Can RDB2RDF Tools Feasibily Expose Large Science Archives for Data Integration?

Alasdair J.G. Gray[1], Norman Gray[2,3], and Iadh Ounis[1]

[1] Computing Science, University of Glasgow, Glasgow, UK
[2] Physics and Astronomy, University of Leicester, Leicester, UK
[3] Physics and Astronomy, University of Glasgow, Glasgow, UK

Abstract. Many science archive centres publish very large volumes of image, simulation, and experiment data. In order to integrate and analyse the available data, scientists need to be able to (i) identify and locate all the data relevant to their work; (ii) understand the multiple heterogeneous data models in which the data is published; and (iii) interpret and process the data they retrieve. RDF has been shown to be a generally successful framework within which to perform such data integration work. It can be equally successful in the context of scientific data, if it is demonstrably practical to expose that data as RDF.

In this paper we investigate the capabilities of RDF to enable the integration of scientific data sources. Specifically, we discuss the suitability of SPARQL for expressing scientific queries, and the performance of several triple stores and RDB2RDF tools for executing queries over a moderately sized sample of a large astronomical data set. We found that more research and improvements are required into SPARQL and RDB2RDF tools to efficiently expose existing science archives for data integration.

1 Introduction

Due to the ease with which information can be published on the Internet, scientists potentially have access to more data than at any time in history. The Internet makes it viable for researchers to publish the results of their experiments for others to use as text or binary files, XML documents, or relational databases. There are also numerous archive centres which gather information from various sources and make it available from one place, often as a query endpoint to a relational database due to the size and structure of the data.

For example, consider the situation in astronomy of the Virtual Observatory (VO), which is being developed by the many national and international projects that form the International Virtual Observatory Alliance (IVOA)[1]. One of the goals of the VO is to allow an astronomer to seamlessly access any piece of astronomical data, regardless of whether it has been archived in one of the many archive centres (e.g. ROE[2], ESAC[3]), generated by a simulation

[1] `http://www.ivoa.net` accessed 8 December 2008.
[2] `http://www.roe.ac.uk/ifa/wfau/` accessed 10 December 2008.
[3] `http://www.esa.int/SPECIALS/ESAC/` accessed 10 December 2008.

L. Aroyo et al. (Eds.): ESWC 2009, LNCS 5554, pp. 491–505, 2009.

(e.g. VIRGO[4], MODEST[5]), or published by an individual as the result of some analysis (e.g. NVO publish your own data interface[6]).

However, for astronomy, as with any other domain, there is no single data model of the world that is suitable for storing and querying all the available data—each data archive, simulation, etc., uses its own, tailor-made model that is a compromise between the storage and querying requirements for the particular data. Therefore, in order for an astronomer to be able to access, extract, and process the data relevant to their needs, they must be able to: (i) locate those data sources which potentially contain information relevant to their research; (ii) understand the data model of each relevant data source in order to compose a query to extract the required data; and (iii) interpret and process the result data with either bespoke or general purpose visualisation and analysis software.

The first of these problems is commonly addressed by using a registry service which stores details of the available resources. Users query the registry service by describing their information need which is *matched* to find those resources that are potentially relevant [7]. For example, in the case of the VO an XML Schema and access protocols are used to discover resources [15]. These registry services return a list of potentially relevant data sources, which the user must then understand in order to extract the data relevant to their research.

One solution to the second problem is to follow a traditional data integration approach where the data sources are queried through a global data model [9]. However, generally there is not a common data model that is capable of accessing all of the available data with the granularity required by all users. One of the benefits often stated for RDF is the ease with which data can be integrated from distributed RDF sources. This, in association with a peer-to-peer data integration approach—whereby the data models of the sources are exposed as RDF and linked to each other by alignment mappings [6]—should allow a user to query using a data model with which they are familiar and exploit the mappings in order to retrieve the data that is relevant to their research.

The third problem is reliant on a viable solution to the second. As such, there is currently limited software support within the VO for the analysis of data from multiple sources.

A first step to enable the peer-to-peer data integration approach is the ability to expose existing science archives as *virtual* RDF graphs. RDB2RDF tools provide such functionality for relational databases [13]. In this work, we consider the feasibility of using these tools to query existing relational databases as SPARQL endpoints, in the context of an astronomical data archive. Specifically, we investigate whether: (i) SPARQL can be used to express the types of query required in scientific applications, and (ii) the performance of RDF triple stores and RDB2RDF tools can meet the requirements of the large data sets encountered in the scientific domain.

[4] `http://www.virgo.dur.ac.uk/` accessed 10 December 2008.
[5] `http://manybody.org/` accessed 10 December 2008.
[6] `http://www.us-vo.org/publish.cfm` accessed 10 December 2008.

Section 2 presents the issues of integrating distributed scientific data and states how RDF can overcome these. Section 3 describes a typical scientific data set, its structure and the types of queries posed over it. The section then discusses the capabilities and limitations of SPARQL. In Section 4 we describe the performance experiments and discuss the results. Section 5 presents related work and our conclusions are given in Section 6.

2 Integrating Scientific Data

The aim of a data integration system is to seamlessly allow a user to access multiple autonomous data resources as if they were a single, coherent data source. This is generally achieved by (i) agreeing on a mediated *global* schema, (ii) providing a wrapper for each data source, and (iii) generating mappings that relate the source schema to the global schema (see [9] for an overview). However, there are a few well known problems with creating a data integration system.

First, there must be agreement on a single global schema. Within the astronomy setting of the VO, such a global schema does not exist, although over the last few years there have been attempts to create such a global schema [12]. Reaching this limited consensus has proven to be a time consuming process, and the resulting model still does not achieve the goal of a full global model: it falls short of being expressive enough to capture the existing data source models.

Second, insisting that all data sources map their data to a single global model leads to a loss of data precision. This is because the mappings must translate between the granularity of the data source and the global model, which generally limits the available data.

To evade these problems, we offer a more "peer-to-peer" approach. In such a system, the data resources would publish their schema in some common modelling language and produce mappings to one or more different models in the system. These mappings might be

Direct: if there is a close association between two (probably specialised) data models then equivalence mappings can be used, thus maintaining the level of expressible data and reducing the loss in data precision;

Indirect: using some widely understood data model which may be at a different level of granularity to link terms through a super-class relationship;

Multi-stepped: using some chain of mappings, although this risks some loss of fidelity.

An advantage of this approach is that the user can use a data model they are familiar with to express their information needs as a query. The system is able to convert the query into a set of queries over the linked data sources using the web of mappings that exist between the various data sources. Providing a suitable direct mapping exists, the user does not suffer from data granularity problems.

The peer-to-peer data integration approach outlined requires a common modelling language for expressing all of the available data source models. Since we consider that data sources can be published either as files, XML documents,

or relational databases, we propose using RDF as the common data modelling language and leveraging the Semantic Web reasoning and mapping techniques available in RDFS and some limited OWL concepts to generate the mappings between the data models. It remains an open question whether such mappings are flexible enough, we do not cover this. A pre-condition for this form of data integration system is a wrapper for exposing an existing data source as an RDF model over which SPARQL queries can be posed.

Two approaches are possible. The first would be to replicate the data contained in a relational source into an RDF triple store such as Jena [14] or Sesame [4], aka Extract-Transform-Load (ETL) or an RDF dump. However, this would suffer from all the usual problems of replicating data, e.g. maintaining multiple copies of the data in different data models and data freshness. The second would be to use a wrapper that can expose the existing relational database schema as an RDF model and perform *on-the-fly* or *query-driven* conversions to translate SPARQL queries over the exposed RDF model into SQL queries over the relational schema. D2RQ [2] and SquirrelRDF [18] are two prototype RDB2RDF systems capable of performing these on-the-fly translations. This paper considers the viability of these data integration wrapper approaches in the context of scientific data.

3 Scientific Data Set

Various studies compare the performance of the storage and querying capabilities of RDF triple stores and RDB2RDF tools in non-scientific applications [11,19,3,17]. While these benchmarks have considered data sets with a significantly large number of triples (up to 100 million triples [3]), they have not considered data sets with similar structures to those of a science archive, nor have they considered queries using complex selection conditions, commonly found in scientific applications. In order to enable the integration framework outlined in Section 2, it is important to investigate the viability of using RDB2RDF tools to expose existing science archives. We conducted performance tests using real science queries over an existing archive.

3.1 The SuperCOSMOS Science Archive

The SuperCOSMOS Science Archive (SSA[7]) [10] contains data extracted from scans of photographic Schmidt plates, e.g. the position of objects, their luminosity, etc. The archive is about 4 terabytes in size and contains data on nearly 6.4 billion object detections covering the whole sky in three wavebands. This data is stored in a relational database on a Microsoft SQLServer 2000 DBMS located at the Royal Observatory, Edinburgh. It was chosen as the data set for the performance measures for several reasons.

First, it is a typical example of an astronomical data archive in that the information is stored in a relational database and contains a vast amount of data.

[7] `http://surveys.roe.ac.uk/ssa/` accessed 9 December 2008.

Table 1. Details of the relations contained in the Personal SuperCOSMOS Science Archive, providing the number of attributes in each relation and the number of rows

Relation	No. Attributes	No. Rows
CrossneighboursEDR	5	129,474
CurrIsoCal	5	3,535
Detection	46	169,558
Field	9	2,695
FieldSystem	3	4
HtmTable	6	263,200
HtmTableId	2	10
IsoCalVersion	4	3
JunkDetection	7	6,125
Neighbours	3	585,560
Plate	152	3,535
PrevIsoCal	6	3,535
Source	57	90,947
Survey	24	9

The data is accessible for SQL query from a web page and also through various VO data access protocols. Second, the schema of the database was designed so that it could answer 20 scientifically significant queries which are representative of the types of query that would be expressed to the production system, *cf. G*ray *et al.* [8]. These queries, which are provided both as English statements and SQL, can be used to test the expressive power of SPARQL, and as the basis for the performance measures to test the retrieval speed of the various query engines. Finally, a subset of the data in the SSA is available for download, as the Personal SuperCOSMOS Science Archive (PSSA). The PSSA contains the data relating to the area of the sky given by ($184 < RA/deg < 186$, $-1.25 < Dec/deg < 1.25$) which is roughly 0.1% of the data contained in the full SSA. This provides a suitably sized dataset for running performance measures, without becoming a trivial toy example, and is comparable in size to other benchmark tests conducted over RDF triple stores [19].

3.2 Schema

The schema of the PSSA consists of 14 relations, the names of the relations together with the number of attributes in each relation and the number of data records are given in Table 1. The relations contain details of the photographic plates, the detection of objects on the plates, and the survey in which the photographic plate was captured. Of note, are the large number of attributes in the Detection, Plate, Source, and Survey relations. The large number of attributes are required to capture the parameters used when making the measurements, and is a distinguishing characteristic of scientific data sets not found in the existing benchmark datasets.

The schema of the PSSA also contains two views, ReliableGalaxies and ReliableStars. Both views select a subset of data from the Source relation by applying selection criteria that define ranges for certain attributes which when satisfied mean that the object can be reliably identified as a galaxy or a star respectively.

3.3 Example Scientific Queries

The design of the schema of the SSA was guided by 20 scientifically significant queries. The queries involve complex selection criteria for describing the required data as well as joins, including some that involve an external data source (the

Provide the positions and magnitudes of stars for which the magnitudes from the two R band surveys differ by more than 3 magnitudes.

```
SELECT TOP 30 ra, dec, sCorMagB, sCorMagR1, sCorMagR2, sCorMagI
FROM   Source
WHERE ellipR1 < 0.33 AND qualR1 < 2048 AND
        (prfstatR1 > -5 AND prfstatR1 < +5) AND
        ellipR2 < 0.33 AND qualR2 < 2048 AND
        (prfstatR2 > -5 AND prfstatR2 < +5) AND
        ABS (sCorMagR1-sCorMagR2) > 3 AND
        scorMagR1 > 0.0 AND sCorMagR2 > 0.0
ORDER BY ra,dec
```

Fig. 1. Query 2 from the 20 example SSA queries expressed as an English statement and as an SQL query over the SSA schema

Table 2. Features used in the 20 example queries

Feature	Contained in Query
Absolute value	2, 12, 15, 17, 19
Arithmetic in body	1, 2, 3, 4, 5, 7, 9, 12, 13, 15, 16, 17, 18, 19, 20
Arithmetic in head	7, 8, 9, 12, 13
Between	3, 5, 8, 12, 13, 15, 18, 19, 20
Count	7, 8, 9, 18
External data join	16, 17, 19
Group By	7, 8, 9
Join	12, 13, 15, 16, 17, 19
Log	4, 16
Negated sub-query	18, 20

Feature	Contained in Query
Ordering	1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20
Power	4, 9, 16
Rounding	7
Self-join	14, 15
Server function	10, 11
Square root	4, 16
Trigonometry	8, 12
Type casting	7, 8, 12
Views	1, 3, 4, 5, 7, 8, 9, 10, 12, 15, 16, 17, 18, 19

Sloan Digital Sky Survey [1]). As an example of the types of constructs used in the queries, a selection query is shown in Fig. 1.

An analysis of the SQL features used in the 20 example queries is given in Table 2. The analysis shows a significant number of the queries involve arithmetic, mathematical, and trigonometric functions. This is likely to be, in part, a result of the selected test domain, astronomy. However, similar scientific domains will have similar features in their queries, and aggregate queries are a common SQL query form across a wide variety of domains.

Note that queries 10 and 11 involve a server defined function. This function identifies nearby objects by calculating their distance on the celestial sphere. Since the server function extends the capabilities of the query engine, for simplicity we decided to omit the queries for the purpose of the performance tests.

3.4 SPARQL Capabilities

Fig. 2 provides query 2 of the 20 SSA queries as a SPARQL query. Since there is no separate schema for an RDF graph, each of the returned variables must be explicitly bound in the query. Also, some of the syntactic shorthands found in SQL such as between and abs are missing from the SPARQL language, although their functionality can be provided using suitable filter conditions. The filter operations allow for basic comparisons of attributes. They can express numeric range expressions, e.g. $?attr > 35$; simple arithmetic operations, e.g. $?attr_1 * attr_2$;

```
PREFIX vocab: <http://surveys.roe.ac.uk/pssa/vocab/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?ra, ?dec, ?sCorMagB, ?sCorMagR1, ?sCorMagR2, ?sCorMagI
WHERE {
        ?source a vocab:source ; vocab:source_ra ?ra ; vocab:source_decl ?decl ;
                vocab:source_sCorMagB ?sCorMagB ; vocab:source_ellipR1 ?ellipR1 ;
                vocab:source_sCorMagR1 ?sCorMagR1 ; vocab:source_ellipR2 ?ellipR2 ;
                vocab:source_sCorMagR2 ?sCorMagR2 ; vocab:source_qualR1 ?qualR1 ;
                vocab:source_sCorMagI ?sCorMagI ; vocab:source_qualR2 ?qualR2 ;
                vocab:source_prfStatR1 ?prfStatR1 ; vocab:source_prfStatR2 ?prfStatR2 .
        FILTER (?ellipR1 < 0.33)
        FILTER (?qualR1 < 2048)
        FILTER (?prfStatR1 > -5 && ?prfStatR1 < 5)
        FILTER (?ellipR2 < 0.33)
        FILTER (?qualR2 < 2048)
        FILTER (?prfStatR2 > -5 && ?prfStatR2 < 5)
        FILTER (?sCorMagR1 > 0.0 && ?sCorMagR2 > 0.0 &&
                ((?sCorMagR1 - ?sCorMagR2 > 3) || (?sCorMagR1 - ?sCorMagR2 < -3)))
```

Fig. 2. Query 2 (see Fig. 1) from the 20 example SSA queries expressed as a SPARQL query over a suitable RDF version of the SSA

and string matching can be performed, e.g. regex($?attr$, "$\wedge$ $Star$"). However, filter operations do not have the capabilities to perform mathematical functions such as logarithms, nor are they able to compute trigonometric functions.

There are also no group by or aggregate operators, which would also be an issue for a larger number of domain applications, e.g. for expressing business logic [16]. While it is possible to extend the functionality of SPARQL by using external functions to compute operators like aggregates and mathematical functions, this would have an impact on the performance of the query engines since the data must be transferred to the external function. When considering the size of the data sets found in scientific domains—several terabytes—this is not a feasible option due to the network delays of transporting the data.

SPARQL does not meet the requirements for a scientific query language. Of the 18 queries, which do not require a server function, only 9 are expressible in SPARQL: queries 1, 2, 3, 5, 6, 14, 15, 17, and 19. This is due to the lack of aggregate functions and to the large number of mathematical and trigonometric functions required in scientific queries.

4 Performance Test

The experimental hypothesis considered is

> RDF can be used to integrate data from existing scientific archives.

To test the hypothesis, we conducted experiments to investigate the feasibility of exposing an existing science archive as RDF. Specifically, we investigated whether

1. Large quantities of scientific data can be stored and queried as RDF;
2. SPARQL queries posed over an RDF model can be converted into queries over a currently deployed relational database.

To investigate these issues, we used the PSSA and the example queries used in its design.

Note that in Section 3.4 we identified that 9 of the example queries were expressible in SPARQL. For the experiments we were only able to use five of these queries; 1, 2, 3, 5, and 6. Queries 14 and 15 were not used as their execution did not complete with the RDB2RDF tools; this was most likely due to the self-join in these queries. Queries 17 and 19 were not used as they rely on additional data stored in the Sloan Digital Sky Survey [1].

4.1 Compared Systems

We compared the performance of five query engines: a relational database management system to provide a benchmark performance, two RDB2RDF systems, and two RDF triple stores which used a precomputed RDF version of the PSSA data set[8]. Details of the systems, the specific version used and any configuration required are outlined below.

MySQL v5.1.25 is an open source relational database management system. As a base case for the experiment we used the time to execute the SQL versions of the queries over the relational PSSA. The MySQL database was used in its distribution state, i.e. no changes were made to the configuration. The PSSA data was loaded using the downloadable scripts. In particular, note that no indexes were created other than the automatically created primary key indexes. The database was accessed using the MySQL JDBC connector v3.1.12.

D2RQ v0.5.2 [2] is a tool for mapping existing relational data to an RDF model. It provides mechanisms for automatically exposing a relational database as an RDF view, and a variety of mechanisms for accessing the relational data through the RDF model, including offering a SPARQL query endpoint. D2RQ relies on Jena v2.5.1 for providing the RDF model abstraction and access mechanisms. The PSSA database was hosted in the MySQL database used for the base case. The script for generating an RDF model of the relational data was used. The output of the script was edited so that the relations without primary keys, e.g. CrossneighboursEDR, could generate a unique URI to identify each row.

SquirrelRDF v0.1 [18] is a tool which allows an existing relational database to be queried using SPARQL. This is achieved by automatically generating a mapping file that exposes the relational schema as an RDF view. As with the D2RQ mappings, the SquirrelRDF automatically generated model needed to be edited for some of the relations so that each row of the relation could be identified with a unique URI. SquirrelRDF relies on Jena v2.4 for providing the RDF model abstraction and access mechanisms. The PSSA database was hosted in the MySQL database used for the base case.

[8] The RDF version of the PSSA was generated using D2RQ.

Jena v2.5.6 [14] provides a Java API for storing and accessing RDF data. To provide persistent storage for RDF data, Jena relies on a database system. However, there are a multitude of ways to configure Jena to work with a database system. After some initial experimentation, we opted for SDB[9] v1.1 as the mechanism to interface between Jena and the MySQL database system. Jena using SDB was configured to use the recommended settings, i.e. layout 2 with the InnoDB database engine. The RDF version of the PSSA was loaded using a Jena model and a chunk size of 200,000. Note that Jena with SDB uses a database system to store RDF triples. It did not access the relational version of the PSSA.

Sesame v2.1.3 [4] also provides a Java API for storing and accessing RDF data. Sesame provides a variety of mechanisms for storing RDF data: in a "MemoryStore" where small temporary RDF stores reside in memory data structures, in a "NativeStore" which stores data on disc using indexed data structures, or using a database system. We used the recommended NativeStore for the experiments.

Note that MySQL, Jena, and Sesame are mature software products whereas SDB is semi-mature, and D2RQ and SquirrelRDF are prototype systems.

4.2 Experiment Design

To compare the query engines of the systems, a Java application was used to measure the time taken to execute a query and provide the result set. Each run took two configuration parameters, the system to be tested and the query to be executed, and consisted of three parts. The *setup* phase installed and configured the desired test system, loaded the appropriate format of the PSSA, ran a warm-up query, and created an executable version of the test query. The *experiment* phase measured the time taken to execute the query by getting the current time in milliseconds before and after the query execution step in the Java application. Finally, the *checking* phase ensured that the result set was of the expected size and computed the difference between the start and end timestamps.

4.3 Experimental Environment

The performance measures were conducted on a cluster consisting of 8 quad core Intel Xeon 2.4GHz 64bit nodes. Each node has 4GB of RAM and 100GB of disc space. Java version 1.6.0_06 for the 64 bit architecture was used. Each repetition of the experiment was executed on a node in isolation. A cluster was used to allow separate repetitions to take place in parallel. This was essential due to the time taken to load in the data set, see Section 4.5 for more details.

4.4 Results

Table 3 presents the performance results for the five example SSA queries, expressible and executable as SPARQL queries, against each of the systems tested. The results presented are an average of ten runs for each query, together with their standard deviation.

[9] `http://jena.hpl.hp.com/wiki/SDB` accessed 19 September 2008.

Table 3. Average query execution times over ten runs for the five SSA queries

	MySQL (ms)		D2RQ (ms)		SquirrelRDF (ms)		Jena (ms)		Sesame (ms)	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Query 1	34	0.5	352	17.2	613	13.5	3,450	775.5	39	24.2
Query 2	38	1.0	5,339	36.5	21,492	259.7	485,932	169,800.6	83	12.0
Query 3	33	1.2	2,733	34.4	837	11.2	7,229	2,549.8	69	26.4
Query 5	34	2.5	4,090	43.0	1,307	10.0	17,793	8,849.5	65	13.4
Query 6	1	0.4	7,468	224.5	19,984	87.8	372,561	60,204.8	56	32.7

The MySQL database accessed through JDBC markedly outperformed all of the other systems on all of the queries, except for Sesame's performance on query 1, where the MySQL system still performed better. Sesame markedly outperformed all of the other SPARQL query engines. D2RQ outperformed SquirrelRDF on queries 1, 2, and 6.

All of the systems performed well with query 1, and for the four SPARQL query engines this was their best performing query. This is due to query 1 containing one selection condition over the view ReliableGalaxies (see below for more details about the effects of the views), although the condition does involve a multiplication which is why it is not the best performing query for MySQL.

With the exception of D2RQ, all the systems had their worst performance with query 2, and it was still one of the worst queries for D2RQ. This is probably a result of the number of variables returned, a total of 6, and the fact that the entire Source relation must be considered. There is no clear reason why D2RQ should perform worst with query 6 instead.

Query 6 is the simplest of the queries: it does not involving any arithmetic operations in either the head or the body of the query, nor any views. This explains the dramatic difference in speed in the MySQL execution of the query when compared to the other queries.

The SPARQL query engines perform well on the queries involving views: queries 1, 3, and 5. This is because both the RDB2RDF systems and the RDF triple stores benefit from the relational views[10]. The RDB2RDF systems benefit because they can treat the view as if it is a relation in the schema, and thus the underlying MySQL database system computes the view conditions for them. The triple stores benefit because the view has been stored as triples in the RDF data and can be directly referenced in the SPARQL queries, rather than being computed within the SPARQL query execution. This means that the queries involving the views do not need to consider so much data. The ReliableGalaxies view of the Source relation contains 1,242 rows while the ReliableStars view of the Source relation contains 5,123. Indeed, the size of these views accounts for the performance difference between queries 1 and 3, and query 5 which involve ReliableGalaxies and ReliableStars respectively, when compared to queries 2 and 6, which involve the Source relation containing 90,947 rows.

To investigate the effects of the relational views, we ran the experiments for queries 1, 3, and 5, over D2RQ and Sesame without using the relational views in the SPARQL queries, i.e. we converted the view definitions into the SPARQL

[10] The views were computed and represented as triples in the RDF version of the PSSA.

Table 4. Average query execution times over ten runs when the views are not used

	D2RQ (ms)		Sesame (ms)	
	Mean	SD	Mean	SD
Query 1	39,374	1,651.2	88	23.9
Query 3	40,021	1,803.1	122	73.7
Query 5	153,392	5,739.7	128	55.1

query. The results are displayed in Table 4. As expected, both systems took longer to execute the queries: a doubling in the execution time for the Sesame query engine, and at least 20 times slower for the D2RQ query engine. The scale of increase for the D2RQ query execution, demonstrates (i) the effects of query optimisation that the relational database management system is able to perform, and (ii) the translated query provided by D2RQ cannot be optimised by the DBMS (see Section 4.5 for more details).

4.5 Discussion

The experiments were conducted to investigate two issues relating to whether RDF makes a suitable data model for integrating scientific data. We will now discuss how well these objectives were achieved and the limitations of our study.

RDF as a Data Model for Scientific Data. The query performance of the RDF triple stores, when no precomputed version of the views was involved, were at least twice as slow as the performance of the relational database system. This is largely due to query optimisation research for SPARQL still being in its infancy.

A second issue is the time taken to load the data. Although this was not formally measured in this study, the MySQL database typically loaded in 8-10 seconds while Sesame took 2-3 hours and Jena was in the region of 30-35 hours. The difference in bulk loading mechanisms is very striking from a user perspective. For MySQL, batch files containing tabbed separated data were used. For Jena, a "chunk" size needed to be set to alter the number of triples read in for each stage of processing. This was a memory management issue, although varying the chunk size appeared to have little effect since most of the time was spent indexing the data. The Sesame system just read in the file of triples and managed itself. Again, indexing the data took most of the time. However, the loading time is a one-off penalty for which some delay is acceptable.

When dealing with domains containing large quantities of structured data, such as scientific applications, the RDF triple stores are currently unable to compete with the performance of relational database systems. This is because currently RDF triple stores are unable to exploit the structure of the data.

Mapping SPARQL queries to Relational Data. Both of the RDB2RDF systems performed poorly in the experiments, with query times being at least ten times slower than the database system, and at least four times slower than Sesame. However, the long execution times of the RDB2RDF systems is down to

the way in which queries are translated—as shown by the log files—which prevents exploitation of the database system's capabilities to process large amounts of structured data and to optimise its query processing.

In both systems, a SPARQL query is translated into multiple queries to the database in order to retrieve the rows of the required relation. Additionally, the generated SQL queries involve a join for each bound variable in the SPARQL query. This often results in a large number of self-joins in the generated query since multiple attributes from the same relation need to be retrieved. This affects the performance of the underlying database system since for each self-join it must replicate the relation and the query optimiser cannot recognise that the query involves attributes from the same row in the relation. Once the RDB2RDF systems had retrieved all of the rows from the relations involved in the query, they then proceeded to perform comparisons to check the query conditions, no comparisons were "pushed-down" to the SQL query.

Limitations of the Study. The experiments focused on one scientific data archive. This meant that the data was structured, and was biased towards numeric data. This played to the advantages of the relational data model, but it is typical of the data required in certain application areas.

Only select-project queries were considered in the experiments, they did not involve any joins, although they did involve foreign key references. If queries 14 or 15 were to be executed, then a suitable index would be required to perform the self-join in a reasonable time period. This would benefit both the relational database and the RDB2RDF systems.

Both of the RDB2RDF tools considered relied upon Jena for providing their RDF modelling capabilities. The experiments showed that Jena did not perform well, and there are similar patterns in the execution times in these systems to the Jena execution times: the systems performed well on queries 1 and 3, moderately on query 5, and perform badly on query 2 and 6. However, this is most likely due to the use of views in the queries since Jena is not used for querying and storing data in the RDB2RDF tools.

There is a another class of RDB2RDF systems that offer a SPARQL query endpoint, but these have not been considered in this study. These are existing relational database management systems that have been extended to accept SPARQL queries, e.g. OpenLink's Virtuoso system [5]. These database system expose an RDF model of their relational data, and accept SPARQL queries over that model. The SPARQL queries are processed in the same way as a SQL query, i.e. they are parsed directly into a set of executable operations over the internal data structures of the database system. Thus, they can exploit the query optimiser of the database system. However, these systems were not considered in this study since existing legacy science archives are hosted on deployed DBMS without support for RDF and would require external tools to support SPARQL queries.

Summary. Current RDB2RDF systems are not capable of providing the query execution performance required to implement a scientific data integration system based on the RDF model. However, the considerable difference in the query

execution times of D2RQ when the relational views were used, compared to when the relational views were not used, highlights the improvements that the underlying database system can provide when it is able to optimise the query execution for the underlying views. It is likely that with more work on query translation, suitable mechanisms for translating queries could be developed. These mechanisms should focus on exploiting the underlying database system's capabilities to optimise queries and process large quantities of structure data, e.g. pushing the selection conditions to the underlying database system.

5 Related Work

Several benchmark datasets for testing the performance of RDF triple stores have been developed and there is a community effort to catalogue these[11]. Generally, the benchmarks have considered models which contain far fewer attributes per relation than the SSA schema, and have only tried to simulate the real world. The focus of these benchmarks has been on the size of the data and query throughput rather than the complexity of the queries. Below we detail those benchmarks of most relevance to the current study.

The SP2B SPARQL performance benchmark [17] and the Berlin SPARQL Benchmark (BSBM) [3] both aim to test the SPARQL query engines of RDF triple stores. The SP2B uses bibliographic data from DBLP[12] as its test data set, while the BSBM benchmark considers eCommerce as its subject area. Both benchmarks allow for the creation of arbitrary sized data sets, although the number of attributes for any given class is lower than the numbers found in the SSA. Additionally, the queries used in the benchmarks are more simplistic than those found in the scientific domain, with few or only simple selection conditions.

The THALIA testbed is specifically aimed at benchmarking RDB2RDF tools [11]. The data set represents the university course catalogues of 45 computer science departments around the world. This limits the overall size of the data. While a variety of challenging queries have been considered, e.g. exploiting structural heterogeneity in the data, the complexity of the selection conditions are more limited than the example queries for the SSA, with a filter containing at most one condition and a maximum of two filter conditions in their most complex query.

Svihla and Jelinek [19] ran experiments to investigate the performance of their METAmorphoses processor in comparison to RDF triple stores, and RDB2RDF tools. Their experiments were conducted over a copy of the DBLP bibliographic database of comparable size to the PSSA. The experiments considered carefully constructed queries with increasing complexity. However, the most complex query considered consisted of a graph pattern with 4 attributes and one filter expression. This is far simpler than the queries found in the scientific queries considered for our experiments. The METAmorphoses processor was not included in our experiments because it does not provide a SPARQL query point.

[11] `http://esw.w3.org/topic/RdfStoreBenchmarking` accessed 10 December 2008.
[12] `http://www.informatik.uni-trier.de/~ley/db/` accessed 10 December 2008.

6 Conclusions

This paper has shown

> It is *currently unfeasible* to use RDB2RDF tools to expose large science archives for data integration.

This is due to two main issues which have been demonstrated in this paper.

The first problem highlighted in this study is the inability to express scientific queries in SPARQL—only 9 of the 18 standard queries could be expressed. Currently SPARQL only has support for basic arithmetic operators. However, in scientific domains such as astronomy there is a requirement for more advanced mathematical operations such as trigonometric functions. It is likely that as SPARQL matures and is used in more domains such functions will be added as standard features.

The second problem investigated in this study was the performance of current RDB2RDF tools—which enable existing relational databases to be exposed as RDF. The experiments showed that RDB2RDF tools were markedly slower than accessing the data using either RDF triple stores or the underlying relational model. A large amount of the poor performance of the RDB2RDF systems can be accounted for by (i) the translated queries not exploiting the strengths of the relational model or the query optimiser, and (ii) the selection conditions being computed in the RDB2RDF systems rather than being pushed down to the database.

This study has shown that more research and development on appropriate query translation and optimisation techniques for RDB2RDF is required. With such work, it should become possible to use RDB2RDF tools to expose existing legacy databases as RDF models which can then be used within a data integration framework.

Acknowledgements

This research has been funded by the EPSRC grant number EP/E01142X/1.

This research has made use of data obtained from the SuperCOSMOS Science Archive, prepared and hosted by the Wide Field Astronomy Unit, Institute for Astronomy, University of Edinburgh, which is funded by the UK Particle Physics and Astronomy Research Council.

References

1. Adelman-McCarthy, J.K., et al.: The fourth data release of the sloan digital sky survey. ApJSup 162(1), 38–48 (2006)
2. Bizer, C., Cyganiak, R.: D2RQ – lessons learned. In: W3C Workshop on RDF Access to Relational Databases, Cambridge, MA, USA (October 2007)
3. Bizer, C., Schultz, A.: Berlin SPARQL benchmark. Technical report, Free University of Berlin (September 17, 2008)

4. Broekstra, J., Kampman, A., van Harmelen, F.: 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)
5. Erling, O., Mikhailov, I.: RDF support in the Virtuoso DBMS. In: CSSW 2007, Leipzig, Germany, pp. 59–68 (September 2007)
6. Euzenat, J., Shvaiko, P.: Ontology matching, 1st edn. Springer, Heidelberg (2007)
7. Galanis, L., Wang, Y., Jeffery, S.R., Dewitt, D.J.: Locating data sources in large distributed systems. In: VLDB 2003, pp. 874–885, Berlin, Germany (September 2003)
8. Gray, J., Szalay, A.S., Thakar, A., et al.: Data mining the SDSS skyserver database. The Computing Research Repository (CoRR) (February 2002)
9. Halevy, A., Rajaraman, A., Ordille, J.: Data integration: The teenage years. In: VLDB 32, Seoul, Korea, pp. 9–16 (September 2006)
10. Hambly, N., Read, M., Mann, B., et al.: The SuperCOSMOS science archive. In: ADASS XIII, San Francisco,CA, USA, pp. 137–140 (2003)
11. Hammer, J., Stonebraker, M., Topsakal, O.: THALIA: Test harness for the assessment of legacy information integration approaches. In: ICDE 2005, Tokyo, Japan, pp. 485–486 (April 2005)
12. Louys, M., Richards, A., Bonnarel, F., et al.: Data model for astronomical dataset characterisation. Recommendation, IVOA (November 8, 2007)
13. Malhotra, A.: Progress report from the RDB2RDF XG. In: Poster and Demo Session, ISWC 2008, Karlsruhe, Germany (October 2008)
14. McBride, B.: Jena: A semantic web toolkit. IEEE Internet Computing 6(6), 55–59 (2002)
15. Plante, R., Benson, K., Graham, M., et al.: VOResource: An XML encoding schema for resource metadata. Recommendation, IVOA (February 22, 2008)
16. Pöss, M., Floyd, C.: New TPC benchmarks for decision support and web commerce. SIGMOD Record 29(4), 64–71 (2000)
17. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL performance benchmark. Technical report, Albert-Ludwigs-Universität Freiburg (April 28, 2008)
18. Seaborne, A., Steer, D., Williams, S.: SQL-RDF. In: W3C Workshop on RDF Access to Relational Databases, Cambridge, MA, USA (October 2007)
19. Svihla, M., Jelinek, I.: Benchmarking RDF production tools. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 700–709. Springer, Heidelberg (2007)