universität freiburg

Sparqloscope

A generic benchmark for the comprehensive and concise performance evaluation of SPARQL engines

Talk @ ISWC 2025

Nara, 4th November 2025

Hannah Bast, J. Kalmbach, C. Ullinger, R. Textor-Falconi
Department of Computer Science
University of Freiburg

Performance evaluation is hard

- Let's look at the following SPARQL query on Wikidata

```
SELECT ?person ?name ?image (COUNT(*) AS ?count)
WHERE {
  ?person wdt:P31/wdt:P279* wd:Q5 .
  ?person wdt:P18 ?image .
  ?person rdfs:label ?name FILTER (LANG(?label) = "en") .
  ?person ^schema:about/schema:isPartOf ?site .
}
GROUP BY ?person ?name ?image
ORDER BY DESC(?count)
```

People on Wikidata ranked by the number of pages linking to them

https://qlever.dev/wikidata/moLK2D

Performance evaluation is hard

- Results for six RDF databases on Wikidata truthy, 8 B triples
  (measured on an AMD Ryzen 9 with enough RAM and disk space)

Apache Jena	Java	**837s**
Blazegraph	Java	**157s**
GraphDB	Java	**153s**
MillenniumDB	C++	**101s**
Virtuoso	C	**79s**
QLever	C++	**3s**

All six systems implement the SPARQL standard (with some deviations)

Why the huge performance differences?

Performance evaluation is hard

- Why the huge performance differences

```
SELECT ?person ?label ?image (COUNT(*) AS ?count) WHERE {
  ?person wdt:P31/wdt:P279* wd:Q5 .
  ?person wdt:P18 ?image .
  ?person rdfs:label ?label FILTER (LANG(?label) = "en") .
  ?person ^schema:about/schema:isPartOf ?site .
} GROUP BY ?person ?label ?image ORDER BY DESC(?count)
```

Is it because of the first property path?	wdt:P31/wdt:P279* is huge
Is it because of the huge rdfs:label?	717 millions triples
Is it because of the large GROUP BY?	1.2 million groups
Is it because of the large final result?	materialize 1.2M × 4 table

Or is it the combination of these or of some of these?

And is this query typical or an outlier?

Sparqloscope

- We introduce a new benchmark called **Sparqloscope**, with three distinguishing features
  - **Specific** Exactly one query for each SPARQL feature
    Pinpoint what a system does good and what it does bad
  - **Comprehensive** Covers most of the SPARQL standard
    Most other benchmarks focus on a subset of the features
  - **Generic** Generates benchmark for any given RDF dataset
    Input: RDF dataset, output: benchmark for that dataset
  - **Easy to use** You just need a SPARQL endpoint for the dataset
    python3 generate-benchmark.py --sparql-endpoint …

  The idea is that **you cannot overfit on Sparqloscope**
  if a system is good on Sparqloscope, then it's a good system

Queries

- Sparqloscope has only 105 queries ... here a four of them
  - JOIN of two very large tables with a small result
    Evaluates how efficient the basic join algorithm is
  - GROUP BY with COUNT, with many small groups
    Different algorithm required for "many groups" vs. "few groups"
  - LANGuage filter on large predicate
    Used in many queries, a good system should optimize this
  - Export a very large result
    Systems vary widely in their ability to materialize large results

  Sparqloscope finds queries with these properties automatically
  (via carefully engineered SPARQL queries on the given endpoint)

  See **https://github.com/ad-freiburg/sparqloscope**

Performance evaluation

- Part of the paper is a performance evaluation of the six aforementioned systems on two RDF datasets
  - **DBLP** ~ 500 million triples
  - **Wikidata Truthy** ~ 8 billion triples

We would have loved to evaluate on larger datasets as well, but most systems struggle already with 10 billion triples

Side remark: QLever can handle more than **one trillion triples** (on a single machine, without any special treatment whatsoever)

Performance evaluation

- Results on DBLP with ~ 500 million triples
  ("Query time" is the geometric mean over all 105 queries)

System	Code	Load time	Index size	Query time
Apache Jena	Java	70.6 min	54 GB	15.3 s
Blazegraph	Java	40.3 min	34 GB	13.4 s
GraphDB	Java	25.4 min	35 GB	5.8 s
MillenniumDB	C++	14.7 min	20 GB	1.3 s
Virtuoso	C	12.4 min	14 GB	0.5 s
QLever	C++	5.3 min	9 GB	0.2 s

Detailed results on **https://qlever.dev/evaluation**

Performance evaluation

- Results on Wikidata Truthy with ~ 8 billion triples
  ("Query time" is the geometric mean over all 105 queries)

System	Code	Load time	Index size	Query time
Apache Jena	Java	21.0 h	684 GB	216.0 s
Blazegraph	Java	22.6 h	500 GB	124.4 s
GraphDB	Java	20.4 h	453 GB	108.3 s
MillenniumDB	C++	4.2 h	317 GB	23.3 s
Virtuoso	C	13.8 h	373 GB	10.1 s
QLever	C++	3.1 h	149 GB	2.3 s

Detailed results on **https://qlever.dev/evaluation**