Mauroux. 2011. BowlognaBench – Benchmarking RDF Analytics. In *International Symposium on Data-Driven Process Discovery and Analysis*. Springer, 82–102.
[5] David C Faye, Olivier Curé, and Guillaume Blin. 2012. A survey of RDF storage approaches. *Arima Journal* 15, 11–35.
[6] François Goasdoué, Zoi Kaoudi, Ioana Manolescu, Jorge-Arnulfo Quiané-Ruiz, and Stamatis Zampetakis. 2015. CliqueSquare: Flat plans for massively parallel RDF queries. In *ICDE*. IEEE, 771–782.
[7] Damien Graux, Louis Jachiet, Pierre Genevès, and Nabil Layaïda. 2016. SPARQLGX: Efficient Distributed Evaluation of SPARQL with Apache Spark. *ISWC*.
[8] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics*.
[9] Steve Harris, Nick Lamb, and Nigel Shadbolt. 2009. 4store: The design and implementation of a clustered RDF store. *SSWS*.
[10] Zoi Kaoudi and Ioana Manolescu. 2015. RDF in the clouds: a survey. *The VLDB Journal* 24, 1, 67–91.
[11] Günter Ladwig and Andreas Harth. 2011. CumulusRDF: linked data management on nested key-value stores. *SSWS 2011*, 30.
[12] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. 2011. DBpedia SPARQL Benchmark – Performance assessment with real queries on real data. *ISWC*, 454–469.
[13] Roshan Punnoose, Adina Crainiceanu, and David Rapp. 2012. RYA: a scalable RDF triple store for the clouds. In *International Workshop on Cloud Intelligence*. ACM, 4.
[14] Shi Qiao and Z Meral Özsoyoğlu. 2015. Rbench: Application-specific RDF benchmarking. In *SIGMOD*. ACM, 1825–1838.
[15] Alexander Schätzle, Martin Przyjaciel-Zablocki, and Georg Lausen. 2011. PigSPARQL: Mapping SPARQL to Pig Latin. In *Proceedings of the International Workshop on Semantic Web Information Management*. ACM, 4.
[16] Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, and Georg Lausen. 2016. S2RDF: RDF Querying with SPARQL on Spark. *VLDB*, 804–815.
[17] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. 2009. SP$^2$Bench: a SPARQL performance benchmark. *ICDE*, 222–233.