

R-Quill: Reasoning with a Billion Triples

<http://kt.abdn.ac.uk/btc>

Edward Thomas and Jeff Z. Pan

Department of Computing Science
University of Aberdeen, Aberdeen AB24 3UE, UK

Abstract. The Semantic Web is growing at a huge rate, with more sources of data being added all the time, and lightweight reasoning is believed to play a key role in dealing with large scale data sets. Semantic data are published against various schemata, which commonly contain transitive subsumption of classes and, sometimes, properties; this means that when users ask for all individuals belonging to a particular class, query engines should also return all individuals of all its direct and indirect subclasses. In this paper, we propose an optimised RDF DL reasoner R-Quill and show how R-Quill can be used to reason with the Billion Triple data set. We will also discuss further extensions of R-Quill for OWL 2 QL, OWL 2 DL and SKOS data.

1 Introduction to the Problem

The growing popularity of the semantic web has led to a large amount of data being published to the World Wide Web in RDF format. Much of this data is structured, using some dialect of RDF Schema or OWL. To query data which uses RDF Schema or OWL, inferencing must be performed on the data before querying will be complete. For example, if one queries the DBpedia YAGO ontology¹ to retrieve all instances of the YAGO class <http://dbpedia.org/class/yago/Book106410904>, then all instances of its over 1500 sub-classes should be returned. It is expected light weight reasoners should play a key role here; however, even for light weight reasoners, there are still pressing challenges:

Query Complexity As the above example shows, input queries could be re-written as thousands of queries. Light weight reasoners might only be able to answer some of the re-written queries, but not all of them. For examples, if we use some SPARQL endpoints (such as Virtuoso²) to query instances of <http://dbpedia.org/class/yago/Book106410904>, the results are rather incomplete; e.g., instances of its sub-class <http://dbpedia.org/class/yago/BaseballBooks> are not returned.

Data Size One approach is to cache the reasoning results, storing them in the repository. This could, however, significantly increase the size of the ontology.

¹ <http://www.mpi-inf.mpg.de/yago-naga/yago/>

² <http://dbpedia.org/sparql>

Let us take the above YAGO class http://dbpedia.org/class/yago/Book_106410904 as an example, assuming each of its subclass has 1000 instances: if all the subclasses are direct subclasses, then we need to add over 1.5 million class assertions into the ontology; if the class hierarchy is deep, then we would need to add much more class assertions, so as to make the instance-of relation for the intermediate classes explicit as well. When a data set is getting huge, some useful optimisations might no longer as effective.

Data Loading Time The web of data are increasing dramatically. For example, there were only about 1 billion RDF triple in the Linked Open Data set early 2007, while the number is close to 5 billions as of September 2009. To deal with such large amount of increasing data, it could be very expensive, if one has to recompute the cached reasoning result every time new data arrive.

In this paper, we will present our R-Quill reasoner, which uses a low cost solution to inferred query answering over RDF DL ontologies, without having to use high performance clustered computer architectures (such as that used by MaRVIN) or store redundant data. The remainder of the report is organised as follows. Section 2 highlights the innovations of our R-Quill reasoner. Section 3 introduces R-Quill, explaining its features, optimisations and reporting some preliminary evaluation results over the YAGO ontology. Section 4 concludes the report and discusses some ongoing extensions of R-Quill.

2 Innovation

In this section we will look at the two most common solutions to this problem, and discuss the advantages and drawbacks of each, we will then compare that with the approach we have taken in R-Quill. We will be using the YAGO ontology³ as an example in this section. YAGO was created at Max Planck Institute at Saarland University and it combines Wordnet and DBpedia (and by extension, Wikipedia) to create a rich hierarchy of concepts populated with instances (the instances generally corresponding to Wikipedia articles). We have chosen it for examination here because the large number of TBox axioms and the total size of the ontology make reasoning over this with current lightweight reasoners difficult. Furthermore, it is important due to its position as a central hub of the Linking Open Data project.

2.1 Current Approaches

Realisation Realisation is the term used to describe the process of applying a set of inference rules to a knowledge base, and using these rules to generate all of the additional required axioms in the knowledge base. For example, the rule used to perform transitive closure of a `subClassOf` relationship is $subClassOf(x, y), subClassOf(y, z) \rightarrow subClassOf(x, z)$. This rule looks

³ YAGO: <http://www.mpi-inf.mpg.de/yago-naga/yago/>

for any pair of subClassOf relationships which connect to each other (via y), and it infers a new subClassOf relationship between the two outlying classes (x and z) To perform instance realisation over a knowledge base, a role such as $type(a, x), subClassOf(x, y) \rightarrow type(a, y)$ must also be employed. Taken together these two rules mean that it is possible to determine every sub-class of a particular class and every instance of a particular class by performing trivial queries over the realised knowledge base. A similar approach may be taken with properties where property hierarchies (sub-property relationships) can exist in a knowledge representation.

The drawbacks of using this approach come when there is a very large taxonomy of classes. In the case of a schema containing C classes, an average depth of D in the class hierarchy and N concept membership axioms, the approximate number of axioms A_r which must be stored to fully realise the knowledge base can be calculated by the formula $A_r = CD + DN$. In the case of the YAGO ontology, populated with instance data from DBPedia, there are 224,193 classes, with an average class depth of 8.8, and a total of 4,121,336 concept membership axioms, giving an approximate total of 38,240,654 axioms in the realised knowledge base. Actually, this total is even higher, since most of the instances in the YAGO ontology exist in leaf classes, the actual total number of inferred axioms in the ontology is 86,364,091.

Although in this case, triple stores are capable of handling this number of axioms, the time taken to run the inference rules, as well as the additional workload querying a larger knowledge base, such as that used in the Billion Triple Challenge causes this to be a far from ideal solution.

Query Rewriting Query rewriting is a technique which expands a query based on the schema of a knowledge base, in order to include subsumed classes in a result set. We will here assume a query rewriting algorithm with equivalent behavior to the PerfectRef algorithm employed in the DL-Lite family.

This algorithm examines every class inclusion in the hierarchy and creates a new query for every concept which is subsumed by the concept named in the atomic query. It iterates over the query and the TBox, generating an ever larger set of queries until the algorithm is exhausted. Using this algorithm creates a union of queries - one query for every class subsumed by the class being investigated.

Executing this algorithm to find instances of classes which are leaf nodes in a class hierarchy results in only a single query to be executed, but executing it on classes near to the root of a hierarchy may result in hundreds or thousands of individual queries which each has to be executed, and the union of the results is taken to form a complete result set. Looking at the worst case scenario, under the YAGO ontology, a query for the root element in the ontology `yago:Entity` would result in a set of 214,193 SQL queries which would individually need to be evaluated on the database. The expansion of a conjunctive query which includes many query atoms suffers from polynomial complexity with respect to the size of the ontology TBox and the number of atoms in the query. A worst case

conjunctive query over an ontology such as YAGO could result in a set of queries which could number hundreds of millions of individual database queries. In the case of querying all but the most trivial of datasets currently available on the Semantic Web, executing this large union of queries would result in unacceptable performance.

2.2 Our Approach

R-Quill uses an alternative method, where query expansion is performed within the database itself, taking advantage of the b-tree indices used within databases, and using this to perform queries containing the closure of the `subClassOf` and `subPropertyOf` axioms within a single query, containing no unions.

Because both the RDF hierarchy and the index structures in a database are trees, we can use the database index to almost directly represent the class and property hierarchies present in the source dataset. By joining the query across this structure to tables representing instances of classes and properties from the source data, we can perform sound and complete queries against the RDFS (or OWL 2 QL) datastore at unprecedented speeds for a disk based, non-clustered, RDFS store.

3 R-Quill

In this section, we present the features and optimisations used in R-Quill, as well as some preliminary evaluation results over the YAGO ontology.

3.1 Features

Full support for RDFS `subClassOf`, `subPropertyOf`, and type within queries

Unlike basic RDF triple stores, R-Quill supports the correct semantic behaviour for these meta-properties. If a knowledge base asserts that some class A is subsumed by some class B, then a query for all instances of class A should also include all instances of class B. The same behaviour also applies when one property subsumes another.

Support for OWL 2 QL inference within queries

OWL 2 QL's query rewriting rules require that property atoms in the query which contain a non-distinguished variable, should be replaced by a query over class representing the existential restriction of that property (or of the negation of the property where the non-distinguished variable exists in the object position in the query). R-Quill fully supports queries over OWL 2 QL knowledge sources with correct adherence to OWL 2 QL semantics.

Conjunctive query answering through a SPARQL endpoint

R-Quill supports answering of conjunctive queries over the SPARQL standard.

Support for queries which cannot be translated into a conjunctive query is currently limited, but improvements to the query rewriting algorithm and to the underlying structure of the database should allow more arbitrary SPARQL queries in future releases of R-Quill.

Support for several repositories

R-Quill allows information to be stored in separate named repositories. These are completely separate at the database level, and so can be used to store multiple versions of a single dataset, without clashes between the data. Each repository has its own SPARQL endpoint for queries.

Multi-threaded reasoners for RDFS and OWL 2 QL for improved performance on multicore systems

The RDFS and R-Quill reasoners in R-Quill are fully multi-threaded, allowing optimum performance on modern multi-core systems. Informal testing with the Billion Triple dataset, showed an average increase in loading speed of 2.6 times when using a multi-threaded design over a single threaded design, on the machine described below (four cores).

Minimal storage of redundant data

The data stored in the R-Quill database contains minimal redundant data. Every triple in the ABox of a dataset results in the storage of a single row of data in our database. A complex data schema does not increase the size of the stored ABox. Using a rule engine and performing ABox realisation over a standard triple store will result in a much larger stored representation of the input dataset.

Atomic Query Rewriting

Our query rewriting algorithm, and our database representation of hierarchical data structures means that every SPARQL query that can be performed in R-Quill is rewritten into exactly one (atomic) SQL query. No query expansion is required to get sound and complete results.

3.2 Optimisation

The most costly part of the query is the join between the hierarchy table and the instance table. Querying very deep hierarchies gives a large degree of overlap between these two tables (possibly thousands of matching rows in each) which reduces the performance of the query significantly. It can also be seen in many deep hierarchies, that the vast majority of instances occur on or near to leaf nodes of a hierarchy, meaning that many concepts are completely unpopulated. For example, in the YAGO/DBPedia ontology which is used for the evaluation, there are a total of 214,193 concepts, of which more than half are unpopulated.

Since, for the purposes of instance retrieval in the query, we are not interested in checking concepts or properties which have no instances, we provide an optimisation to the database schema. In this revised schema, we create new tables containing the abbreviated hierarchies containing only those classes and properties which are actually populated in the dataset. This can then be used in queries to reduce the number of rows involved in the join between the hierarchical structure and the instance tables.

To maintain this table as the repository is updated, triggers can be used on insert and delete operations on the instance tables to check that the concept that belongs to any newly asserted individual is present in the populated hierarchy table, and that if the last instance of a concept is deleted, then it is removed.

3.3 Implementation and Evaluation

We have implemented our algorithms in two systems. The first is in the Semantic Approximation and query components of TrOWL and ONTOSEARCH2. We have also written a streaming reasoner as a TrOWL component which can convert and load RDF-DL or normalised OWL 2 QL files into the database, while also traversing, generating and updating the class and property hierarchies in-situ as new axioms are loaded. Query rewriting is performed in the R-Quill query component of TrOWL.

All queries were run on an Dell PowerEdge server, with a Quad core 1.86GHz Xeon CPU, 4GiB RAM, and 2x500GB disk in a RAID 1 mirror configuration. The server was configured with Ubuntu Linux, version 9.04, with the default installation of Apache Tomcat 6, the Java virtual machine, and Postgres 8.3. No performance tuning was performed on any aspect of the system, except to configure Java to use up to 2GiB of RAM (using the `-Xmx2048m` option), and to give Postgres access to up to 2GiB of RAM to cache.

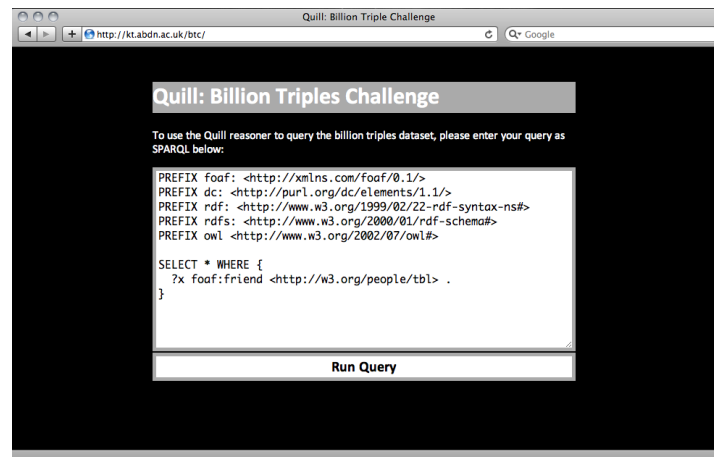


Fig. 1. The R-Quill BTC SPARQL interface

We loaded the Billion Triples Challenge dataset in a single run using the streaming RDFS reasoner. The total time required to load the dataset was 22 hours, 58 minutes. This equates to a triple throughput of 18,974 triples per second. Comparing this with the results of various plain RDF triple stores in

the Berlin Benchmark⁴ shows that, although the different hardware makes direct comparisons difficult, our system offers competitive loading performance compared to other RDF triple stores.

4 Conclusions

In this paper we have presented a novel method for storing and querying over hierarchical data on the Semantic Web. The approach allows reasoning to be performed in a database using an index, giving extremely good performance for conjunctive queries. The result of this is that there is no need to increase data complexity by using rules to expand the knowledge base to include inferred axioms, nor is there any need to perform query expansion which increases query complexity.

The R-Quill system shows that by exploiting the characteristics of relational databases, the advances made in data storage and retrieval performance can be fully used for semantic web knowledge bases. The performance both for loading and querying data is comparable to simple, non-inferring, RDF triple stores; and the amount of disk space required to store the resulting database is also comparable as redundant information stored is minimal (particularly compared to rule inference systems). By making use of a rewriting algorithm which results in a single SQL query (with no sub queries, unions, or views involved in the query), query answering is both sound and complete, as well as giving better performance than other RDFS or OWL 2 QL inference engines.

⁴ Berlin: <http://www4.wiwiw.fu-berlin.de/bizer/BerlinSPARQLBenchmark/V1/results/index.html>