Benchmarking Fulltext Search Performance of RDF Stores

Enrico Minack, Wolf Siberski, and Wolfgang Nejdl

L3S Research Center
Leibniz Universität Hannover
30167 Hannover, Germany
lastname@L3S.de

Abstract. More and more applications use the RDF framework as their data model and RDF stores to index and retrieve their data. Many of these applications require both structured queries as well as fulltext search. SPARQL addresses the first requirement in a standardized way, while fulltext search is provided by store-specific implementations. RDF benchmarks enable developers to compare structured query performance of different stores, but for fulltext search on RDF data no such benchmarks and comparisons exist so far. In this paper, we extend the LUBM benchmark with synthetic scalable fulltext data and corresponding queries for fulltext-related query performance evaluation. Based on the extended benchmark, we provide a detailed comparison of fulltext search features and performance of the most widely used RDF stores. Results show interesting RDF store insights for basic fulltext queries (classic IR queries) as well as hybrid queries (structured and fulltext queries). Our results are not only valuable for selecting the right RDF store for specific applications, but also reveal the need for performance improvements for certain kinds of queries.

1 Introduction

Since the emergence of the Semantic Web [1], more and more data is stored in semi-structured form, using the RDF framework[1]. This annotated data becomes easier processable, or even "understandable" by machines. However, the data on the Semantic Web is frequently related to textual human-readable information (e.g., publication metadata). Therefore, most applications that provide their users access to RDF data also need support for fulltext search. This was confirmed in the Semantic Web Survey [2], where all 35 studied semantic search applications index textual data, and most of them provide fulltext search.

Whereas structured query languages such as SQL or SPARQL [3] provide a high expressivity regarding the *structure* of the data, fulltext queries target at the *content*. Integrating fulltext search (IR) [4] with structured search (DB) [5] is an actively discussed undertaking [6,7]. A lot of research has been

[1] Resource Description Framework: http://www.w3.org/RDF/

L. Aroyo et al. (Eds.): ESWC 2009, LNCS 5554, pp. 81–95, 2009.

conducted in DB fields such as relational DBMS [8,9], XML databases [10], and RDF databases [11,12], to name only some. Fulltext search clearly adds a large set of rich features to SPARQL, *e.g.* boolean, phrase, wildcard, and proximity queries [4]. A score that represents the relative relevance of matching resources can also be obtained. Further, the so called *snippet generation* provides a small summary of the text around the matching keywords.

While SPARQL [3] offers a high expressivity with respect to structured queries, fulltext-related queries are not well supported. The only feature being targeted on fulltext search is regular expression evaluation against string values. Interestingly, none of the widely used RDF stores support the standardized regular expressions efficiently, *i.e.,* with the help of fast index structures [13,14]. The reason is probably that regular expressions are much more expressive than required, and thus much more difficult to implement in an efficient manner, than the usual keyword search. In contrast, all RDF store implementations support efficient keyword search, putting significant effort in a feature that is *not* part of the W3C Recommendation. Some RDF indexing systems even consider the combination of inverted keyword indices and statement indices as fundamental building blocks for efficient RDF indexing [15]. This investment signifies the high demand of Semantic Web applications for support of fulltext search or *hybrid queries* which combine structured and fulltext queries [11].

For benchmarking RDF stores, a number of benchmarks have been proposed which evaluate performance regarding structured queries [16,17,18]. However, none of them addresses fulltext search capabilities. Therefore, RDF store developers test their fulltext search implementations with their own ad-hoc benchmarks [15,12,19], rendering efficiency evaluations difficult to be repeated or compared. A commonly available RDF fulltext benchmark is still missing.

In this paper, we propose such a benchmark that measures performance and feature richness of fulltext search facilities of RDF stores[2]. More precisely, we extend the well-known and widely used *Lehigh University Benchmark* (LUBM) [16] with fulltext content and hybrid queries. Using this new benchmark, we evaluate the most widely used open-source RDF stores and discuss the results. In particular, our work makes the following contributions:

1. We identify properties of datasets needed for an RDF fulltext benchmark, taking into account real fulltext characteristics such as term distribution.
2. We extend the LUBM benchmark dataset accordingly, and propose an algorithm for scalable generation of synthetic fulltext literals.
3. We design RDF fulltext queries to investigate different aspects of fulltext search on RDF, and evaluate well-known open-source RDF stores with our benchmark (in alphabetical order): *Jena*, *Sesame2*, *Virtuoso*, and *YARS*.

The paper is structured as follows. In section 2, we define objectives and desired features of an RDF fulltext benchmark, and describe our extension of the LUBM benchmark. Section 3 presents our evaluation of popular RDF stores against our proposed RDF fulltext benchmark. Relevant related work is discussed in section 4. We close with a conclusion.

[2] LUBMft: http://www.l3s.de/%7eminack/rdf-fulltext-benchmark/

2 A Fulltext Extension for LUBM

2.1 LUBM Overview and Discussion

The *Lehigh University Benchmark*[3] (LUBM) is a very frequently used synthetic benchmark for RDF stores [20,15,12]. It provides the *Univ-Bench Artificial data generator* (UBA) and a set of 14 test queries. Its dataset complies to the *Univ-Bench ontology*[4] describing universities. It contains 43 classes, such as `ub:University`, `ub:Department`, `ub:Professor`, and `ub:Publication`. It further contains 25 object properties (pointing to resources), as well as 7 datatype properties (pointing to literals), such as `ub:name` and `ub:publicationAuthor`. Datasets are generated using a scaling factor N which determines the size of the generated dataset and serves to investigate RDF store scalability. In addition, a seed M for the random number generator can be specified to allow repeated regeneration of the same synthetic dataset. These generated datasets are referred to as *LUBM(N, M)*. The notation *LUBM(N)* indicates that M is set to 0.

Even though this benchmark was primarily designed to evaluate OWL and DAML+OIL capabilities of RDF stores, it was also very often used to benchmark RDF stores and RDF related systems without OWL or DAML+OIL support [20,15,12]. This may be due to the very easy generation of arbitrary size RDF datasets and the familiar ontology domain. As LUBM, our benchmark works perfectly on stores without any kind of reasoning.

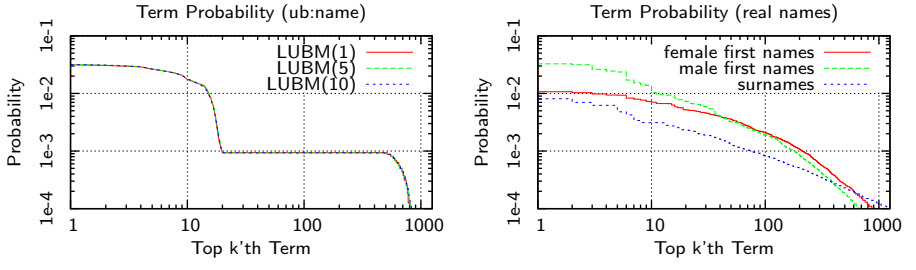

Fig. 1. (Left) Term distribution of the `ub:name` predicate. (Right) Term distribution of first names and surnames provided by the 1990 U.S. Census.

During the generation process, instances and relations are created on the basis of uniform distributions. For instance, a university contains 15 to 25 departments, each employing 7 to 10 full professors, whereas each authored 15 to 20 publications. This gives the LUBM datasets quite a complex structure with well defined connectivity properties. However, the literal values lack such sophisticated characteristics. The name of a person, for instance, is a one term literal like `"FullProfessor0"`. A fulltext search for this term would have exactly one

[3] Lehigh University Benchmark (LUBM): http://swat.cse.lehigh.edu/projects/lubm/

[4] Throughout this paper we refer to the Univ-Bench namespace using `ub`, which resolves to http://www.lehigh.edu/%7Ezhp2/2004/0401/univ-bench.owl

match for each department of each university. In addition, the `ub:name` property unfortunately is used to label all kinds of resources. Therefore, the terms of this predicate follow a mixture of uniform distributions, a highly unrealistic setting. Figure 1 shows the distribution of the `ub:name` terms (left-hand side), and, as an example, a real world distribution of first names and surnames (right-hand side). One can see that surnames and male first names are even power law distributions, as most real-world term distributions. We provide further details on first name and surname distributions in section 2.2.1.

2.2 Fulltext Content Generation

The proposed fulltext extensions modifies the LUBM dataset in two places. First, names are generated such that they follow realistic term distributions. Second, to measure performance on large text literals, fulltext content is added to the publication instances created by LUBM. To realize this extension, we add two sources of literals to the LUBM data generator, 1) the *Name Generator* adding real person names and 2) the *Document Generator* providing publication content. The enhanced datasets contain the same statements as the original LUBM datasets, extended by the `ub:firstname`, `ub:surname`, `ub:fullname`, as well as the `ub:publicationText` predicate, containing the respective content. Whereas the original LUBM dataset is referred to as LUBM(N), the fulltext extended dataset is denoted as LUBMft(N).

2.2.1 Name Generatator

The first new source of literals we add is the *Name Generator* which produces random names consisting of a first name and a surname. The generated terms follow a real distribution of names. As input for the term frequencies we use the data provided by the U.S. Census Bureau from the 1990 census[5]. This dataset provides probabilities for the top $\approx 1,200$ male, and for the top $\approx 4,300$ female first names. It also contains the top $\approx 89,000$ surnames, but due to rounding errors, only the top $\approx 19,000$ surnames have sufficiently accurate frequency information.

2.2.2 Document Generator

The second source of literals with realistic term distributions is the *Document Generator*. Since LUBM is a synthetic benchmark that produces data of any specified size, the content generation also needs to scale. Therefore, we can't directly use a real-world document collection as content. Instead, we use a Probabilistic Topic Model [21] to learn features that characterize such a collection. We then generate a synthetic set of documents which exhibit the learned real-world characteristics. This approach preserves the desired scalability feature of LUBM and at the same time ensures realistic term frequency distributions of the generated collection.

[5] Frequently Occurring First Names and Surnames From the 1990 Census: http://www.census.gov/genealogy/names/

To model a realistic topic distribution of documents, we apply *Latent Dirichlet Allocation* (LDA) [21] where a document is the result of a mixture of topics, and a topic is a distribution over words. We reuse the Matlab® implementation of the LDA model provided by the Steyvers and Griffiths [21]. As training set we employ the NIPS document collection[6], consisting of 1,740 papers, written by 2,037 authors with a total of 2,301,375 terms, out of 13,649 unique terms. From the topic distribution of documents, we derive a topic cooccurrence probability: when a topic has a high probability for a document, then it also gets a higher cooccurrence probability with all other topics of this document.

Given the topics and their cooccurrence, we first assign a dedicated topic to each department. Then, we distribute topics cooccurring with the department topic among the faculty staff of the department, according to their probabilities. Together with the department topic, the topics of the authors determine the topic mixture of a publication. This topic assignment algorithm leads to publications that cluster around departments by means of used terms: each department and author has its own specific vocabulary. This better reflects term usage among universities, departments, and authors than one global term distribution. Table 1 gives an overview of the resulting LUBMft datasets.

Table 1. Statistics of the LUBM fulltext datasets

	LUBMft(1)	LUBMft(5)	LUBMft(10)	LUBMft(50)	LUBMft(100)
Universities	1	5	10	50	100
Person Names	8,330	51,955	106,409	555,815	1,120,834
Publications	5,999	37,854	76,529	402,142	808,741
Unique Terms	17,611	28,171	32,438	36,456	36,518
Terms	6,032,320	38,061,820	76,954,636	404,365,260	813,224,336
Statements	134 k	840 k	1.7 M	8.96 M	18 M
Size (XML)	56 MB	356 MB	719 MB	3.8 GB	7.6 GB

2.3 Fulltext Test Queries

The fulltext dataset generated needs to be complemented with related queries to form a complete benchmark. The goal is to design these queries that evaluating them against the LUBMft datasets provides insights in the strengths and weaknesses of RDF stores regarding fulltext search. We deem query plan optimization an integral part of an efficient query evaluation. Therefore, we follow the same principle as LUBM where query patterns are stated in descending order, *w.r.t.* their cardinality. This gives RDF stores the maximum possible opportunity to optimize the query plan, and reduces the probability that a store just by chance executes a better query plan than another store.

Our benchmark queries are subdivided into three sets, each targeting a different area:

[6] Author-Topic Model: NIPS: http://www.datalab.uci.edu/author-topic/NIPs.htm

1. *Basic IR performance*: These queries are equivalent to classic IR queries, *i.e.,* keyword queries without any semantic relations.
2. *Semantic IR performance*: This set of queries targets at the performance of the combination of keyword queries and semantic relations. These are hybrid queries. Furthermore, the quality of integration of fulltext search into the SPARQL evaluation process is investigated.
3. *Advanced IR features*: Finally, this set of queries explores advanced IR features like boolean, phrase, or wildcard queries, score or snippet retrieval.

Queries. In the following, we present the three query sets, in which the queries get more and more complex, and where subsequent queries build on the results of previous ones. Note that some queries have variants, *e.g.* Query 1 is evaluated with different keywords to investigate the impact of those different keywords for the same dataset size. The two variants are denoted as Query 1.1 and 1.2.

All queries focus on fulltext performance, their *combination* with structured queries and their *integration* into the query evaluation process. We do not investigate the performance of the structural part of the queries. The performance of the structured query parts is sufficiently investigated by RDF benchmarks discussed in section 4.

2.3.1 Basic IR Performance

The first set of queries that we design will only contain queries that can equivalently be expressed in pure IR queries as well. Further, only those IR features are exploited that are expected to be the minimum set of features all RDF stores with fulltext search do support, namely keyword and phrase queries.

Table 2. Basic IR Queries

Q1: "All resources matching the keyword k". $k \in$ {'engineer', 'network'}
Q2: "All resources matching k_i in `ub:publicationText`". $k \in$ {'engineer', 'network'}
Q3: "All resources matching 'network' or 'engineer' in `ub:publicationText`".
Q4: "All resources matching the phrase 'network engineer' in `ub:publicationText`".
Q5: "All resources matching keyword 'smith' / having literal "Smith" in `ub:surname`".

We start with a simple one-keyword query. Its performance is of interest, since in the more complex benchmark queries the evaluation of keyword query conditions produces equivalent intermediate result sets with a proportional performance and scalability impact. We use one infrequent term, 'engineer', and one very frequent term, 'network', to investigate influence of result set cardinality. For LUBMft(1), the result set cardinality for these terms is 40 and 1,013, respectively.

In Query 2, we restrict the terms to match in the `ub:publicationText` predicate. The result set is the same, so the difference in performance compared to Query 1 is caused by this additional constraint. This query is equivalent to fielded keyword search in IR. Result sets of two keywords have to be looked

up and merged to evaluate Query 3, which is the union of both result sets. To evaluate a phrase query of the two keywords, the position information of each occurrence of both keywords have to be processed by the fulltext search engine. Query 5 finally let us compare keyword lookup performances with literal lookup, since both queries are semantically equivalent and have the same result set.

2.3.2 Semantic IR Performance

The following queries contain structural conditions to investigate their impact on the query evaluation performance. These queries show the quality of integration of IR features with general SPARQL query processing.

Table 3. Semantic IR Queries

Q6:	"All `ub:Publications` matching 'engineer' in `ub:publicationText`".
Q7:	"All `ub:Publications` and their `ub:title` matching 'engineer' in predicate `ub:publicationText`".
Q8:	"All `ub:Publications`, their `ub:titles`, and the `ub:FullProfessor` author's `ub:fullname`, matching 'engineer' in `ub:publicationText`".
Q9:	"All resources matching 'engineer' in `ub:publicationText` and all resources matching 'smith' in `ub:fullname`, being connected via the predicate `ub:publicationAuthor`".
Q10:	"All resources matching 'network' in `ub:publicationText` and all resources matching 'engineer' in `ub:publicationText`, that are both connected via `ub:publicationAuthor` to the same `ub:FullProfessor`".
Q11:	"All distinct `ub:FullProfessors` matching 'smith' in `ub:fullname` that authored both, resources matching 'network' and resources matching 'engineer' in the `ub:publicationText` property".

From Query 6 to 8, we increase the number of structural constraints of the query. The result set of the keyword condition of the query has to be joined with more and more triple patterns. This allows us to evaluate the impact of structural parts of the query. The next step is the extension to two-keyword queries, where each keyword matches a different resource in the query graph. In Query 9, both matching resources are tested to be connected via a `ub:publicationAuthor` predicate. Query 10 interconnects both matching resources via a `ub:FullProfessor` resource. Finally, we extend our query to three keywords, where the intermediate full professor additionally has to match the third keyword. This requires three result sets of the fulltext search to be joined via structural restrictions.

2.3.3 Advanced IR Features

In our final query set we investigate the existence of certain advanced IR features. In contrast to the feature evaluated by Query 1 to 5, the advanced features are not expected to be supported by all RDF stores offering fulltext search.

Table 4. Advanced IR Queries

Q12:	"All resources matching 'network' **and** 'engineer' in `ub:publicationText`".
Q13:	"All resources matching 'network', **but not** 'engineer' in `ub:publicationText`".
Q14:	"All resources matching 'network' and 'engineer' in `ub:publicationText`, both keywords appearing within a distance of at most 10 words of each other".
Q15:	"All resources matching wildcard keyword 'engineer*' in `ub:publicationText`".
Q16:	"All resources matching wildcard keyword 'engineer?' in `ub:publicationText`".
Q17:	"All resources matching fuzzy keyword 'engineer 0.8' in `ub:publicationText`".
Q18:	"All resources and their relevance to the keyword 'engineer' in predicate `ub:publicationText`".
Q19:	"All resources and a snippet matching keyword 'engineer' in predicate `ub:publicationText`".
Q20:	"The top-10 resources with their relevance to the keyword 'network' in `ub:publicationText`".
Q21:	"The resources and their relevance to 'network' in predicate `ub:publicationText`, that have a relevance of > 0.75".

We test for conjunction and negation of fulltext related conditions, for a proximity query, where both keywords have to appear in a distance of at most 10 words, and for wildcard queries. Query 17 matches all terms that have a similarity of 0.8 to the given keyword. The similarity measure is application dependent, so for different implementations, different similarity values may be used to retrieve a comparable result set size. The relevance score and a snippet is retrieved by Query 18 and 19, respectively. Some RDF stores allow to limit the results of a fulltext search to the top-k, or to all results that exceed a certain relevance score threshold. These features are tested by Query 20 and 21. If these features are not supported, then they can be mimicked by a SPARQL LIMIT or FILTER operation. In that case, a similar performance to Query 2.2 is expected.

3 Evaluation

After generating the enhanced LUBMft dataset and designing 21 queries, we now evaluate a number of well-known open-source RDF stores using our RDF fulltext benchmark. We decided for the following RDF stores (in alphabetical order): Jena [22], Sesame2 [23], Virtuoso[7], YARS [20]. Since Jena provides a large number of different backend stores, we performed preliminary tests to identify the fastest configuration, where Jena+TDB performed best. We used the following configurations, which we will further refer to using these abbreviations:

Jena: Jena 2.5.6, ARQ 2.5.0, Lucene 2.3.1, TDB 0.6.0
Sesame2: Sesame 2.2.1, NativeStore, LuceneSail (Hits-Set) 1.3.0, Lucene 2.3.2
Virtuoso: Virtuoso 5.0.9
YARS: YARS post beta 3, Lucene 1.9.1

[7] Virtuoso Open-Source Edition: http://virtuoso.openlinksw.com/wiki/main/Main/

All Java-based RDF stores employ the Lucene[8] full-featured text search engine Java library, the state-of-the-art Java implementation for Information Retrieval.

3.1 Evaluation Methodology

The evaluation is conducted over LUBMft(N) with $N \in \{1, 5, 10, 50\}$. We reused the *LUBM benchmark test tool* (UBT) in version 1.1 to perform the tests. We modified its program sequence towards the following behavior:

Since we have designed our queries in an incremental manner, subsequent queries build on findings of previous queries. This means, that the result set of an early query is sometimes the intermediate result set of a later query. Due to RDF store specific caching mechanisms and filesystem cache managed by the operating system, subsequent executions of the same query, as well as similar queries (as described above), will benefit from those "warmed up" caches. In order to surpress these side-effects, we clear the RDF store caches by restarting it for each query, and clear the filesystem cache. We then evaluate each query six times, where the first duration is considered as "uncached", while the subsequent durations are considered as "cached". This will provide us insights in the performance of each store in the two cases of insufficient and sufficient memory for caching. Queries that exceed a limit of 1,000 s are not evaluated any further. As a duration of a query we define the time that passes from parsing the query until no more results are provided by the RDF store.

For each RDF store and each LUBMft dataset, all defined 21 queries are evaluated in this manner. This whole process is repeated 5 times. In the end we have 5 evaluation times for each query referring to the "uncached" case, as well as 25 times under the presence of "warmed up" caches.

For the evaluation we use a GNU/Linux machine with a 2 GHz AMD Athlon™ 64bit Dual Core Processor, 3 GByte RAM, and a RAID 5 array. The UBT Tester runs with Java™ SE Runtime Environment 1.6.0_10 and 2 GB Memory.

3.2 Evaluation Results

In the following, we present the results of our benchmark evaluation. For easier comparison, all figures will share the same layout. Each figure depicts one benchmark query and aggregates all RDF stores. Along the x-axis the different datasets are aligned by increasing scaling factor N. The logarithmic y-axis depicts the evaluation time of a query in *milliseconds*. The bars of the RDF stores are subdivided into two parts: the whole bar represents the evaluation time with cleared caches, whereas the lower part of the bar illustrates the evaluation performance under the existence of "warmed up" caches.

3.2.1 Basic IR Performance

Figure 2 depicts the one-keyword Queries 1.1 and 1.2, which represent the basis of fulltext search. For Query 1.1, Jena is slower than Sesame2, but Jena can better

[8] Apache Lucene: http://lucene.apache.org/

deal with the larger result set of Query 1.2, and can therefore exhibit a little performance advantage. Virtuoso, which is very fast for the smallest dataset, performs worse than Jena and Sesame2 with increasing dataset size for small result sets (Query 1.1), but performs better for larger result sets (Query 1.2). For both queries, YARS reveals the worst performance. For all queries, Virtuoso has an impressive performance with "warmed up" caches. The results of Query 2.1 and 2.2 exhibit for all stores the same performance as for the Queries 1.1 and 1.2. We therefore omitted the figures of Query 2.1 and 2.2 due to space limitations.

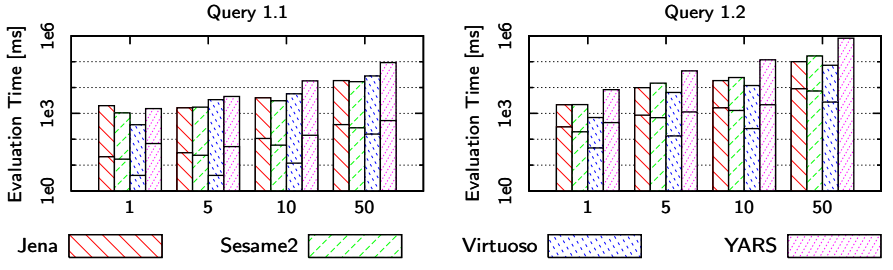

Fig. 2. Single keyword queries without predicate binding

With Query 3 we did not significantly increase result set size compared to Query 1.2 and 2.2, but we use multiple keywords in a boolean *OR* semantic. As we can see in Figure 3, all three stores that use Lucene exhibit exactly the same performance for Query 3 and 1.2. Note that Virtuoso does not support this query. Compared to Sesame2, Jena also yields a better performance for Query 4. Here, Virtuoso is even slower than Sesame, but it shows impressive response times in case of "warmed up" caches. YARS does not allow for phrase query articulation. Looking at Query 5.1 and 5.2, all RDF stores have a better performance evaluating the latter one.

Discussion. From Query 1 and 2 we can see that none of the stores has a problem with missing predicate bindings for the fultext search, nor does it gain performance when it is given. The implementations are robust in both situations.

Regarding basic IR queries, Jena is slightly faster than Sesame2. Both use Lucene as their fulltext search engine, so the overhead of the integration of the IR engine into the RDF evaluation might be a little higher in Sesame2. YARS also uses Lucene, but is much slower. This may be due to the outdated lucene version they use. Virtuoso is for all but one cases the slowest RDF store. Further, it does only support a subset of the basic IR features.

3.2.2 Semantic IR Performance

With our semantic IR queries depicted in Figure 4, we tested for the performance of fulltext search combined with structural queries. From Query 6 to 8

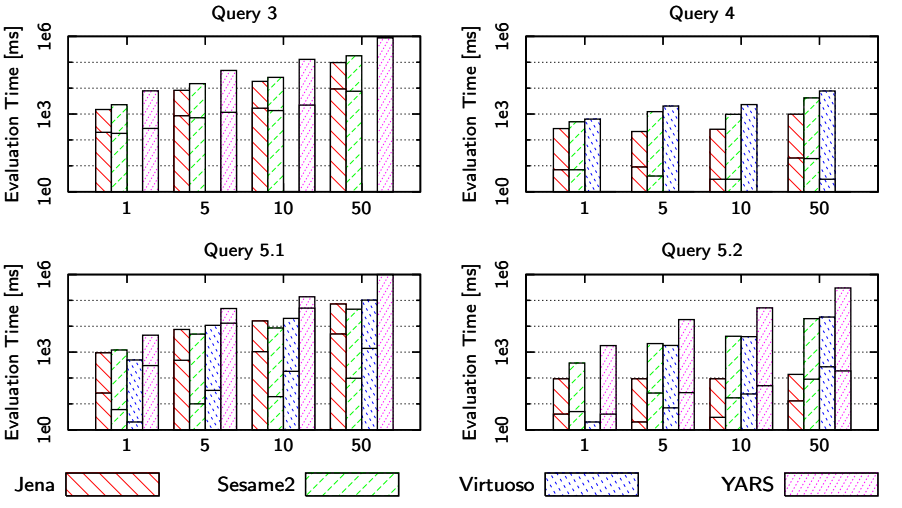

Fig. 3. Multiple-keyword, phrase, and keyword vs. literal queries

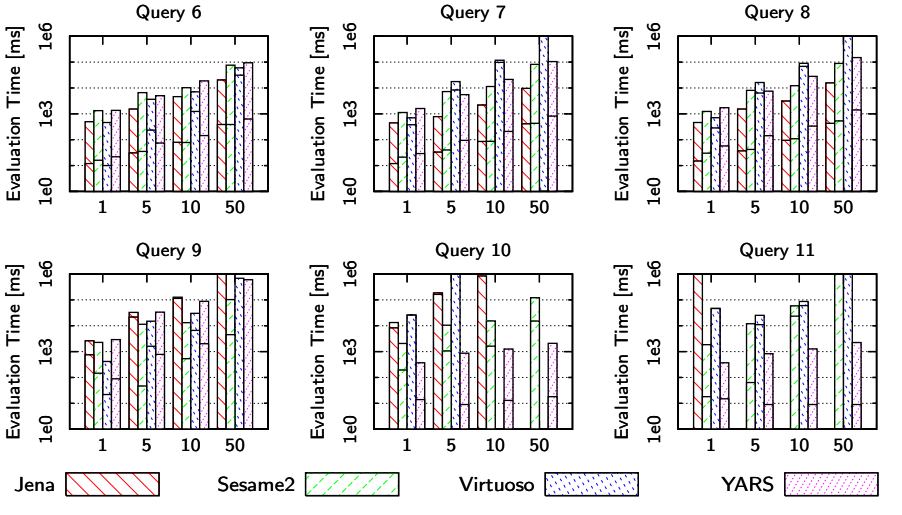

Fig. 4. Semantic IR queries combining fulltext search with structured queries

an increasing amount of structural information around the resources that match the fulltext search are retrieved. For Query 6, all stores have similar performance and scaling properties, but Jena always performs best. In Query 7 and 8, Virtuoso has significant scaling problems, while all other stores exhibit quite the same performance as for Query 6.

With multiple keywords (Query 9 to 11), Jena is completely overloaded. Virtuoso exceeds our query evaluation limit for the second smallest dataset already. Sesame2 scales up best with multiple keywords. Note that YARS is not capable of retrieving the correct results for queries with multiple fulltext searches.

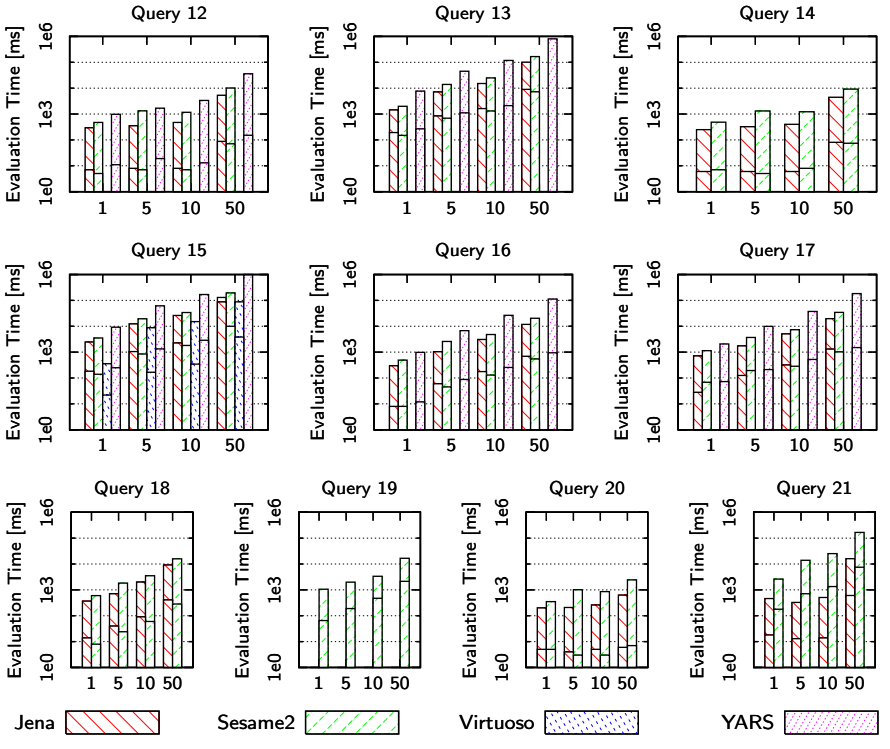

Fig. 5. Advanced IR feature queries: boolean, proximity, wildcard, and fuzzy queries, score and snippet retrieval, result and relevance score limitation queries

Discussion. By evaluating the semantic IR queries, we could recognize that Jena and Virtuoso have scalibility problems for hybrid queries in general, and in particular with those containing multiple fulltext queries assigned to different resources in the query. YARS could also not properly evaluate these queries. Further investigations on Jena showed that this is highly influenced by the very simple query optimizer provided by TDB. With an optimized query planning, Jena could potentially outperform Sesame2. This class of queries seems not to be considered by the evaluated RDF stores (except by Sesame2), however, it is an important class of queries in the field of semantic keyword queries.

3.2.3 Advanced IR Features

Finally, we tested for the advanced IR features. Virtuoso only supports one out of ten queries, but this one with the smallest response time of all RDF stores. YARS only supports a subset of five queries. Jena and Sesame2 both have similar performance, but Jena is always faster. For Query 20 one can see that the response time is quite constant for all datasets, whereas for Query 21, this only holds for Jena.

Discussion. Looking at advanced IR features, Sesame2 stands out by supporting all features. For those supported by Jena, its overall performance is better. The result set limitation feature of Query 20 perfectly works for both. The evaluation time neither grows with the dataset size nor with the correlating complete result set size. Limiting the results yields to an almost constant response time. Due to the fact that Sesame2 does not support the relevance score limitation feature of Query 21, this feature has to be achieved using a SPARQL FILTER. This, first evaluates the query completely and then filters all results that match the score limit. Jena provides this feature with its fulltext querying facility and therefore performs as expected.

Summary. For basic or advanced IR queries, Jena and Sesame2 are the best choices. Considering complex semantic IR queries, Sesame2 is the only RDF store sufficiently solving the tasks of fulltext search on RDF data. With "warmed up" caches, Virtuoso yields an impressive performance, but does only supported a small subset of queries.

4 Related Work

In software engineering domains, benchmarks are used to assess the relative performance and absolute feature-richness of a target system [24]. For this, well designed standard tasks mimic a particular type of workload. The *Full-Text Document Retrieval Benchmark* [24, ch. 8] (FTDR) generates as workload a mixture of searching and retrieving documents from the system, as well as adding documents. In contrast, our benchmark only investigates search.

In the area of relational databases, the *TEXTURE* benchmark [25] investigates performance of combining text processing with relational workload. As the FTDR Benchmark, it randomly generates queries based on certain term occurrence characteristics, whereas we intend to provide fixed queries that can be repeatably evaluated and studied for different RDF stores and hardware configurations. Similarily to our benchmark, they synthetically generate fulltext, but they only maintain one global word distribution.

In the area of RDF stores, a number of benchmarks are available. The *Berlin SPARQL Benchmark* [17] (BSBM) also generates fulltext content and person names. The SP^{2}Bench [18] uses a dataset that refers to the structure of the DBLP Computer Science Bibliography[9]. Both benchmarks pick terms from dictionaries with uniform distribution. The SP^{2}Bench and BSBM were not considered for our RDF fulltext benchmark simply due to the fact of their very recent publication. Enriching these benchmarks with real world fulltext content and fulltext queries is very much in our favor. To the best of our knowledge, the presented fulltext extension to the LUBM is the first fulltext Benchmark for Semantic Web systems.

Due to the lack of RDF fulltext benchmarks, the authors of [19] employed the RDF version of Wordnet[10] and queried for randomly selected terms. However, this dataset does not simply scale up to an arbitrary size. Similarily to

[9] DBLP Computer Science Bibliography: http://dblp.uni-trier.de/
[10] Wordnet RDF: http://www.semanticweb.org/library/

our approach, the authors of [15] generated a LUBM(50,000) dataset, where literals were generated by keywords randomly selected from a dictionary. Unfortunately, the authors did neither sufficiently report on this benchmark in order to reproduce results or reuse the benchmark, nor did they apply that benchmark on other RDF stores. Further, our benchmark provides more realistic content characteristics.

5 Conclusion

The work presented in this paper fulfills the strong need of application and RDF store developers for an RDF fulltext benchmark. With the provided dataset and hybrid queries, fulltext capabilities of any RDF store can be evaluated and compared. Our evaluation shows that on the one hand basic as well as advanced IR features are already sufficiently supported by today's RDF stores. On the other hand, it also uncovered several areas where performance needs to be improved. This holds primarily for complex hybrid queries that contain more than one fulltext search condition. The evaluation results also show that a good query plan optimization is crucial for competitive performance, as well as a tight integration of fulltext search features into this planning phase. We are convinced that the availability of a standard fulltext benchmark for RDF data stores will foster the development of more efficient hybrid query processing algorithms.

Acknowledgement

This work was supported by the European Union IST fund (Grant FP6-027750, Project NEPOMUK).

References

1. Berners-Lee, T., et al.: The Semantic Web. Scientific American 279(5) (May 2001)
2. Hildebrand, M., et al.: An analysis of search-based user interaction on the Semantic Web. Report, Centrum Wiskunde & Informatica (2007) INS-E0706, ISSN 1386-3681
3. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF (January 2008)
4. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press / Addison-Wesley (1999)
5. Garcia-Molina, H., et al.: Database Systems: The Complete Book. Prentice Hall, Englewood Cliffs (2008)
6. Chaudhuri, S., et al.: Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In: CIDR, pp. 1–12 (2005)
7. Amer-Yahia, S., et al.: Report on the db/ir panel at sigmod 2005. SIGMOD Rec. 34(4), 71–74 (2005)
8. Hristidis, V., et al.: Efficient IR-Style Keyword Search over Relational Databases. In: VLDB, pp. 850–861 (2003)

9. DeFazio, S., et al.: Integrating IR and RDBMS Using Cooperative Indexing. In: SIGIR, Seattle, Washington, USA, July 9–13, 1995, pp. 84–92. ACM Press, New York (1995)
10. Consens, M.P., et al.: XML Retrieval: DB/IR in Theory, Web in Practice. In: VLDB, University of Vienna, Austria, September 23–27, 2007, pp. 1437–1438. ACM, New York (2007)
11. Bhagdev, R., et al.: Hybrid Search: Effectively Combining Keywords and Semantic Searches. In: ESWC, Tenerife, Canary Islands, Spain, June 1-5, 2008, pp. 554–568 (2008)
12. Zhang, L., et al.: Semplore: An IR Approach to Scalable Hybrid Query of Semantic Web Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825. Springer, Heidelberg (2007)
13. Baeza-Yates, R.A., Gonnet, G.H.: Fast Text Searching for Regular Expressions or Automaton Searching on Tries. Journal of the ACM 43(6), 915–936 (1996)
14. Cho, J.: A fast regular expression indexing engine. In: Proceedings of the 18th International Conference on Data Engineering (2002)
15. Harth, A., et al.: YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 211–224. Springer, Heidelberg (2007)
16. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 158–182 (2005)
17. Bizer, C., Schultz, A.: Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints. In: Proceedings of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS 2008) (2008)
18. Schmidt, M., et al.: An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In: ISWC, October 26–30 (2008)
19. Minack, E., et al.: The Sesame LuceneSail: RDF Queries with Full-text Search. Technical Report 2008-1, NEPOMUK (February 2008)
20. Harth, A., Decker, S.: Optimized Index Structures for Querying RDF from the Web. In: Proceedings of the 3rd Latin American Web Congress. IEEE Press, Los Alamitos (2005)
21. Steyvers, M., Griffiths, T.: Probabilistic Topic Models. Lawrence Erlbaum, Mahwah (2006)
22. Carroll, J.J., et al.: Jena: Implementing the Semantic Web Recommendations. In: WWW Alternate track papers & posters, pp. 74–83. ACM, New York (2004)
23. Broekstra, J., et al.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54–68. Springer, Heidelberg (2002)
24. Gray, J. (ed.): The Benchmark Handbook For Database and Transaction Processing Systems. Morgan Kaufmann, San Francisco (1993)
25. Ercegovac, V., et al.: The TEXTURE Benchmark: Measuring Performance of Text Queries on a Relational DBMS. In: VLDB, Trondheim, Norway, pp. 313–324 (2005)