Benchmarking RDF Production Tools

Martin Svihla and Ivan Jelinek

Czech Technical University in Prague,
Karlovo namesti 13, Praha 2, Czech republic,
{svihlm1, jelinek}@fel.cvut.cz,
WWW home page: http://webing.felk.cvut.cz

Abstract. Since a big part of web content is stored in relational databases (RDB) there are several approaches for generating of semantic web metadata from RDB. In our previous work we designed a novel approach for RDB to RDF data transformation. This paper describes experimental comparison of our system with several means of RDF production. We benchmarked both systems for the RDB to RDF transformation and native RDF repositories. The test results show a good performance of our system but also bring a new look at the effectiveness of the RDF production.

1 Introduction

It is widely accepted fact that the growth of the semantic web is dependent on the mass creation of metadata that will cover current web resources. Since the most of web content is backed by relational databases our previous work is focused on the transformation of relational data into RDF metadata, which is based on mapping between a relational database schema and existing RDF-S ontology. We have recently proposed *METAmorphoses* [1] – a new data transformation model based on two layers. This model is designed with regard to its performance, robustness and usability. Our work stands on the theoretical foundations of the semantic web technologies, but we also took into the account practical issues while developing the formal model. In this paper we briefly introduce our approach and compare our system with several other approaches for the RDB to RDF transformation as well as with some native RDF repositories built over relational databases. Our approach appears to be the fastest solution for RDF production between selected systems but results shows much more general conclusions – it appears that a well designed RDF production directly from RDB can be faster than querying the native RDF repositories.

2 METAmorphoses

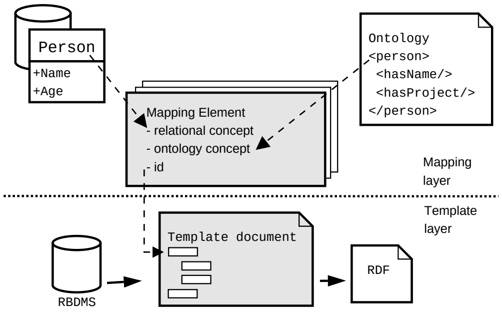
The *METAmorphoses* processor[1] is a data transformation tool developed in our previous work [1]. It transforms relational data to RDF according to mapping

[1] Available at http://metamorphoses.sourceforge.net/

from a relational schema into an existing ontology. The transformation process has two steps: (i) mapping from a particular RDB schema into an existing RDF-S ontology and (ii) creating RDF documents from relational data based on the mapping from the first step. Thus the model has two layers (figure 1). The mapping between a database schema and ontology consists of mapping elements and is processed in the mapping layer. A mapping element addresses relational concepts by SQL queries. The mapping is used in the template layer, which processes templates – XML-based documents for querying RDB. Since templates have a form of RDF and uses SQL from mapping elements to fetch data from RDB, they can be considered as a RDF view to RDB. METAmorphoses supports all RDF and RDF-S features and is relational complete. We designed the system so that it uses no RDF API, which is not neccessary for a data transformation and can slow a performance of the system. This fact together with the template user interface instead of RDF query language is a novel contribution to the RDB to RDF data transformation. The system is implemented in Java language.

The scope of this paper does not allow us to describe the system in detail. The complete description is available in [1].

Fig. 1. Two-layer data transformation in METAmorphoses



3 Experiment Overview

3.1 Compared Systems

We compared 5 different systems in our experiments – 3 tools for the RDB to RDF transformation (*METAmorphoses*, *D2RQ* and *SquirrelRDF*) and 2 native RDF repositories with RDB back-end (*Jena* and *Sesame1*). Moreover, we performed 2 different tasks with D2RQ and Jena in the most of tests – we queried dataset with both SPARQL and graph API.

METAmorphoses v0.2.5 is briefly described above. To perform tests we created a schema mapping between the relational schema of the experimental dataset and RDF-S ontology and then queried the dataset using our template documents as queries.

D2RQ v0.5 [2] is a plug-in for the Jena [4], which uses mappings to rewrite Jena API calls to SQL queries and passes query results as RDF triples up to the higher layers of the Jena framework. Using D2RQ mapping it is possible to access relational database as a virtual RDF graph via classical Jena API. This way the relational database can be queried by SPARQL [8] or `find(s p o)` functions and a result is an RDF. When testing D2RQ, we performed two separate experiments: one with `find(s p o)` functions and another with SPARQL. We run D2RQ in *Jena v2.5.1* in these tests.

SquirrelRDF [3] is a tool which allows relational databases to be queried using SPARQL. It provides a tool that creates just a rough mapping for a database schema (its *the naïve RDB to RDF mapping*, described in [9], which does not consider an ontology) and a set of different SPARQL interfaces. Result of the SPARQL query over RDB is RDF. SquirrelRDF requires *Jena v2.4* and we used its API to perform SPARQL queries in our experiments.

Jena v2.5.1 (persistent DB model) [4] is a Java framework for building semantic web applications. It provides a programmatic environment for RDF, RDF-S, SPARQL and includes a rule-based inference engine. Jena can also store RDF data persistently in relational databases. We stored testing dataset in the persistent storage (backed by MySQL RDBMS) and then performed exactly the same experiments as in the case of D2RQ.

Sesame v1.2.6 (persistent DB model) [6] is an open source Java framework for storing, querying and reasoning with RDF and RDF-S. It can be used as a database for RDF and RDF-S, or as a Java library for applications that need to work with RDF internally. Sesame provides also relational storage for RDF data (so called *RDBMS-Sail*). We uploaded our testing RDF dataset to the Sesame persistent datastore (backed by MySQL RDBMS) and queried it with SeRQL (the internal query language of Sesame) in our experiments.

3.2 Experiment Methodology

To compare the tools listed above we used micro-benchmarks. The measured aspect was a time of an RDF production on a given query. Each test task consisted of *(i) preliminary phase*, where the source data, query engine and query were prepared and *(ii) measured phase*, where the query was executed and resulting RDF was written to standard output in the RDF/XML syntax. We decided to measure also RDF output because the tested aspect is the RDF production. According to granularity of the benchmarking tool (10ms) and the very short time of a query execution, we executed each query 100 times in a measured phase of each task. A warm-up was executed before each measured task in order to avoid JVM performance unbalance. A test consisted of the 5 same tasks executed in a row and its result was an arithmetic mean computed of the 5 task times.

3.3 Testing Dataset

The dataset used for the benchmarks is an XML dump of the DBLP computer science bibliography [7]. XML was converted into an SQL database dump and into an RDF representation[2]. The relational version of the dataset consists of 6 tables (`InProceeding`, `Person`, `Proceeding`, `Publisher`, `RelationPersonInProceeding`, `Series`) without indexes and contains 881,876 records. These relational data were stored in a MySQL database and used while testing *METAmorphoses*, *D2RQ* and *SquirrelRDF*. The RDF representation of DBLP contains 1,608,344 statements and it was loaded into relational backends of *Jena2* and *Sesame1* to test them. To obtain more granular data, we created the tables `Proceeding500` and `Proceeding1500`, which contains the first 500 and 1500 records respectively from the table `Proceeding`. These data were also added to the RDF version of the dataset.

3.4 Testing Environment

The tests ran on Intel Pentium M processor 1400MHz with 1536 MB of RAM. The operating system was Linux (i386) with kernel version 2.6.12. Java Virtual Machine was the one implementated by Sun Microsystems Inc., version 1.5.0_01-b08. The RDBMS for storing data was MySQL server 5.0.30-Debian_1. All tests were performed within a simple Java benchmarking framework called JBench[3], which provides an easy way to compare Java algorithms for speed. The CPU timer based on the native JVM profiling API was used to obtain more accurate times. This timer reports the actual CPU time spent executing code in the test case thread rather than the *wall-clock* time that is affected by CPU load. The granularity of the timer was 10ms.

4 Experiments and Results

Testing queries are described in SPARQL formal terminology even their form vary between systems. In case of SPARQL and SeRQL queries we used *CONSTRUCT* form so that the result was a graph - as well as in the case of METAmorphoses templates or Jena Graph API. To compare various aspects of the RDF production, we divided our tests into three groups. In these groups we tested RDF production according to a *(i)* result size, *(ii)* query graph pattern complexity and *(iii)* query condition complexity.

[2] Thanks to Richard Cyganiak (FU Berlin) for providing us the converted datasets.

[3] Available at http://www.yoda.arachsys.com/java/jbench/

4.1 Experiments with the Result Size

In this test set we performed very simple query, based on the following general graph pattern:

$$(?s\ \ < rdf : type > < particular\_RDF - S\_class\_URI >)$$

We applied this query on RDF-S classes with different amount of RDF individuals and we compared the behaviour of the tools according to the size of resulting RDF graph. We performed 5 tests in this group - we stepwise selected all resources with type `Series`, `Publisher`, `Proceeding500`, `Proceeding1500` and `Proceeding`. The specific SPARQL queries for these tests with number of triples in the resulting graphs are in the table 1, results are listed in the table 2.

Table 1. Tests with the result size: queries

Test no.	**Query**	**Result triples**
1.1	CONSTRUCT * WHERE {?r rdf:type d:Series.}	24
1.2	CONSTRUCT * WHERE {?r rdf:type d:Publisher.}	64
1.3	CONSTRUCT * WHERE {?r rdf:type d:Proceeding500.}	500
1.4	CONSTRUCT * WHERE {?r rdf:type d:Proceeding1500.}	1500
1.5	CONSTRUCT * WHERE {?r rdf:type d:Proceeding.}	3007

Table 2. Tests with the result size: results (times in ms)

	Test no. (number of result triples)				
System	**1.1** (20)	**1.2** (64)	**1.3** (500)	**1.4** (1500)	**1.5** (3007)
METAmorphoses	*84*	*230*	*1840*	*5692*	13124
SquirrelRDF	830	1314	5180	16228	42504
D2RQ SPARQL	522	1332	7730	25530	45704
Jena SPARQL	482	1444	8296	27478	49968
D2RQL Graph API	366	1134	6434	20922	38253
Jena Graph API	368	1028	6902	22874	42588
Sesame1 SeRQL	194	423	2144	6624	*12826*

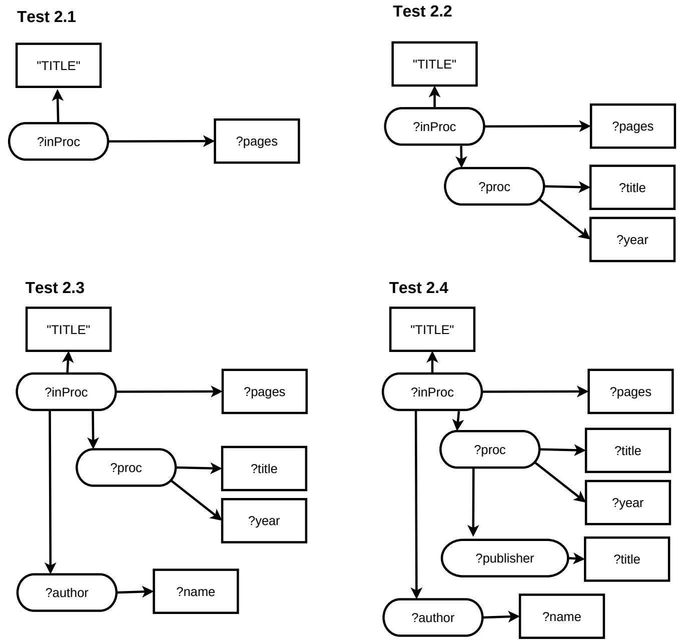
4.2 Experiments with the Graph Pattern Complexity

Test queries from this group consist of one graph pattern matching condition, which identifies exactly one RDF resource:

$$(?s\ \ < my\_ontology : hasTitle >\ \ "TITLE"\text{^^}xsd : string)$$

These queries differ in amount of resources and literals linked by graph patterns in the query. The size of resulting RDF graph does not differ too much in these queries so that we can compare tools according to complexity of the query graph pattern. The graph patterns are depicted on the figure 2, test results are listed in the table 3.

Fig. 2. Tests with the graph pattern complexity: graph patterns



4.3 Experiments with the Query Condition Complexity

The tests in the third test set have a very simple graph pattern and they reffer to individuals from only one RDF-S class. These tests differ in number and type of conditions in the query. We performed these tests only with 5 systems – we ommited Jena and D2RQ graph API because this API does not allow straightforward queries with more conditions.

In SPARQL, there are two ways how to restrict possible solutions of a query: graph pattern matching and constraining values. This test set contains 4 tests that combines these conditions in various ways. The size of a resulting RDF is very small so that tests are focused on the query algorithm performance.

Table 3. Tests with the graph pattern complexity: results (times in ms)

	Test no. (number of result triples)			
System	**2.1** (2)	**2.2** (4)	**2.3** (6)	**2.4** (8)
METAmorphoses	*28*	*76*	*110*	*124*
SquirrelRDF	640	678	768	808
D2RQ SPARQL	252	426	674	850
Jena SPARQL	212	360	456	552
D2RQL Graph API	106	224	434	506
Jena Graph API	94	150	204	262
Sesame1 SeRQL	100	198	272	324

The test queries are depicted on the figure 3. The first query (test 3.1) contains a single graph pattern matching condition (similar to queries from the second test set) and the resulting graph contains 8 triples. The second query (test 3.2) adds one graph pattern matching condition to the first query and fetches 2 triples from the dataset. The query in the test 3.3 uses a condition with constraining value to obtain the same result as the test 3.2. The last query (test 3.4) combines conditions from test 3.1 and 3.3. The test times are in the table 4.

Fig. 3. Tests with the query condition complexity: graph patterns

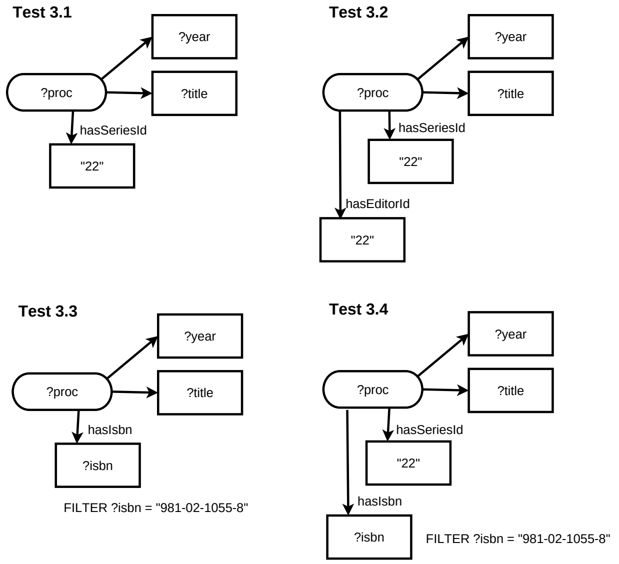


Table 4. Tests with the query condition complexity: results (times in ms)

	Test no. (number of result triples)			
System	**3.1** (8)	**3.2** (2)	**3.3** (2)	**3.4** (2)
METAmorphoses	*78*	*36*	*30*	*32*
SquirrelRDF	636	582	9670	598
D2RQ SPARQL	618	336	17480	360
Jena SPARQL	544	240	30794	232
Sesame1 SeRQL	238	124	110	126

5 Discussion on Results

The test results show that our approach (METAmorphoses) was the fastest one almost in all tests (except test 1.5, where Sesame1 had slightly better performace).

In the first test set all systems have approximatelly linear computation performace. This relation between the result size and performance is illustrated in the table 5, which contains average times for producing 1 triple (100 times). It is obvious that these times are almost the same for each system, but vary between systems. Interesting is also the test 1.1, where the *time-for-one-triple* index is slightly higher for the most systems. We reason that this is caused by a *starting phase* of the query execution, which does not depend on the result size and is obvious at the query with a small resulting RDF (24 triples). The relatively shortest starting phase appears with METAmorphoses, the longest is with SquirrelRDF (more than 2 times higher than in other tests). Considering this starting phase there is no point to compute the time-for-one-triple index in the second and third test set – a resulting RDF is very small in these tests (2-8 triples).

The METAmorphoses was the fastest system also in the second test set. It keeps its high performance and small growth of the test time with the growing graph pattern complexity. The METAmorphoses has the best performance also in all tests in the third test set 4. Sesame1 is a little bit slower but similarity of its results with those of METAmorphoses is interesting. On the other hand, other three systems are much more slower. It is obvious in the test 3.3, where result times are very high. This is probably caused with non-optimized constraint value handling in the Jena SPARQL query engine, which is used by all these systems.

An interesting observation is that all systems based on Jena (SquirrelRDF, D2RQ and Jena itself) have very similar results, especially in the first and third test set. We explain this by the same algorithms for a resulting RDF graph composition (in the first test set) and SPARQL query execution (in the third test set). This means all solutions build above Jena shares all its advantages and disadvantages and are limited by its performance. Sesame1 and METAmorphoses

Table 5. Time (in ms) for producing one triple (100 times) - based on the first test set

	Test no. (number of result triples)				
System	**1.1** (20)	**1.2** (64)	**1.3** (500)	**1.4** (1500)	**1.5** (3007)
METAmorphoses	3,5	2,67	3,68	3,79	4,36
SquirrelRDF	34,58	15,28	10,36	10,82	14,14
D2RQ SPARQL	21,75	15,49	15,46	17,02	15,2
Jena SPARQL	20,08	16,79	16,59	18,32	16,62
D2RQL Graph API	15,25	13,19	12,87	13,95	12,72
Jena Graph API	15,33	11,95	13,8	15,25	14,16
Sesame1 SeRQL	8,08	4,92	4,29	4,42	4,27

had considerably different (and usually much better) test performance. Sesame1 is obviously optimized for querying big amounts of data and METAmorphoses was designed to be a *high performance* data transformation tool. According to the test results, we can say that our concepts implemented in METAmorphoses shows higher performance compared to other tested data transformation systems (D2RQ and SquirrelRDF). Our assumption that using RDF API is performance limitation was correct – our system is faster than those with RDF API.

METAmorphoses is also faster than tested native RDF persistent storages (persistent DB model in Jena and Sesame1). This is very interesting point. We proved that if one needs just to publish relational data in RDF, there is no need to migrate RDB to RDF repository and query this repository. On-the-fly data transformation (using METAmorphoses) can be done faster than queries over native RDF repository.

We did not measured RAM footprint of tested systems. However, METAmorphoses does not build RDF graph in a memory (it is a stream data transformation processor) thus its memory consumption does not depend on a size of a resulting RDF. All other tested systems first create resulting graph in memory and then serialize it, which means that their RAM footprint depends on a size of resulting RDF graph.

6 Related Work

There are not many similar comparison experiments for RDF production tools because the lack of a common query language and access method make benchmarking RDF stores a time consuming task (as mentioned in [5]). However, several attemps are described in [5], [11] or [10]. Due to different metodologies and tested systems it is very difficult to compare results, but our performance comparison can be considered as one of the most complex due to the number of tested systems and performed tests, too.

7 Conclusion

In this work we performed three test sets focused on computational performance to compare our ideas implemented in METAmorphoses with other RDB to RDF transformation tools (D2RQ and SquirrelRDF) and native RDF stores with RDB back-end (Jena and Sesame1). METAmorphoses had the best performance in the most tests (12 out of 13) and also other performance aspects discused in the section 5 were better. We proved that our system of data transformation has higher performance than other similar tools as well as native RDF repositories.

The main contribution of this paper is that we showed the on-the fly data transformation can be faster than queries over native RDF repository – thus it is not neccessary to migrate relational data to RDF repositories in order to publish them as RDF.

Acknowledgements

This research has been supported by MSMT under research program no. 6840770014 and by the grant of the Czech Grant Agency no. 201/06/0648.

References

1. Svihla, M., Jelinek I.: The Database to RDF Mapping Model for an Easy Semantic Extending of Dynamic Web Sites. Proceedings of IADIS International Conference WWW/Internet. Lisbon, Portugal, 2005.
2. Seaborne, A. and Bizer, C.: D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. Proceedings of the 3rd International Semantic Web Conference (ISWC2004), Japan, 2004.
3. Steer, D.: SquirrelRDF. http://jena.sourceforge.net/SquirrelRDF/
4. Jeremy J. et al.: Jena: implementing the semantic web recommendations.Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. New York, USA, 2004.
5. Harth, A. and Decker, S.: Optimized index structures for querying RDF from the Web. Proceedings of LA-WEB, 2005.
6. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, 2002.
7. Ley, M.: DBLP Bibliography. http://www.informatik.uni-trier.de/ ley/db/
8. Prud'hommeaux, E. and Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation. February 2005.
9. Beckett, D. and Grant, J.: Semantic Web Scalability and Storage: Mapping Semantic Web Data with RDBMSes. SWAD-Europe deliverable. February 2003.
10. Streatfield, M. and Glaser, H.: Report on Summer Internship Work For the AKT Project: Benchmarking RDF Triplestores. Technical Report. Electronics and Computer Science, University of Southampton. November 2005.
11. Cyganiak, R.: Benchmarking D2RQ v0.2. Technical Report. Freie Universitt Berlin, Germany. June 2004.