How to Benchmark RDF Engines and Not Die Trying

Tomas Lampo[1], Amadís Martínez[2,3], Edna Ruckhaus[2] and María-Esther Vidal[2]

[1] University of Maryland, College Park, USA
tlampo@cs.umd.edu
[2] Universidad Simón Bolívar, Caracas, Venezuela
{amadis,ruckhaus,mvidal}@ldc.usb.ve
[3] Universidad de Carabobo, Valencia, Venezuela

Abstract. Existing RDF engines have developed optimization techniques and efficient physical operators that speed up execution time. Additionally, some of these approaches have implemented structures to efficiently store and access RDF data, and have developed execution strategies able to reuse data previously stored in cache. In order to study the benefits of the techniques offered by each approach, particular datasets have to be considered as well as different benchmarks of queries. Queries must be of different complexity, number of patterns, selectivity and shape, while datasets have to be of different sizes and correlations between subjects, predicates and objects. In this paper, we describe the benchmarks that we have developed to analyze the performance of existing RDF engines when creating and indexing RDF structures, as well as the quality of the optimization and execution strategies implemented by these engines.

1 Introduction

In the context of the Semantic Web, several query engines have been developed to access RDF documents efficiently [3, 10, 11, 13, 14, 18, 22, 25]. The majority of these approaches have implemented optimization techniques and developed execution engines on top of effective physical operators and accurate estimators [3, 15, 18, 22]. Additionally, some of these approaches have implemented structures to efficiently store and access RDF data, and have developed execution strategies able to reuse data previously stored in the cache. To evaluate the performance of the implemented techniques, datasets and queries with specific properties must be taken into account [9]. Queries should be of different complexity, number of patterns, selectivity and shape. Datasets ought to be of different sizes and correlations between subjects, predicates and objects; also, they can be characterized according to the number of classes, properties, axioms, hierarchies or instances.

In this paper, we aim at the characterization of benchmarks that exhibit the performance of existing RDF engines. We are particularly interested in identifying the type of datasets that can be used to analyze the efficiency and effectiveness of loading and indexing procedures, as well as the properties of the queries that can stress existing optimization and query execution techniques. Thus, more than proposing a benchmark, our purpose is to illustrate how issues such as the size and correlations between RDF data, the shape of query plans, and the selectivity of intermediate join results, can reveal important characteristics of existing RDF engines.

Proceedings of the International Workshop on Evaluation of Semantic Technologies (IWEST 2010). Shanghai, China. November 8, 2010.

First, the dataset size is a relevant issue in experiments where the load and index time are analyzed; thus, we consider synthetic datasets like LUBM [7], or real-world datasets like YAGO [4] and government data. YAGO can be classified as a large dataset, LUBM can be generated with an increasing number of concepts, while government data repositories can be extended by considering different years and RDF properties. Thus, these datasets can be used to evaluate RDF engines' scalability, i.e., how much the load time is affected when the dataset size increases.

Furthermore, queries that can be partitioned into large numbers of star shaped groups, can reveal an interesting behavior of state-of-the-art optimization and execution techniques. Additionally, queries that produce a very large number of intermediate results or require CPU-intensive processing, can show the efficacy of cache management techniques and physical operators. In this paper, we characterize some significant properties of the benchmarks, illustrate the results that we have obtained by exploiting them in our empirical studies, and motivate the consideration of these issues during benchmarking.

To summarize, the main contributions of this paper are the following:

- The characterization of a family of query plans that can benefit from warming up cache. These queries reduce the number of intermediate results and CPU processing, and can be rewritten as bushy plans comprised of small-sized star-shaped groups.
- Benchmarks comprised of queries and plans that show how a query plan shape can impact on an RDF query engine performance.
- An empirical study of the RDF engines' performance in terms of load and indexing time, and query optimization and execution time.

This paper is comprised of four additional sections. Section 2 describes the main properties of existing RDF engines and benchmark frameworks; section 3 presents the characteristics of the developed benchmarks. Section 4 presents an experimental study where we report on the performance of state-of-the-art RDF engines. Finally, we conclude in section 5 with an outlook to future work.

2 Existing RDF Engines and Benchmark Frameworks

During the last years, several query engines have been developed to access RDF data [3, 10, 11, 13, 14, 18, 25]. Jena [13, 26] provides a programmatic environment for SPARQL; it includes the ARQ query engine and indices, which provide an efficient access to large datasets. Tuple Database or TDB [14] is a persistent graph storage layer for Jena; it works with the Jena SPARQL query engine (ARQ) to support SPARQL together with a number of extensions (e.g., property functions, arbitrary length property paths). Sesame [25] is an open source Java framework for storing and querying RDF data; it supports SPARQL and SeRQL queries, which are translated to Prolog. These engines have not been defined to scale up to large datasets or complex queries, so benchmarks comprised of queries of different complexity and large datasets could be used to stress these engines' performance.

[4] Ontology available for download at *http://www.mpi-inf.mpg.de/yago-naga/yago/*

Additionally, different storage and access structures have been proposed to efficiently retrieve RDF data [6, 17, 23, 24]. Hexastore [24] is a main memory indexing technique that exploits the role of the arguments of an RDF triple; six indices are designed so that each one can efficiently retrieve a different access pattern. A secondary-memory-based solution for Hexastore has been presented in [23]; this solution scales up to larger datasets, but because the same object may be stored in several indices, memory can be used inefficiently. AllegroGraph [3] uses a native object store for on-disk binary tree-based storage of RDF triples. AllegroGraph also maintains six indices to manage all the possible permutations of subject (s), predicate (p) and object (o). The standard indexing strategy is to build indices whenever there are more than a certain number of triples. Fletcher et al. [6] propose indexing the universe of RDF resource identifiers, regardless of the role played by the resource; although they are able to reduce the storage costs of RDF documents, since the proposed join implementations are not closed with respect to the index structures, the properties of these structures can only be exploited in joins on basic graph patterns. In addition to query complexity and dataset size that affect the performance of these engines, another issue that needs to be considered in benchmarks for these engines, is the correlation between the values of the subjects, predicates and objects; this property can help to impact on the load and index time.

MacGlothlin et al. [16] propose an index-based representation for RDF documents that materializes the results for subject-subject joins, object-object joins and subject-object joins. This approach has been implemented on top of MonetDB [12] and it can exploit the Monet DB cache management system. Recently, Atre et al. [4] proposed the BitMat approach which is supported on a fully inverted index structure that implements a compressed bit-matrix structure of the RDF data. An RDF engine has been developed on top of this bit-based structure, which exploits the properties of this structure and avoids the storage of intermediate results generated during query evaluation. Although these structures can speed up the evaluation of joins, this solution may not scale up to very large strongly connected RDF graphs. Thus, to analyze the performance of these approaches, dataset size and correlation between the values of the subjects, predicates and objects have to be set up during benchmark configuration.

Abadi et al. [1, 2] and Sidirourgos et al. [21] propose different RDF storage schemas to implement an RDF management system on top of a relational database system. They empirically show that a physical implementation of vertically partitioned RDF tables may outperform the traditional physical schema of RDF tables. In addition, any of these solutions can exploit the properties of the database manager to efficiently manage the cache. RDF-3X [18] focuses on an index system, and has implemented optimization and evaluation techniques that support an efficient and scalable evaluation of RDF queries. RDF-3X optimizer relies on a dynamic-based programming algorithm that is able to identify left linear plans comprised of star-shaped groups; however, this optimizer does not scale up to complex queries. In addition, RDF-3X is able to load in resident memory portions of data, and thus differences between execution time in both cold and warm cache can be observed for certain types of queries. Additionally to the properties of the benchmarks previously described, the shape of the queries and the selectivity of the intermediate joins, should be considered in the benchmarks.

Furthermore, in the context of the Semantic Web, benchmarking has motivated the evaluation of these query engines, and contributed to improve scalability and performance [9]. Among the most used benchmarks, we can mention: LUBM [8], the Berlin SPARQL Benchmark [5], the RDF Store Benchmarks with DBpedia[5], and the SP^{2}Bench SPARQL benchmark [20].

The LUBM benchmark has been defined to compare performance, completeness and soundness of OWL reasoning engines. This allows us to generate ontologies of different sizes and expressiveness, and also provides a set of queries; so performance and scalability of reasoning tasks such as subsumption, realization and transitive closure, can be evaluated. The Berlin SPARQL Benchmark is settle in an e-commerce domain, and it has been designed to test storage workloads and the performance SPARQL endpoints on large datasets; properties, depth and width of the subsumption hierarchies can be configured, as well as queries that emulate search and navigation patterns in the generated datasets. The SP^{2}Bench SPARQL benchmark generates DBLP-like RDF documents, and a set of SPARQL queries that cover a large number of SPARQL operators whose performance can be studied in different RDF engines. Finally, the RDF Store Benchmarks with DBpedia provides a set of SPARQL queries to be executed against DBpedia and provides the basis to test the performance of existing RDF engines when small portions of a large dataset are required to execute a query.

Additionally, RDF engines' authors have conducted empirical studies with selected datasets and tailored queries to exhibit the engines' performance [4, 16, 19, 22]. Atre et al. [4] chose UniProt and a synthetic LUBM dataset with 10,000 universities to stress the BitMat storage capabilities; in addition, previously published queries, were adapted to reveal the performance of the BitMat query processing algorithms in complex join queries with low-selectivity intermediate results. Neumann et al. [19] used two large datasets to exhibit storage workload properties, and a set of queries comprised of chained triple patterns to exploit performance of warm caches, physical join operators, and selectivity estimation techniques; although the queries are composed of up to 14 triple patterns, they are not complex enough to stress the optimizer or to diminish warm cache performance.

Similarly to existing benchmarks, we tailored a family of queries that allow us to reveal the performance of existing RDF engines; however, we focus on illustrating the impact of the shape of query plans on the performance of warm caches and physical join operators.

3 Analysis of the Benchmark Characteristics

Several factors are considered for query benchmarks: the number of patterns in each query, the number of instantiated patterns, the number of answers (including no-answer queries), the number of intermediate triples that are produced during its evaluation, and the query groupings together with their size and shape, e.g., bushy plans comprised of small-sized star-shaped groups. All of these features are adjusted according to the properties of the RDF engines on which the experimental study is being developed.

[5] http://www4.wiwiss.fu-berlin.de/benchmarks-200801/

For example, RDF-3X implements a dynamic-based query optimizer that does not easily scale-up to queries with a large number of sub-goals or star-shaped join binding patterns[6], so query size and shape should be considered. Additionally, RDF-3X offers cache techniques for an efficient execution, so query benchmarks must be comprised of queries that produce a large number of intermediate results that require CPU-intensive processing.

Furthermore, we have developed extensions to some of the existing RDF query engines, this is the case of GRDF-3X and GJena, which implement a version of the gjoin operator that we have defined to combine small-sized star-shaped groups in a bushy plan; for these extensions it is important to define benchmarks that contain bushy plans comprised of small-sized star-shaped groups [22].

The influence of the benchmark factors on the RDF engine features is illustrated in Table 1.

Table 1. Relationship between benchmark factors and RDF engine features

benchmark factor	**cache management**	**query optimization**	**physical operators**	**indexing structures**
# patterns		X	X	
# pattern instantiations			X	X
# small star-shaped groups	X	X	X	
# intermediate triples	X		X	
# number of answers			X	
different instantiation values		X		

RDF engines may implement different types of optimization algorithms. Some engines implement greedy algorithms that explore a limited number of query plans. When the number of patterns increases, the optimization effectiveness may be compromised. Other engines implement dynamic programming-based algorithms that only explore linear plans; so performance of these algorithms may be degraded when the number of query patterns is large or the join binding pattern is not a chain. Randomized optimization algorithms, when properly configured, may efficiently identify bushy query execution plans; however, temperature and number of iterations have to be adjusted.

Queries that generate a large number of intermediate results will not benefit from warming the cache because of frequent page faults. On the other hand, bushy plans comprised of small-sized star-shaped group may produce intermediate join results that fit in memory and could be reused in further join operations; thus, cold cache times can be reduced for these types of queries.

Indexing structures improve triple search on subject, property or object specific values. Therefore, index structures are better exploited when patterns are instantiated. Also, the number of answers affects the performance of RDF engines that can detect if the query answer is empty; so queries that produce empty answers are also required.

[6] Queries comprised of star-shaped join binding patterns can be shaped as bushy plans composed of star-shaped groups.

Finally, query optimization may be affected by dependencies among property values and non-uniform distribution of these values. Thus, query optimization that relies on cost estimates that do not take into account these dependencies, may take wrong decisions based on imprecise estimates. Thus, benchmarks that contain queries with same patterns, orderings and groupings but different instantiations of dependent values, should be also considered.

4 Experimental Study

Considering the characteristics described in previous sections, we have defined several benchmarks and conducted extensive experimental studies. We considered the following RDF engines: RDF-3X versions 0.3.3 and 0.3.4, AllegroGraph RDFStore version 3.0.1, Jena version 2.5.7, Jena with TDB version 0.7.3., and GRDF-3X[7] built on top of RDF-3X version 0.3.4. We studied efficiency and effectiveness of the load and indexing processes, as well as the optimization and evaluation techniques of these engines; additionally, we studied the effects of query shape in cache usage.

Dataset	Number of triples	N3-Size
LUBM $Univ(1,0)$	103,074	17.27 MB
LUBM $Univ(5,0)$	645,649	108.28 MB
LUBM $Univ(10,0)$	1,316,322	220.79 MB
LUBM $Univ(20,0)$	2,781,322	468.68 MB
LUBM $Univ(50,0)$	6,888,642	1.16 GB
US Congress votes 2004	67,392	3.61 MB
YAGO	44,000,000	4.0 GB
wikipedia	47,000,000	6.9 GB

(a) Dataset Cardinality

Benchmark	**Dataset**	**# Queries**	**# Basic Patterns (min;max)**
1	US Congress votes 2004	9	(3;7)
2	US Congress votes 2004	20	(12;35)
3	YAGO	10	(17;25)
4	YAGO	9	(3;7)
5	YAGO	9	(17;27)
6	LUBM	8	(1,6)

(b) Query Benchmark Description

Fig. 1. Datasets and Benchmarks

Datasets: we used four different datasets: the Lehigh University Benchmark (LUBM) [7], Wikipedia [8], the real-world dataset on US Congress vote results of 2004, and the

[7] Our extension of RDF-3X is able to efficiently execute bushy plans.

[8] http://labs.systemone.net/wikipedia3

real-world ontology YAGO [9]. We generated the LUBM datasets for 1, 5, 10, 20 and 50 universities. The number of triples and size of the N-triple representation of each dataset is described in Figure 1(a).

Query Benchmarks: We considered six sets of queries (Figure 1). Benchmark 1 is comprised of queries that have at least one pattern whose object is instantiated with a constant and the answer size varies from 3 to 14,868 triples. Benchmark 2 has queries composed between one and seven gjoin(s) among small-sized groups of patterns; queries have more than 12 triple patterns. All queries in Benchmark 3 have an empty answer, except $q6$ that produces 551 triples. Benchmark 4 is comprised of queries that answer between 1 and 5,683 triples, while answers of Benchmark 5 queries range from 238 to 22,434. Finally, benchmark 6 is comprised of the first eight queries of the LUBM benchmark. The first five benchmarks are published in *http://www.ldc.usb.ve/~mvidal/OneQL/datasets/queries/*.

Evaluation Metrics: We report on runtime performance and optimization time, which are both measured by using the *real time* produced by the *time* command of the Linux operation system. Runtime represents the elapsed time between the submission of the query and the output of the answer; optimization time just considers the time elapsed between the submission of the query and the output of the query physical plan. Experiments were run on a Sun Fire X4100 M2 machine with two AMD Opteron 2000 Series processors, 1MB of cache per core and 8GB RAM, running a 64-bit Linux CentOS 5.5 kernel. Queries in benchmark 4 and 5 were run in cold-cache and warm cache. To run cold cache, we cleared the cache before running each query by performing the command `sh -c "sync ; echo 3 > /proc/sys/vm/drop_caches"`. To run on warm cache, we executed the same query five times by dropping the cache just before running the first iteration of the query. Additionally, the machine was dedicated exclusively to run these experiments.

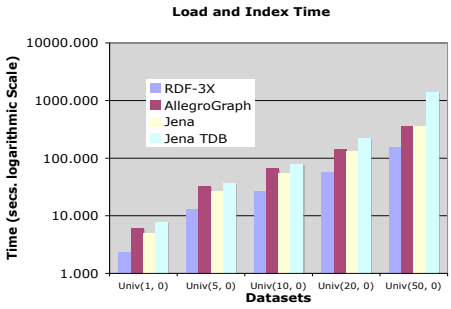
4.1 Loading and Indexing RDF Data

We used the LUBM dataset to study the time and memory consumed by RDF-3X, AllegroGraph RDFStore, Jena, and Jena TDB during loading and indexing RDF data. We report on the average execution time and the amount of main-memory required to store the structures.

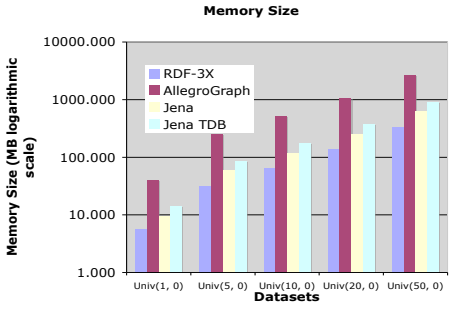
Figure 2(a) reports on the time (seconds and logarithm scale) required to load and index all triples of each dataset. In all test cases, LUBM datasets allow us to observe that RDF-3X is faster than the other RDF engines in loading and indexing RDF data. This may be because RDF-3X exploits in-memory caching to maintain hash tables that store: mappings between strings/URIs in RDF data to unsigned integer IDs, a single triples table of these IDs, and the highly compressed indices of the triples table.

Figure 2(b) reports on the amount of main-memory required for the structures to store all triples of a dataset (MB and logarithm scale). RDF-3X makes use of LZ77 compression in conjunction with byte-wise Gamma encoding for all the IDs and gap compression [19]; thus, RDF-3X is able to consume less main-memory than the rest of the RDF engines.

[9] Ontology available for download at *http://www.mpi-inf.mpg.de/yago-naga/yago/*



(a) Load and Index time-LUBM datasets (secs. and log scale)



(b) Memory Size of Index Structures-LUBM datasets (MB and log scale)

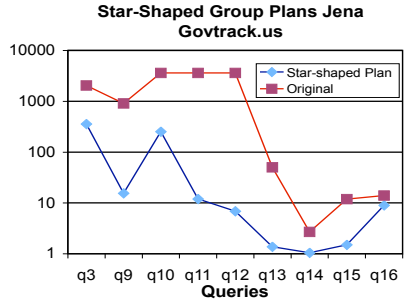
Fig. 2. LUBM datasets (logarithmic scale)

4.2 Query Optimization and Execution Techniques

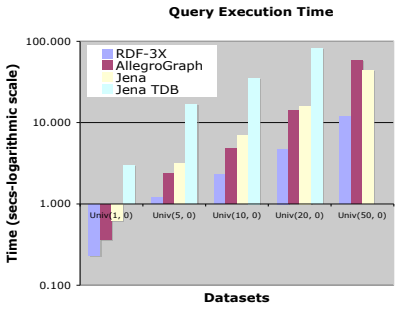
We used all the datasets and benchmarks to analyze the RDF engines' performance. First, we studied how the shape of a query affects the optimization and execution time. We considered the queries in benchmark 2, and for each query we built a bushy plan comprised of star-shaped groups free of Cartesian products. We ran different versions of Jena and Jena TDB, and we could observe that bushy plans comprised of star-shaped groups outperform flat queries (Figure 3(a)).

Second, we analyzed how the size of a dataset can affect the performance of RDF engines, i.e., we evaluated scalability. We used benchmark 6 against the LUBM datasets for 1, 5, 10, 20, and 50 universities and studied the performance of these queries on RDF-3X, AllegroGraph, Jena and Jena TDB. Figure 3(b) reports on the average query execution time over each LUBM dataset (seconds and logarithmic scale). All the RDF

engines except Jena TDB, spent, on average, less than 1 second running the queries against the dataset for 1 university. Also, all the queries were comprised of a large number of join binding patterns, thus the performance of the RDF-3X Merge join and indices could be exploited and RDF-3X overcame other RDF engines in the five datasets. In the case of Jena TDB, we turned off the optimizer and we could observe that queries against the dataset with 50 universities timed out after 14 hours in Jena TDB.



(a) Effects Query Shape Jena (secs. and log scale)



(b) Average Query Execution Time-Benchmark 6 (secs. and log. scale)

Fig. 3. Effects of the Query Shape

Third, we studied how the shape of the queries affects the cache usage by using benchmarks 3, 4 and 5; for each query, we also built a bushy plan comprised of star-shaped groups. We observed that in benchmark 3, RDF-3X is able to improve cold cache execution times by a factor of 35 in the geometric mean when the queries were

run in warm cache. However, for benchmarks 4 and 5, we could observe that RDF-3X performs poorly in warm cache and flat queries, while in bushy plans comprised of star-shaped groups, CPU time was reduced from 96% to 25% and the cold cache execution time was sped by up to one order of magnitude. Table 2 reports on cold cache execution times, the minimum value observed during the execution in warm cache, and the geometric means. Flat queries were run in RDF-3X, and the star-shaped group bushy plans were run in GRDF-3X. First, flat queries are dominated by CPU-intensive processing that consumed up to 98% of the CPU time; however, star-shaped group bushy plans consumed up to 25% of the CPU time, and the execution time in both cold and warm caches was reduced by up to five orders of magnitude. Finally, because the star-shaped group queries were bushy trees comprised of small-sized star-shaped groups, the number of intermediate results was smaller; thus, intermediate results could be maintained in resident memory and used in further iterations.

Table 2. Benchmark 5-Run-Time Cold and Warm Caches (secs)

Cold Caches										
Flat	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
Queries	62.30	84.87	100,657.34	85.95	61.2	188,909.69	0.14	1.47	827.75	166.03
Bushy Trees	1.60	1.80	2.34	1.22	1.38	1.36	0.99	1.05	1.75	1.45
Warm Caches										
Flat	q1	q2	q3	q4	q5	q6	q7	q8	q9	Geom. Mean
Queries	58.21	59.54	72,584.71	58.52	59.73	175,909.80	0.14	1.46	808.77	144.13
Bushy Trees	0.34	0.26	0.93	0.14	0.31	0.17	0.12	0.31	0.69	0.29

Finally, we used benchmarks 4 and 5 to study how query plan shapes affect the performance of the RDF-3X optimizer. We could observe that for some queries, more than 93% of the total execution time was spent in query optimization and the generation of the physical plan. The reason for this is that the RDF-3X optimizer relies on a dynamic-based programming algorithm that traverses the space of linear plans in iterations, where linear sub-plans comprised of i joins are generated during iteration i. Although the space of linear plans can be considerably smaller than the space of bushy plans, this approach does not scale up to queries of more than 20 triple patterns with star-shaped join binding patterns, such as the ones in benchmark 5.

5 Conclusions

In this paper, we characterized some of the properties of the benchmarks that can be used to evaluate the performance of existing RDF engines when RDF structures are created and indexed, and queries are optimized and executed. We have identified that dataset sizes and query plan shapes must also be considered during benchmarking to exhibit the performance of existing RDF engines.

First, we have confirmed that the RDF-3X query engine and data structures scale up to large datasets and to large number of chained join binding patterns. We also reported experimental results suggesting that the benefits of running in warm caches depend on the shape of executed queries. For simple queries, the RDF-3X engine is certainly able to benefit from warming up cache; however, for queries with several star-shaped groups, the RDF-3X optimizer generates left-linear plans that may produce a large number of intermediate results or require CPU-intensive processing that degrades the RDF-3X engine performance in both cold and warm caches. On the other hand, if these queries are rewritten as bushy plans, the number of intermediate results and the CPU processing can be reduced and the performance improves. So, we recommend to consider the query plan shape during benchmarking.

In the future we plan to study the effects of the correlations between the instantiations of the queries and between the subjects, predicates and objects of the RDF data; additionally, we plan to conduct a similar study in Sesame, MonetDB, BitMat and RDFVector. Finally, developing tools able to generate the described benchmarks, is also in our future plans.

References

1. D. J. Abadi, A. Marcus, S. Madden, and K. Hollenbach. SW-Store: a vertically partitioned DBMS for Semantic Web data management. *VLDB Journal*, 18(2):385–406, 2009.
2. D. J. Abadi, A. Marcus, S. Madden, and K. J. Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 411–422, 2007.
3. AllegroGraph. http://www.franz.com/agraph/allegrograph/, 2009.
4. M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler. Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In *Proceedings of the WWW*, pages 41–50, 2010.
5. C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.
6. G. Fletcher and P. Beck. Scalable Indexing of RDF Graph for Efficient Join Processing. In *CIKM*, 2009.
7. Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 274–288, Japan, 2004.
8. Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.
9. Y. Guo, A. Qasem, Z. Pan, and J. Heflin. A requirements driven framework for benchmarking semantic web knowledge base systems. *IEEE Trans. Knowl. Data Eng.*, 19(2):297–309, 2007.
10. A. Harth, J. Umbrich, A. Hogan, and S. Decker. YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In *Proceedings of the The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC/ASWC)*, pages 211–224, 2007.
11. G. Ianni, T. Krennwallner, A. Martello, and A. Polleres. A Rule System for Querying Persistent RDFS Data. In *Proceedings of the 6th European Semantic Web Conference (ESWC2009)*, Heraklion, Greece, May 2009. Springer. Demo Paper.
12. S. Idreos, M. L. Kersten, and S. Manegold. Self-organizing tuple reconstruction in column-stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–308, 2009.

13. Jena Ontology Api. http://jena.sourceforge.net/ontology/index.html, 2009.
14. Jena TDB. http://jena.hpl.hp.com/wiki/TDB, 2009.
15. T. Lampo, E. Ruckhaus, J. Sierra, M.-E. Vidal, and A. Martinez. OneQL: An Ontology-based Architecture to Efficiently Query Resources on the Semantic Web. In *The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems at the International Semantic Web Conference (ISWC)*, 2009.
16. J. McGlothlin. RDFVector: An Efficient and Scalable Schema for Semantic Web Knowledge Bases. In *Proceedings of the PhD Symposium ESWC*, 2010.
17. J. McGlothlin and L. Khan. RDFJoin: A Scalable of Data Model for Persistence and Efficient Querying of RDF Dataasets. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2009.
18. T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *PVLDB*, 1(1):647–659, 2008.
19. T. Neumann and G. Weikum. Scalable join processing on very large rdf graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 627–640, 2009.
20. M. Schmidt, T. Hornung, N. Küchlin, G. Lausen, and C. Pinkel. An experimental comparison of rdf data management approaches in a sparql benchmark scenario. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 82–97, 2008.
21. L. Sidirourgos, R. Goncalves, M. L. Kersten, N. Nes, and S. Manegold. Column-store support for RDF data management: not all swans are white. *PVLDB*, 1(2):1553–1563, 2008.
22. M.-E. Vidal, E. Ruckhaus, T. Lampo, A. Martinez, J. Sierra, and A. Polleres. Efficiently Joining Group Patterns in SPARQL Queries. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC2010)*, 2010.
23. C. Weiss and A. Bernstein. On-disk storage techniques for semantic web data are b-trees always the optimal solution? In *The 5th International Workshop on Scalable Semantic Web Knowledge Base Systems at the International Semantic Web Conference (ISWC)*, 2009.
24. C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1):1008–1019, 2008.
25. J. Wielemaker. An Optimised Semantic Web Query Language Implementation in Prolog. In *Proceedings of the ICLP Conference*, pages 128–142, 2005.
26. K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, and J. Database. Efficient RDF Storage and Retrieval in Jena2. In *EXPLOITING HYPERLINKS 349*, pages 35–43, 2003.