Europeana RDF Store Report

The results of qualitative and quantitative study of existing RDF stores in the context of Europeana

co-funded by the European Union

The project is co-funded by the European Union, through the ***e*Content*plus*** programme

http://ec.europa.eu/econtentplus

Österreichische Nationalbibliothek

EuropeanaConnect is coordinated by the Austrian National Library

Europeana RDF Store Report

Bernhard Haslhofer, Elaheh Momeni, Bernhard Schandl, and Stefan Zander

Research Group Multimedia Information Systems
Faculty of Computer Science
University of Vienna

March 8, 2011

Abstract

Expressing data in RDF is one of the principles to be considered when making data available as Linked Data on the Web. This can be achieved using RDF-wrappers for existing (relational) data stores or by using RDF stores as data repositories. The latter requires special RDF storage solutions, many of which are available today. Organizations often have difficulties to decide which solution they should adopt because comprehensive comparisons of existing RDF stores are hardly available and experiences w.r.t performance and scalability are still missing. In this report, we summarize the results of qualitative and quantitative study we carried out on existing RDF stores in the context of the European Digital Library project. We give a detailed overview on existing RDF store solutions, analyze their functional and non-functional features, summarize the outcomes of other, previously carried out studies, and conduct a Linked-Data oriented performance evaluation on a subset of existing triple stores w.r.t to load and query time. The results of this study show that certain RDF stores, such as OpenLink Virtuoso or 4Store, can deal with the Europeana data volume and answer those SPARQL queries that are relevant for exposing Europeana metadata as Linked Data in an acceptable time-range.

1 Introduction

Linked Data has gained momentum [12] and became a widely implemented method of exposing data on the Web in many domains. Many data providers now identify their data items by means of dereferencable HTTP URIs and deliver data not only in the human-readable HTML but also in the machine-readable RDF format. In many cases, the RDF data representations are generated, often on-the-fly, from underlying (e.g., relational) data stores for the purpose of data-exchange. In certain cases, however, it might be necessary to store and retrieve RDF data directly, which requires special storage solutions that support the schema-less and graph-based nature of the RDF data model. Many solutions have been developed throughout the past years and often it is unclear which solutions to adopt.

The Linked Data approach will play a major role in the European Digital Library (`http://europeana.eu`) and solutions that can handle data expressed in the newly created, RDF-based *Europeana Data Model* (EDM) are currently being investigated. This report summarizes the results of a study we performed on existing RDF stores, in the context of Europeana and encompasses the following contributions:

- An inventory of existing RDF store solutions comparing their non-functional features.
- A meta-analysis that surveys and summarizes the results of previous RDF store studies.
- A detailed qualitative analysis of all stores in the RDF store inventory.
- A quantitative load and query-response time evaluation carried out on the complete Europeana data set.

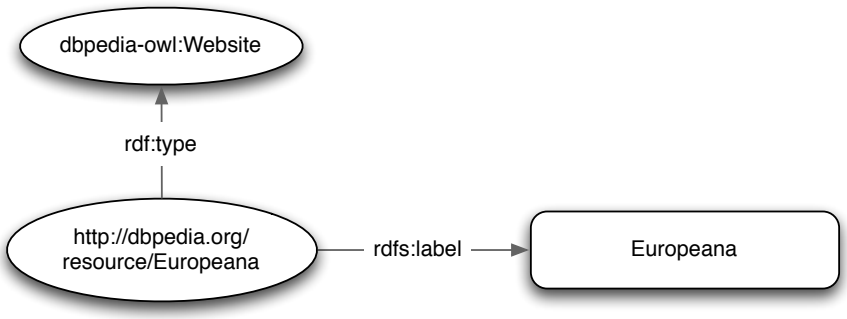

Figure 1: Example DBpedia RDF graph representing Europeana.

The overall objective of this study was to find an appropriate RDF store that can host EDM data and expose these data as Linked Data on the Web. We observed that the majority of stores that were developed in recent years are so called *native-stores*, which are systems that do not make use of a relational database as storage backend, and *hybrid stores*, which combine relational and native RDF storage functionalities. They can load datasets in the size of the Europeana dataset (approx. 380 million triples) in reasonable time (> 1.5 h) and answer basic SELECT and DESCRIBE queries in a time-frame between 0.3 and 1 second respectively. Since these response times can hardly compete with those of full-text search engines such as Apache Lucene, an RDF store should not replace full-text search engines. Given that an RDF store meets Europeana's functional and non-functional requirements, it could, however, serve as persistent and query-able metadata storage that can be used to serve Europeana Data as Linked Data. The aim of this report is to discuss the functional and non-functional features of available RDF store solutions and to give deeper insight into their performance behavior.

The remainder of this report is structured as follows: in Section 2, we briefly introduce the RDF-specific notions used in the subsequent sections and introduce a first RDF store classification. Then, in Section 3, we set up the RDF store inventory and describe their non-functional features. Section 4 summarizes the results of studies carried out by others in the area of RDF store analysis and performance evaluation. In Section 5 we summarize the outcome of a qualitative functional feature analysis we carried out for all stores contained in our RDF store inventory and in Section 6 we present the results of the RDF store performance evaluation. Finally, in Section 7, we summarize the major findings of our study.

2 Background

The capabilities of RDF stores inherently depend on the properties of the RDF data model and other Semantic Technologies built on-top of RDF. In the following, we briefly introduce the technologies and notions mentioned in this report, explain what an *RDF Store* is, and provide a first classification of existing RDF storage approaches. We also outline the major peculiarities RDF has compared to traditional data models we can find in today's relational database management systems. This section also provides the definitions for the terms we use throughout this report.

2.1 Semantic Web Technologies

The Resource Description Framework (RDF) [26] is a graph-based, semi-structured data model for representing metadata *about* a certain resource. It allows us to formulate statements about resources, each *statement* consisting of a *subject*, a *predicate*, and an *object*. The subject and predicate in a statement must always be resources, the object can either be a resource or a literal node (label). A statement is represented as a *triple* and several statements form a *graph*. Figure 1 shows an example RDF graph representing data about the Europeana project. It is identified by the URI `http://dbpedia.org/resource/Europeana`, is of type

```
<http://dbpedia.org/resource/Europeana>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://dbpedia.org/ontology/Website> .

<http://dbpedia.org/resource/Europeana>
<http://www.w3.org/2000/01/rdf-schema#label>
"Europeana" .
```

Figure 2: The example RDF graph from Figure 1 in N-Triples representation.

`dbpedia-owl:Website` and has the label "Europeana". Figure 2 shows the data defined by this graph in N-Triples[1] serialization.

The concept of *Named Graphs* [15] is an extension of the RDF specification an allows for the expression of meta information about graphs and the relationships between them. RDF by itself merely provides means to represent the structure and semantics of one single graph but does not include mechanisms that allow for identifying or referring to a set of triples defined in another graph. By naming RDF graphs with URIs, they can be uniquely identified and assertions about them can be made. Therefore, a single RDF document can host multiple graphs where each graph is identified by its URI. A common proposition of implementing named graphs in RDF is to replace the triple-based model by a *quad-based model* in which the fourth element contains the graph URI to which an RDF triple belongs. Named graphs exhibit some significant advantages compared to the RDF innate concept of *reification*; a detailed discussion regarding the current limitations of RDF reification w.r.t data syndication, ontology evolution and versioning etc. together with a formal specification of the named graphs syntax and semantic is provided by [16, 15].

The *RDF Vocabulary Description Language RDF Schema (RDFS)* [46] and the *Web Ontology Language (OWL)* [48] are means to describe the vocabulary terms used in an RDF model. RDFS provides the basic constructs for describing classes and properties and allows to arrange them in simple subsumption hierarchies. Since the expressiveness of RDFS is limited and misses some fundamental modeling features often required to construct vocabularies, the *Web Ontology Language (OWL)* was created. It is based on RDFS and allows the distinction between attribute-like (`owl:DatatypeProperty`) and relationship-like (`owl:ObjectProperty`) properties and provides several other expressive modeling primitives (e.g., class union and intersections, cardinality restrictions on properties, etc.), which allow us to express more complex models, which are then called ontologies. With RDFS and OWL one has the possibility to define models that explicitly express data semantics and specify and process possible inferences of data. The formal grounding of RDFS and OWL (Description Logics) allows applications to reason on RDF statements and infer logical consequences.

The *SPARQL Query Language for RDF* [47] is an expressive language for formulating structured queries over RDF data sources. It defines a protocol for sending queries from clients to a SPARQL endpoint and for retrieving the requested results via the Web.

2.2 RDF Stores

While small RDF graphs can be efficiently handled and managed in computers' main memory, larger RDF graphs render the deployment of persistent storage systems indispensable. RDF stores are purpose-built databases for the storage and retrieval of any kind of data expressed in RDF. Although the term *triple store* is often used for that kind of systems, we use the term *RDF store* in order to abstract over any kind of system that is capable of handling RDF data, including triple stores, quad stores, etc. For the purpose of this study, we define *RDF stores* as *systems that allow the ingestion of serialized RDF data and the retrieval of these data later on.*

[1] N-Triples is line-based, plain text format for representing RDF data. See `http://www.w3.org/2001/sw/RDFCore/ntriples/`

We distinguish the following types of RDF stores based on their architecture: *native stores*, *DBMS-backed stores*, and *RDF wrappers*.

2.2.1 Native stores

Native stores implement a complete database engine that is optimized for RDF processing and operates independent of any other Database Management System (DBMS). Data are stored directly on the file system, either in a single file or distributed among several segments on the underlying file system.

2.2.2 DBMS-backed stores

DBMS-backed stores make use of the storage and retrieval functionality of existing (mostly relational) database management systems. RDF stores, which are backed by relational DBMS, can apply different kinds of *storage models* for representing the RDF model in the underlying relational schema. We can classify these models into *generic schemas*, which store RDF data regardless of any ontology in place, and *ontology-specific schemas*, which resemble a specific ontological structure in the relational table design.

Generic schemas We can further divide the generic schemas into a *single database table design* (the most simple generic schema) resembling a serialized RDF triple structure using a three-column table for storing subject, predicate, and object, a *property table design* (cf. [3]), which is a flattened denormalized RDF table schema consisting of one subject column followed by many property columns, a *normalized table design* introducing separate tables for storing URIs and literals, as well as *hybrid approaches* that resemble the advantages of singe and normalized table designs[2]. The advantage of a generic schema approach is clearly the flexibility gained by the schema-less data representation approach. The major disadvantage is the high number of self-joins, which are required for answering non-trivial queries (cf. [3]).

Ontology-specific schemas DBMS-backed stores that follow an ontology-specific schema approach do not store triples in a single table but apply a schema structure that reflects the structural properties of the ontologies in place. Evolutions in the ontology design are reflected in database table design, i.e., whenever a class or property changes in the ontology, this change is reflected in the database tables. According to [24] we can distinguish three different ontology-specific schema layouts:

- *Horizontal representation* or *one-table-per-class schema*: each ontology class is reflected in a database table, which holds all instances of a class. Including a new property in the ontology requires all instance-related database tables to be extended by an additional column representing the newly introduced property. The major disadvantages of this approach are the necessity to restructure database tables whenever the adhering ontology changes and the lacking support for multi-value properties.
- *Decomposition storage model*: this schema is also called *one-table-per-property-schema* or *vertically partitioned schema* and stores each property in one single table (including RDF/S properties) consisting of two columns: *subject* and *object*. This table layout also lacks query efficiency (performance) when it comes to complex queries because it leads to many joins among property tables. However, it can outperform single-database schemas by a factor greater than two [3].
- *Hybrid schemas*: are more common and combine the advantages of both *one-table-per-class schemas* and *one-table-per-property schemas* since ontology changes do not lead to database table restructuring. Instead, each class is represented as one single table where only the IDs of the instances of a specific class are stored.

[2] Advantages and limitations of the different generic schema approaches are discussed and exemplified in ([24]).

2.2.3 RDF wrappers

RDF wrappers are another noteworthy type of RDF management system. An RDF wrapper is a lightweight software components that is set up on top of an existing data source and exposes the data stored therein as RDF without affecting existing storage infrastructures. RDF Wrappers exist for relational databases (e.g., D2RQ Server[3], Triplify [5]), for protocols such as OAI-PMH (e.g., OAI2LOD Server [23]), for structured file formats (e.g., TripCel [8]), for complete file systems (TripFS [41]) and other kinds of data sources. Depending on the capabilities of the underlying data source, wrappers can also provide structured RDF access via SPARQL. The distinguishing feature between RDF wrappers and the previously mentioned types of RDF stores (native, DBMS-backed) is that RDF wrappers provide read-only access to RDF data. Since this doesn't fit into our definition of RDF stores, we exclude systems in this category from further analysis.

2.3 RDF Peculiarities Compared to Relational Database Models

RDF Stores are optimized to quickly process operations on RDF models and are highly flexible because they do not rely on a fixed, pre-defined schema. Hence, schema (domain model) changes can be realized without direct schema and data migrations in the underlying storage infrastructure, as it would, for instance, be the case in relational databases. If an RDF store builds on a relational database, schema migrations are typically handled transparently by some RDF middleware. The drawback of this flexibility lies in the loss of performance and scalability in query processing. Existing query optimization algorithms typically rely on the existence of fixed schema information. In the following, we describe some of the distinct and significant features the RDF data model exposes compared to traditional database systems (for a detailed discussion see [34]):

- The unpredictable structure of RDF documents not only affects the execution of (arbitrary) query patterns, but also requires a generic structure for RDF data storage in, for instance, relational database systems. Therefore, RDF stores should provide support for generic storage schemas covering RDF data model characteristics and the efficient execution of arbitrary query patterns.
- Data aggregation in RDF is relatively simple compared to the complex schema realignments known in, for instance, relational database systems since RDF graphs can be easily merged. For instance, if two resources are using the same URI, its valid to assume that they refer to the same (real world) entity.
- URIs exhibit a standard and uniform identification scheme. By using URIs for identifying resources and concepts, data discovery across documents and systems is facilitated since the use of URIs allow for a global identification mechanism among systems and domains.
- RDF is an open world framework with no tightly defined data schemas, wherefore *anyone can say anything about any topic*. Therefore, the RDF data model allows for the insertion of individual RDF statements to existing data sources but also for specifying assertions about a resource in different systems. In this respect, RDF is considered to be a schema-less model.
- RDF offers a powerful and open knowledge representation language that can be used to virtually express any type of knowledge due to its underlying triple-based generic schema. This simplicity and flexibility led to the creation of a multitude of open-source frameworks and tools (e.g., knowledge extraction engines, reasoners etc.) for processing and storing RDF data that can be used in existing applications.

3 RDF Store Inventory

After having introduced the fundamental technical properties of RDF stores, we now given an overview of existing RDF stores. We include all publicly available solutions, which are referenced and described on

[3]D2RQ Server: `http://www4.wiwiss.fu-berlin.de/bizer/d2rq/`

the Web[4] or were discussed in related publications (see Section 4) throughout the past years. Following the categorization introduced in Section 2.2, we categorize them into *native* or *DBMS-backed* stores, or, if both architectural styles are applicable, into *hybrid* stores. For each RDF store solution, we provide a short background description and details on the following **non-functional** features:

- *Licensing model*: some RDF store solutions are `commercial`, others `open source` and/or `free` of charge. The latter may be released under various licenses (GNU GPL, Apache License, etc.).
- *Deployment*: depending on their architecture, RDF Stores can either be installed as a stand-alone `server`, or be used as a remote `hosted service`. Some are designed as `libraries`, which can be integrated into applications, and provide RDF storage functionality via their API.
- *Operating environment*: the operating environment (Windows, Linux, Mac OS X, etc.) for running a particular RDF store. Several RDF stores might also be implemented as `platform-independent` systems that run on any platform supporting a specific programming language (e.g., Java, PHP).
- *Client connectors*: the programming languages (C, C#, Java, PHP, etc) for which client connectors are available.
- *Extensibility mechanism*: if an RDF Store follows a modular architecture, it might be possible to create extensions (e.g., plugins, modules, etc.)
- *Storage model*: the storage model (see Section 2.2.2) a DBMS-based store applies.
- *Latest Release*: the month when the latest release of an RDF store was released at the time of this writing.

3.1 Native RDF Stores

According to the classification introduced in Section 2.2, all RDF stores that implement their own database engine without reusing the storage and retrieval functionalities of other database management systems fall into the category of native stores. In this section, we give an overview on the non-functional features of existing native RDF stores.

3.1.1 4/5store

4Store[5] is an RDF database developed by Garlik Inc. It is implemented in ANSIC99 and available under the GNU General Public License (GPL), version 3. It is designed for UNIX-like systems (Linux, Mac OS) and runs as a server on a single-machine or in cluster-mode on 64bit machines. Client connectors are available for PHP, Ruby, Python, and Java. Dedicated extensibility mechanisms are not foreseen.

5Store[6] is the latest clustered RDF store development by Garlik. Unlike 4Store, it is commercial software and not publicly available. 5Store provides similar features as 4Store, but improved efficiency and scalability. It should scale up to clusters of 1000 machines, with no effective limit on the number of triples stored.

3.1.2 AllegroGraph

AllegroGraph[7] is a commercial RDF graph database and application framework developed by Franz Inc. for the storage and querying of RDF data. It can be deployed as a standalone database server and offers interfaces (entry points) for remote access where the communication between the AllegroGraph Server and client processes is realized through HTTP. AllegroGraph is available for Windows, Linux, or Mac OS X platforms either as 32-bit or 64-bit version. Franz Inc. offers different editions of AllegroGraph together

[4] `http://esw.w3.org/LargeTripleStores`
[5] `http://4store.org/`
[6] `http://4store.org/trac/wiki/5store`
[7] `http://www.franz.com/products/allegrograph/`

with a number of different clients: the free RDFStore server edition is limited to storing less than 50 million triples, a developer edition capable of storing a maximum of 600 million triples, and an enterprise edition whose storage capacity is only limited by the underlying server infrastructure. Clients connectors are available for Java, Python, Lisp, Clojure, Ruby, Perl, C#, and Scala. AllegroGraph's storage capabilities are not limited to storing RDF-data; instead, due to its generic storage model, it can store any form of graphs whose elements can be represented as tuples consisting of node elements s and o, an edge p, and additional data g [25]. AllegroGraph provides a number of additional features such as geospatial or temporal reasoning, social network analysis, or federation. These features are packaged into so-called extensions. However, AllegorGraph does not offer possibilities for building custom extensions in the form of plug-ins or modules.

3.1.3 Jena TDB

Jena TDB[8] is a component of the Jena Semantic Web framework and available as open-source software released under the BSD license. It can be deployed as a server on 64 and 32 bit Java systems and accessed through the Java-based Jena API library as well as through a set of command line utilities. Jena TDB also supports the SPARQL query language for RDF data. The latest version of TDB was released in July 2010.

3.1.4 Kowari

The Kowari Metastore [49] is a multi-platform, open-source, database server for RDF and OWL-based meta data analysis, retrieval, and storage. It is implemented in Java and licensed under the Mozilla Public License Version 1.1[9]. Clients can connect to the database via Java either using JRDF or Kowari's Java-based RDF API library. Kowari is a native RDF repository [10] based on the XA-Triplestore engine that provides native RDF support and allows for the deployment of multiple databases per Kowari server instance. Kowari does not provide a dedicated extension mechanism but can be extended programmatically and expose interfaces for the integration of external RDF data sources, which can be mapped to the internally used RDF graph schema so that they can be queried like internal RDF meta data. Active developments stopped in 2006 with the release of Kowari v1.1 and have been continued as a Mulgara fork.

3.1.5 Mulgara

The Mulgara Semantic Store[10] is the community-driven successor of *Kowari* and is described as a purely Java-based, scalable, and transaction-safe RDF database for the storage and retrieval of RDF-based metadata. The current version as of September 2010 is licensed under the Open Software License v3.0 as well as the Apache License 2.0, which is being applied to new code contributions. Mulgara provides connectors for Java-based applications via the JRDF and Jena libraries. Since Mulgara is a fork of the Kowari project and uses an enhanced version of the XA Triplestore engine (version 1.1), most of the features such as multi-platform support, deployment model, storage model etc. described for Kowari also apply to Mulgara, although Mulgara provides more client connectors (see above). Mulgara is optimized for metadata management [36] and uses a native quad-based model for representing and storing RDF-based metadata where the conceptual RDF model elements subject, predicate, and object are extended by the *named graph* concept.

3.1.6 OWLIM

OWLIM[11] is a family of commercial RDF storage solutions, provided by Ontotext [33]. It is available in two different editions: *SwiftOWLIM* is designed for medium data volumes (up to 100 million triples) since reasoning and query evaluation are performed in main memory, while *BigOWLIM* is designed for large data volumes and uses file-based indices that allow it to scale up to billions of RDF triples. Additionally, OWLIM is available as a SAIL (Storage and Inference Layer) for the Sesame RDF framework.

[8] `http://www.openjena.org/TDB/`
[9] `http://www.mozilla.org/MPL/MPL-1.1.html`
[10] `http://mulgara.org/`
[11] `http://www.ontotext.com/owlim/`

SwiftOWLIM source code is provided free of charge for any purpose under a GNU LGPL license[12]. BigOWLIM is free to use for research, evaluation, and testing purposes; for commercial applications an appropriate license is required. Both versions do not provide a dedicated extensibility mechanism but allow the definition of custom rules and rule languages for the inferencing process.

BigOWLIM is in use for a large number of Semantic Web and Linked Data applications, including the *BBC 2010 World Cup Website*[13] and the *LinkedLifeData* platform[14]. The latest release was published in May 2010.

3.1.7 Talis Platform

The Talis Platform[15] is a "multi-tenant" system, delivered as Software as a Service (SaaS). It uses multiple stores both for arbitrary document data as well as RDF-based semantic data storage, where a store consist of a *ContentBox* for persisting unstructured binary blocks, and a *Metabox* for holding structured metadata. The hosted stores are accessible by clients via numerous web interfaces (e.g., transmitting data to a store can be performed via a HTTP POST request) or software libraries and tools. Currently, Talis offers a number of different programming language libraries (e.g., PHP, Java, Javascript, Ruby, Phyton, and C#) and two licensing models: (1) free development stores for members of the n^2 developer community and public data providers, and (2) a fixed pricing schema oriented on the amount of stored triples for commercial usage.

3.1.8 ClioPatria

ClioPatria[16] is an RDF Store implemented in SWI-Prolog. It supports both the SPARQL and the SeRQL query languages. ClioPatria is open source an can be installed on Unix/Linux and Windows Systems. Client connectors exist for Java (via Sesame) and Prolog. Dedicated extensibility mechanisms are not provided. The ClioPatria download page does not provide explicit releases but allows one to check out code from the code repository.

3.2 DBMS-backed Stores

When an RDF Store uses the storage and retrieval functionality provided by another database management system, it falls into the category of DBMS-backed stores.

3.2.1 YARS2

YARS2[17] is a Java-based, federated RDF server for querying structured graph data using a declarative query language. It uses the Berkeley DB for storing RDF data in B+Trees. YARS2 is released under the GNU General Public License (GPL) and operates in any Java-supporting environment. The interface is built upon the REST principle and for interacting with YARS is plain HTTP. It seems, that this project was not continued because the last release is Beta 3 in 2006.

3.2.2 3store

3store[18] is an RDF triple store developed at the University of Southampton, released under the GNU GPL. It is written in C, runs on Linux and Unix platforms, and is backed either by a MySQL or Berkeley DB. Its storage model follows a triple table layout with additional dictionary tables for prefixes, URIs, and literals; hence, a generic schema as described in Section 2.2.2. Development stopped mid 2006.

[12] `http://www.gnu.org/copyleft/lesser.txt`
[13] `http://bbc.co.uk/worldcup`
[14] `http://www.linkedlifedata.com/`
[15] Talis Platform `http://www.talis.com/platform/` and Documentation `\url{http://n2.talis.com/wiki/Main_Page}`
[16] `http://www.swi-prolog.org/web/ClioPatria.html`
[17] http://sw.deri.org/2004/06/yars/
[18] `http://sourceforge.net/projects/threestore/`

3.2.3 ARC

ARC[19] is a free, open-source semantic web framework for PHP applications released under the W3C Software License and the GNU GPL. It is designed as a PHP library and includes RDF parsers and serializers, an RDBMS-backed (MySQL) RDF storage, and implements the SPARQL query and update specifications. Since it is written in PHP, it runs in all PHP-supporting operating environments. A plugin mechanism allows ARC to be extended with additional, custom extensions. At the moment, ARC applies a generic triple table layout where subject, predicate, object, and graph values are stored in separate dictionary tables.

3.2.4 Jena SDB

Jena SDB[20] is another component of the Jena Semantic Web Framework and provides storage and query for RDF datasets using conventional relational databases. Among the supported databases are: Microsoft SQL Server, Oracle 10g, IBM DB2, PostgreSQL, MySQL, HSQLDB, H2, and Apache Derby. Jena SDB is open source and released under the BSD license. It runs as server and can be integrated into applications that make use of the Jena library. As all Jena components, also SDB runs on all Java-supporting platforms. The latest SDB version considered in this report was released in July 2010.

3.2.5 Oracle 11g

The *Spatial* module of Oracle's Database Enterprise Edition 11g[21] also supports RDF data management. Oracle 11g is a commercial product, which is free of charge for non-commercial purposes. It is deployed as a database server and provides programming interfaces (connectors) for SQL (SQLplus, PL/SQL) and Java-based applications (via Jena/Sesame and JDBC). To the best of our knowledge, it is currently not possible to extend or customize Oracle 11g's semantic features by additional plugins or modules. For each model (graph) Oracle generates a separate triple table containing references to subject, predicate, and object values, which are stored in an external dictionary table. Hence, it follows a generic triple table layout. At the time of this writing the latest release was of September 2010.

3.2.6 Semantics Platform

Intellidimension Semantics Platform[22] is a family of commercial products for RDF-based applications. It is based on the .NET Framework and provides two different storage solutions: *Semantics.Datacenter*, which is an in-memory distributed storage component for RDF data, while *Semantics.Server* uses Microsoft SQL Server as persistence layer. Offered as complementary product, *Semantics.Framework* is a comprehensive RDF framework for the .NET framework, offering support for in-memory graphs, SPARQL, inference rule processing, and APIs for Semantics.Datacenter and Semantics.Framework.

The products of the Semantics Suite are offered under a commercial license, while evaluation versions are available for download. Details about the internal storage model and the latest release data are not available.

3.2.7 Boca

Boca[23] is a Java-based, multi-user RDF repository being developed by the IBM Adtech group. It is part of IBM Semantic Layered Research Platform (SLRP) and released under the GNU General Public License (GPL). It operates in any Java-supporting environment and can be integrated into applications via a Java-based client connector. It uses IBM's DB2 as backend database and follows a generic storage model. This project seems discontinued because the last release is in 2007.

[19] `http://arc.semsol.org/`
[20] `http://www.openjena.org/SDB/`
[21] `http://www.oracle.com/technetwork/database/options/semantic-tech/index.html`
[22] `http://www.intellidimension.com/products/semantics-platform/`
[23] `http://ibm-slrp.sourceforge.net`

3.3 Hybrid Stores

An RDF store that supports both architectural styles (native and DBMS-backed) falls into the class of hybrid stores. In the following, we outline the major non-functional features of these stores.

3.3.1 RedStore

RedStore[24] is a lightweight RDF store based on the Redland RDF library[25] for the C programming language. It is compatible with the interfaces for 4store and supports, in addition to its native persistent or in-memory storage, a variety of storage backend adapters, including MySQL, Postgres, and Virtuoso. In native mode RedStore uses hashtables for persisting RDF data. RedStore is offered under the GNU GPL license[26].

3.3.2 Sesame

Sesame[27] is an open source framework for storage, inferencing and querying of RDF data. It is a library that is release under the Aduna BSD-style license and can be integrated in any Java application. Additionally, it is possible to deploy Sesame as an RDF repository and query service. Sesame includes RDF parsers and writers (Sesame Rio), a storage and inference layer (SAIL API) that abstracts from storage and inference details, a repository API for handling RDF data, and an HTTP Server for accessing Sesame repositories via HTTP. It operates in any Java-supporting environment and can be used by any Java application. Many developers have already contributed extensions and plugins[28]. Sesame provides native as well as RDB-backed triple storage. The latter follows a generic triple table layout. The latest version (2.3.2) was released in July 2010.

3.3.3 Virtuoso

OpenLink Virtuoso Universal Server[29] is a hybrid storage solution for a range of data models, including relational data, RDF and XML, and free text documents. Through it unified storage it can be also seen as a mapping solution between RDF and other data formats, therefore it can serve as an integration point for data from different, heterogeneous sources [18]. Virtuoso has gained significant interest since it is used to host many important Linked Data sets (e.g., DBpedia[30]), and preconfigured snapshots with several important Linked Data sets are offered. Virtuoso is offered as an open-source version; for commercial purposes several license models exist. It can be deployed as a server on major platforms.

3.3.4 BigData

Bigdata[31] is a clustered RDF store for ordered data (B+Trees) and designed to run as as server on commodity hardware. It is available under the GNU General Public License (GPL) and is designed for UNIX systems (Linux) with client connectors available for Java (Sesame). Additional scale can be achieved by simply plugging in more data services dynamically at runtime, which will self-register with the centralized service manager and start managing data automatically. Scale-out is achieved via dynamic key-range partitioning of the B+Tree indices.

[24] http://www.aelius.com/njh/redstore/
[25] http://librdf.org/
[26] http://www.gnu.org/licenses/gpl.html
[27] http://www.openrdf.org/
[28] http://www.openrdf.org/contrib.jsp
[29] http://virtuoso.openlinksw.com
[30] http://dbpedia.org
[31] http://www.systap.com/bigdata.htm

3.4 Summary

In this section, we set up an RDF store inventory that lists and briefly describes currently available RDF store solutions. We assigned each store to one of three categories (native stores, DBMS-backed stores, hybrid stores) and described the non-functional features for each store.

Regarding the results summarized in Figure 3 we can observe that most RDF stores follow a native storage approach. Especially when regarding the latest release date, it seems that RDF store development goes into the direction of native stores. From the DBMS-backed stores, only four are actively being enhanced and have provided recent (after 2008) releases. It also turns out that the majority of triple stores are open source, although the market for commercial stores (6 out of 19) seems to grow. The vast majority of RDF stores can be deployed as a server on any common operating systems (Windows, Mac OS X and Linux). Most RDF stores provide client connectors for Java, some (4Store, AllegroGraph, Talis Platform) also for scripting languages such as Python or Ruby. The predominant storage model of DBMS-backed stores is the generic model, which doesn't take into account the classes and properties defined in the applied ontologies. Regarding the latest release dates, we can conclude that the development of RDF stores is an active, on-going process.

4 Related Work

4.1 Performance Reports

Liu and Hu [28] evaluated 7 RDF storage system configurations: in-memory Sesame (SESAME-MEM), Sesame with relational backend (Sesame-DB), in-memory Jena (JENA-MEM), Jena with relational backend (Jena-DB), Sesame with native RDF storage (SESAME-NATIVE), Kowari and YARS. They used datasets generated from the LUBM (Leigh University Benchmark) dataset, which contained approximately 2.5 million triples. Evaluating the data loading performance of these stores revealed that the memory-based stores have the best performance. The database-backed systems performed badly in data loading, the three native systems loaded data with linearly growing loading time. Also the query response times of DB-backed stores were higher than in all other types of stores (mem, native) mainly because query answering requires numerous joins on data stored in a large single table. The query performance of the native stores depends on their inference support. Sesame-NATIVE, which does not support inference, shows similar results than Sesame-MEM. Kowari can answer queries in reasonable time. YARS could only answer a subset of all queries but performed better than Sesame.

Rohloff et al [37] compared the performance of various RDF stores configurations in deployment scenarios that include very large knowledge bases (hundreds of millions of triples). They derived their test data sets and queries from the LUBM benchmark. The evaluated triple stores were Sesame 1.2.6 and Jena 2.5.2. with various storage backends (MySQL 5.0, DAML DB 2.2.1.2, SwiftOWLIM 2.8.3, BigOWLIM 0.9.2) as well as AllegroGraph 1 and 2.0.1. The generated test set consisted of roughly 200 million triples in the first round and 1 billion triples in the second test round. The applied evaluation metrics were: *cumulative load time*, *query response time*, *query completeness and soundness*, and *disk-space requirements*. All triple store configurations had linear load times. Jena + DAML DB and Sesame + DAML DB had the best cumulative load time performance. Jena and Sesame configurations with BigOWLIM made a factor three more efficient use of disk space then Sesame and Jena with DAML DB. Jena + DAML DB had the best query response performance. The overall result of this study is that RDF stores based on the DAML DB and BigOWLIM technologies exhibit the best performance among the tested stores.

Abadi et al [3] propose *vertically partitioning* as a schema design for storing RDF data in relational databases. Essentially, this approach creates one relation per unique property in a data set. They evaluated their approach against two other schema layouts (single triple relation, property tables) and SESAME-NATIVE using a benchmark created from the Barton Libraries dataset containing approximately 50 million triples with 221 unique properties [2]. The reported result was that property table and vertical partitioning approaches both perform a factor 2-3 faster than the generic schema triple-store approach and that using

		Non Functional Features						
		Licensing Model	**Deployment**	**Operating Environment**	**Client Connectors**	**Extensibility Mechanism**	**Storage Model**	**Latest Release**
Native Stores	**4Store**	GNU GPL	Server	Linux, Mac OS	PHP, Ruby, Python, Java	-	N/A	February 2010
	AllegroGraph	commercial, free	Server	Windows, Linux, Mac OS X	Java, Python, Lisp, Clojure, Ruby, Perl, C#, Scala	-	N/A	September 2010
	Jena TDB	BSD license	Server Library	platform-independent (Java)	Java	-	N/A	July 2010
	Kowari	Mozilla Public License Version 1.1	Server	platform-independent (Java)	Java	-	N/A	December 2004
	Mulgara	Open Software License v3.0 Apache License 2.0	Server	platform-independent (Java)	Java	.	N/A	September 2010
	OWLIM	commercial, GNU LGPL	Server, Library	platform-independent (Java)	Java	Custom Rule Language	N/A	May 2010
	ClioPatria	Open Source, no license	Server	Linux / Unix, Windows	Java, Prolog	-	N/A	October 2010
	Talis Platform	free and commercial	Hosted Service	platform independent (REST interface)	PHP, Java, Javascript, Ruby, Python, C#	-	N/A	August 2010
DBMS-backed Stores	**YARS2**	BSD license	Service	platform-independent (Java)	Java	-	B+ Trees	July 2006
	3Store	GNU GPL	Server	Linux / Unix	-	-	Generic	July 2006
	ARC	W3C Software License, GNU GPL	Library	platform-independent (PHP)	N/A	Plugin Mechanism	Generic	July 2010
	Jena SDB	BSD license	Server Library	platform-independent (Java)	Java	-	Generic	June 2010
	Oracle 11g	commercial, free	Server	Linux, Windows, Solaris, HP-UX, AIX	Java, SQL	-	Generic	September 2010
	Semantics Platform	commercial	Server	Windows	.NET	-	unknown	June 2010
	Boca	GNU GPL	Library	platform-independent (Java)	Java	-	Generic	July 2007
Hybrid Stores	**RedStore**	GNU GPL	Server, Library	Linux, Mac OS X	C	Storage Modules	Hashtables	April 2010
	Sesame	Aduna BSD-style license	Server, Library	platform-independent (Java)	Java	Plugin Mechanism	Generic	July 2010
	Virtuoso	commercial, GNU GPL	Server	Windows, Linux, Mac OS X	Java	Plugin Mechanism	Object-relational	July 2010
	Bigdata	GNU GPL	Server	Linux / Unix	Java	Plugin Mechanism	B+ Trees	July 2010

Figure 3: Existing RDF Stores and their non-functional features.

a column-oriented DBMS for the vertically partitioning approach adds another factor of 10 performance improvement, which is a factor 32 faster than triple stores.

Schmidt et al [42] compared the performance of the triple (single triple relation) and the vertically partitioned (one relation per predicate) scheme for storing RDF data in DBMS using the SP2Bench SPARQL benchmark. They also tested the same benchmark with SESAME-NATIVE and a purely relational scheme. Their experiments include eleven real-world sample queries and show that, when triples are physically sorted by (predicate, subject, object) the triple table approach becomes competitive to a vertically partitioned approach. However, none of the approaches scales for real-world queries to documents containing tens of millions of RDF triples and none of the approaches can compete with a purely relational model.

Minack et al [30] provided a detailed feature and performance comparison of full-text queries (classic IR queries) as well as hybrid queries (structured and full-text queries) of the most widely used RDF stores (Jena, Sesame2, Virtuoso, YARS) by extending the LUBM benchmark with synthetic scalable full-text data and corresponding queries. Their evaluation includes 21 real-world sample queries. Evaluating the query response time of these stores revealed that Jena and Sesame2 are the best choices for basic or advanced IR queries and specially Sesame2 is the only RDF store sufficiently solving the tasks of full-text search on RDF data. However Virtuoso shows an impressive performance, but does only supported a small subset of queries.

Lee [27] evaluated which open source RDF store system (Jena, Joseki, Kowari, 3store, Sesame) holds the most promise for acting as large, remote backing stores for the SIMILE project (browser-like applications). They were choosing a subset of ARTstor, art metadata in addition to a subset of the MIT OpenCourse-Ware (OCW) as a dataset which is 27.4 MB in RDF/XML format. By choosing the configuration time as an interesting metric (e.g., how fast does a store return its results for creating the in-memory cache?) the evaluation results shows that for network models, the fastest were 3store and Sesame. Therefore Sesame and 3store are the most worthwhile for exploring as remotely accessible stores for even larger datasets. They also discussed that non-Joseki-based remote stores are approximately an order of magnitude faster at cache initialization because of the focus of the Jena and Joseki projects on doing things correctly instead of doing them as quickly as possible.

Figure 4 summarizes the observations reported in these surveys. For each survey, it indicates the approximate number of triples and the triple stores included in the evaluation. It also reports which stores performed best w.r.t. data loading, query- and inference performance.

4.2 Other RDF Store Reports

Hertel et al [24] reviewed architecture, storage models, typical functionalities of RDF middleware, (such as data model support and reasoning capabilities), RDF query languages with a special focus on SPARQL and scalability of existing RDF storage and retrieval systems. Their observation is the fact that almost all systems rely on relational databases that provide very limited support with respect to data model and reasoning. For instance most RDF stores are not really specialized database systems for RDF data but rather an intelligent middleware that wraps existing database technology. Therefore two trends for further development of RDF technologies can be identified: first the extension of existing systems to more expressive representation languages. Second, the scalability of RDF infrastructures to internet scale.

Bizer et al [13] introduced the Berlin SPARQL Benchmark (BSBM), which is built around an e-commerce use case. The dataset contains a set of products offered by different vendors and consumer reviews from various sites. The benchmark queries simulate search and navigation patterns of fictitious consumers who are looking for products. The following systems were analyzed in the experiment: the D2R Server, which is an RDF wrapper for relational databases, Sesame, Virtuoso, and Jena SDB. The benchmark revealed: concerning the load times, Sesame and Virtuoso are faster for small datasets while Jena SDB is faster for larger datasets. Regarding the performance of relational database to RDF wrappers, D2R Server has shown a very good performance for specific queries against large datasets, but was very slow at others. Finally they concluded that no store is dominant for all queries. The Berlin SPARQL Benchmark has been updated recently[32] and the following systems were analyzed: Virtuoso, Jena TDB, 4store, BigData, and BigOWLIM.

[32] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/

		Setting			Best -performing RDF Stores		
		Analyzed Triple Stores	**Applied Benchmarks**	**Number of Triples**	**Load Time**	**Query Response Time**	**Inference Support**
Reports	**Liu and Hu**	Sessame-MEM, Sesame-DB, Jena-MEM, Jena-DB, Sesame-NATIVE, Kowari, YARS	LUBM	2.5 M	Sesame-NATIVE, Kowari, YARS	Sesame-MEM, Jena-MEM, Sesame-NATIVE, Kowari, YARS	Kowari, YARS
	Rohloff et al	Sesame, Jena (with MySQL, DAML-DB, SwiftOWLIM, BigOWLIM), AllegroGraph	LUBM	200 M / 1 B	Jena and Sesame with DAML-DB and BigOWLIM	Jena + DAML-DB	-
	Abadi et al	UNKNOWN with triple, property table, and vertical partitioning layout; Sesame-NATIVE	Barton Libraries dataset	50 M	vertical partitioning schema		
	Schmidt et al	UNKNOWN with triple, property table, and vertical partitioning layout; Sesame-NATIVE	SP2Bench SPARQL Benchmark		(sorted) triple table schema competetive to vertically partitioned approach; purely relational approach is still the best		
	Minack et al	Jena, Sesame2, Virtuoso7, YARS	LUBM	7.6 GB (LUBMft(100))	_	Jena and Sesame2	_
	Lee et al	Jena, Joseki, Kowari, 3store, Sesame	_	27.4 MB	3store and Sesame	Jena and Joseki	_

Figure 4: Summery of Related Works

The first results indicate that Virtuoso, 4Store and BigOWLIM are the dominating RDF store solutions.

5 Qualitative Analysis

In the following, a discussion of the features and functional characteristics of a set of RDF storage solutions is presented. We start this section with a discussion of the evaluation criteria that we have applied, followed by a detailed description of each criterion with respect to the analyzed solutions. Figure 5 summarizes our findings.

5.1 Evaluation Criteria

Transactions In the database context, transactions have the purpose of guaranteeing that the data is in a consistent state at any time, regardless of whether operations (modifications) to the data were successful or not, and regardless of whether there are concurrent, probably contradicting modifications. For this purpose, client applications define bounded units of work that are either executed in their entirety, or not at all. These units of work must follow the four basic transaction principles of *atomicity*, *consistency*, *isolation*, and *durability* [21]. Techniques to achieve full support for transaction include *two-phase locking* and *multiversion concurrency control* [9].

Query Languages In general, two approaches for querying RDF data can be distinguished; (1) implementing a specific or dedicated query API, and (2) providing support for query languages that can be translated to SQL in case of relational database management system or being proprietary and DBMS-specific query languages such as SeRQL. However, a vast majority of contemporary RDF stores use common RDF query languages such as SPARQL or RDQL. An important aspect of query translations is to maintain and recover query semantics and translate them into relational calculus, object-specific models, or other database-specific

query formats. Hertel et al. [24] identified six general query properties that help characterizing a query language with respect to expressiveness, completeness, closure, adequacy, orthogonality, and safety[33].

Full-text Queries Full-text queries have become a common requirement for many kinds of applications. With the proliferation of big amounts of data (especially through the World Wide Web) and corresponding search indices, users are acquainted with the common procedure of searching information by keyword, and inspect potential result lists in order to locate the desired information. While RDF query languages usually provide minimum support for full-text queries (e.g., through regular expressions), efficient support for full-text queries, associated data and query pre-processing (e.g., stemming and stop words), and full-text query execution control (e.g., ranking parameters) can be considered as a functional requirement on its own.

Batch Ingest By batch ingest we denote the functionality of loading large amounts of data into the database within a single transaction (if the database supports such). Batch loading is often required in the setup phase of a database system, where existing data from external sources must be loaded, and the database must initialize its index and caching structures according to the loaded data. Ideally, batch ingest does not block other (read) requests.

(Selective) Replication Replication denotes the possibility to automatically copy data across multiple instances of a database in a way that ensures consistency between multiple copies. Replication is beneficial for a number of reasons, including the increase of fault tolerance, performance, and accessibility. Depending on the needs of the application, replication strategies and topologies have to be chosen; a large number of such approaches have been presented and studied in literature (e.g., classification of replication approaches based on their strategies [44], surveys on particular approaches like optimistic replication [39], and specific applications like peer-to-peer-based content distribution [4] or mobile networks [35]). A special case of replication is *partial replication* [29], where only subsets of data are duplicated, which increases the complexity of update synchronization.

Versioning For many applications it is important to access historical versions of data. Versioning denotes the capability to record and store the change history of a given data set, and to make this history accessible to applications. The RDF data model imposes several constraints on versioning algorithms: for instance, there exist multiple, incompatible serialization formats that must be considered [6], and the usage of blank nodes has implications on the comparability of RDF graphs, which is a precondition for calculating diffs in order to represent an RDF graph's change history [50].

Named Graphs The concept of Named Graphs are extensively used and implemented in current RDF stores not only to host multiple graphs and identify the triples belonging to a specific graph, but also to record and keep track of a graph's provenance information. By analyzing provenance information, multiple aspects of a graph can be inferred such as its trustworthiness, proof, or contradictions between graphs can be detected and back-traced. This is an important feature for keeping track of or recording a graph's provenance information, i.e., the source a graph stems from, but also allows for restricting information usage of RDF graphs, implementing access control mechanisms and policies, RDF graph signing, and expressing uncertainty and propositional attitudes (cf. [11]).

Provenance The feature of provenance (also called *lineage* in the database community [17]) describes a RDF store's capability to store and keep track of a graph's origin or source and is an important feature for ensuring data quality, determining the trustworthiness of RDF data sets, and ranking query results [38]. In particular when RDF stores execute forward or backward chaining, provenance information must be stored to determine whether an RDF triple was explicitly asserted or implicitly inferred. Provenance is

[33]For a detailed discussion, the reader is referred to [24],[14],[7]

categorized according to its *granularity* where a distinction is made between *workflow provenance (coarse-grained)* and *data provenance (fine-grained)*. The latter is the more important one from the perspective of database systems and therefore considered for this survey. Data provenance accounts for the derivation of a single data item resulting in the course of a data transformation process applied to a database through a database query [17]. Data provenance by itself can be divided into *Non-annotation approaches* and *Annotation approaches*. Provenance information in non-annotation approaches is typically derived by analyzing transformation queries as well as the involved input and output data bases, whereas in annotation-based approaches, additional information is applied to a transformation query so that the resulting (output) contains both the originally queried data as well as additional information [17]. Provenance information can be derived by analyzing the additional information created as a result of the augmentation and execution of the transformation query.

Inference According to [24], inferencing on RDF/S entailment rules can be classified into inferencing on the transitive closure of the `rdfs:subClassOf` and `rdfs:subPropertyOf`-properties as well as on class memberships by analyzing `rdfs:range` and `rdfs:domain`-properties. Most methods use *recursive algorithms* for computing transitive closures and store these rules in database views as well as *production rule systems* that discover new facts using forward or backward chaining [24, 34]. However, reasoning over RDF/S and in particular over OWL ontologies is rather complex wherefore most RDF stores pre-compute much of the entailments (forward chaining) [34]. In general, inference can be performed in advance (*eager evaluation*) reducing query evaluation times while occasionally increasing the amount of stored triples dramatically—even for computing the full entailments of OWL Lite ontologies [34]—or at query runtime *(lazy evaluation*) where matching entailments are evaluated when a query is issued reducing possible query entailments as well as query processing times [24]. However, a combination for forward and backward chaining can be used to gain the advantages of both inferencing techniques.

Backup and Restore Seamless backup and restore functionality is a requirement for applications where data needs to be secured, and data loss needs to be avoided. In many situations is desirable that backup processes can be performed while the database is still operational, while in other application scenarios it might be acceptable that the database can be shut down while backup is performed.

5.2 Analysis

Transactions Support for transactions had been removed from the current release (version 1.0.5 as of 18th November 2010) of *4store* and *5store* because of incompatibilities with the main index[34]. In *ARC*, bundled SPARQL INSERT and DELETE queries as a form of transactions are not supported. In *AllegroGraph*'s version 4.1, transactions are built on the ACID properties and realized through transaction logs, where modification operations are recorded in dedicated logs before they are executed[35]. A further optimization is that updates to database files are performed on a checkpoint basis, that is, the triple store is updated periodically and commits are only reflected in the transaction log and applied to in-memory graphs for performance purposes. *Jena TDB* does not provide explicit support for transactions, leaving these issues to the application level, while *Jena SDB* reuses the transaction model from the underlying relational data base. *Kowari* and *Mulgara* employ the XA Triplestore that offers full transaction support and a transaction-safe storage infrastructure for RDF data. In addition, *Mulgara* extends transaction support using the Java Transaction API (JTA) to provide transaction-consistent multi-query support and access to the two-phase commit protocol internally used. In *Oracle 11g*, all transactions fully comply with the ACID properties[36]. *BigOWLIM* supports transactions on a 'read committed' isolation level, meaning that write operations will not impact concurrent query evaluation unless the entire transaction they belong to is successfully committed [32]. *Semantics Platform* is based on MS SQL Server and reuses transaction functionality from

[34] `http://groups.google.com/group/4store-support/browse_thread/thread/a6c5fbd72e819c3a?pli=1`
[35] `http://www.franz.com/agraph/support/documentation/v4/warm-standby.html#header2-16`
[36] `http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/transact.htm#CNCPT117`

		Functional Features									
		Transactions	**Query Languages**	**Full-text Queries**	**Batch Ingest**	**(Selective) Replication**	**Versioning**	**Provenance**	**Named Graphs**	**Inference**	**Backup and Restore**
Native Stores	**4Store**	No	SPARQL	SPARQL regex	Files	Segment Replication	No	Graph URI	Quads	no	Database / File system
	5Store	No	SPARQL	SPARQL regex	Files	Segment Replication	No	Graph URI	Quads	no	Database / File system
	AllegroGraph	Transaction Logs	SPARQL	free text-indicies	Files & Programmatically	Database level	Database level	Graph URI	5-Tuples	Spatio-temporal, RDFS	Yes
	Jena TDB	No	SPARQL with Extensions	Yes (Lucene)	Files, SPARQL INSERT	No	No	Graph URI	Quads	RDFS, OWL, DAML, Rules	No
	Kowari	Yes	iTQL, RDQL	Yes (Lucence)	iTQL load	No information available	No information available	Graph URI	Quads	Rules	Database
	Mulgara	Yes	iTQL, SPARQL via JRDF API, REST	Yes	iTQL load - Resolvers and Content Handler	No information available	No	Separate Inference model	Quads	RDFS, Rules (Krule)	Database
	OWLIM	Yes	SPARQL, SeRQL	Yes	Files	Database level	No	Graph URI	Quads	Rules	No
	Talis Platform	No	SPARQL	Yes	HTTP POST/PUT	Internally	Versioned Updates using Changesets	No	Partially	No	Yes, snapshots
DBMS-backed Stores	**YARS2**	No	SPARQL - N3Q	yes	HTTP PUT	No	No	No Information available	Quads	External OWL reasoner	No Information available
	ARC	No	SPARQL, SPARQLScript	No	SPARQL INSERT	Yes - explicit command	No	Graph URI	Quads	via plugins	Database Dump
	Jena SDB	Yes	SPARQL with Extensions	Yes (Lucene)	Files, SPARQL INSERT	No	No	Graph URI	Quads	RDFS, OWL, DAML, Rules, Custom reasoner	Database
	Oracle 11g	Yes - full transaction support	SPARQL, SQL	Yes	Yes	Yes	Tables, Models	Yes (specific tables)	Generic	RDFS, OWL, OWL 2, Rules	Yes
	Semantics Platform	Yes	SPARQL, SQL	?	SPARQL INSERT	Database level	?	Graph URI	Quads	Rules	Database
Hybrid Stores	**RedStore**	No	SPARQL	No	Files	No	No	Graph URI	Quads	No	Database
	Sesame	Yes	SPARQL, SeRQL	Yes (Lucene)	Files, API	No	No	Graph URI	Quads	RDFS, Custom reasoner	Database / File system
	Virtuoso	Yes	SPARQL, SQL	Yes	Files, SPARQL INSERT	Snapshot, Transactional	No	Graph URI	Quads	RDFS, OWL (limited)	Yes
	BigData	yes	SPARQL, SeRQL	Yes	Files	?	Yes	Statement identifiers, Named Graphs	Quads	RDFS, OWL (limited)	Database / File system

Figure 5: Existing RDF Stores and their functional features

this underlying database. *Redstore*, being a lightweight RDF storage solution, does not provide transaction support. *Sesame*, on the other hand, provides full transaction support through its SAIL API; however, it depends on the underlying concrete storage module whether transactions are guaranteed or not. *Virtuoso* offers full ACID-compliant transaction support as well as a distributed two-phase commit transaction model. *Bigdata* provides an optional MVCC-based optimistic transaction model and provides built-in resolution strategies for conflicting updates [45].

Query Languages SPARQL is supported by most analyzed solutions, including *AllegroGraph*, *Jena TDB*, *Jena SDB*, *Oracle 11g*, *OWLIM*, *Semantics Platform*, and *Redstore*. *4store*, *5store*, and *Redstore* support the execution of SPARQL queries either via a command line client, or alternatively via a SPARQL HTTP protocol server instance, which can be started optionally. *Kowari* incorporates the two query languages interactive Tucana Query Language (iTQL)[37] and the RDF Data Query Language (RDQL) [43]. *Mulgara* provides support for TQL which provides a high-level interface for executing commands on database level, and offers SPARQL support via external (e.g. JRDF) APIs[38] or a REST interface[39]. *ARC* extends its support for SPARQL with SPARQLScript, therefore providing a script-like syntax for queries and several functional extensions, like query chaining, variable assignments, structured variables, loops, and templating. The *Oracle 11g* database provides full SPARQL support through connectors for Jena, Sesame, and Joseki, or alternatively by extending SQL with SPARQL-like clauses. *Jena* provides with *ARQ*[40] a dedicated SPARQL processing engine, which extends the language with a number of features, such as arbitrary length property paths, property functions, and aggregates. Jena's *Joseki*[41] component exposes these SPARQL processing capabilities to the outside via the SPARQL/HTTP protocol. These components can be used with *Jena TDB* as well as *Jena SDB* stores. *OWLIM*, *Sesame*, and *Bigdata* (through its SAIL interface) support, in addition to SPARQL, evaluation of queries formulated in the *SeRQL* language, while *Virtuoso* provides access to data via SPARQL as well as SQL.

Full-text Queries *4store*, *5store*, and *Redstore* do not offer full-text queries natively due to the lack of an appropriate index for full text search [22]. Instead the SPARQL *regex* function provides such functionality whereas true full-text query support is offered via an additional non-standard extension. *AllegroGraph* supports multiple configurable free-text indices, which are based on a locality-optimized *Patricia trie* and allows for wildcard and fuzzy searches[42]. Jena provides full-text query capabilities through the possibility to couple the RDF store with a Lucene index, which is then accessed via special predicates in SPARQL queries[43]. A similar approach is taken by *Sesame* through the *LuceneSail* component[44]. *Kowari* uses the Lucene search engine to index string literals and XML data [20]. *Mulgara* also offers full text search functionality. Similarly, full-text query functionality is also offered by the *Talis Platform*. *ARC* does not provide full-text queries; however they can be implemented as a plug-in-based custom extension. *Oracle 11g* offers full-text queries via Oracle Text that employs fast and accurate full-text retrieval technology that indexes any document or textual content, which can be searched based on their textual content, metadata, or attributes[45]. *OWLIM* supports full-text queries through special RDF predicates that are evaluated as part of query execution [32]. Similarly, *Virtuoso* and *Bigdata* are equipped with full-text indices that can be queried using special predicates in SPARQL queries.

Batch Ingest Many solutions provide bulk import of RDF data from files in various formats, including *4store*, *5store*, *AllegroGraph*, *Jena TDB* and *SDB*, *OWLIM*, *Redstore*, *Sesame*, *Virtuoso*, and *Bigdata*. Alle-

[37] iTQL: `http://docs.mulgara.org/tutorial/itql.html`
[38] cf. `http://lists.mulgara.org/pipermail/mulgara-general/2007-July/000144.html`
[39] `http://www.mulgara.org/trac/wiki/RESTInterface`
[40] `http://jena.sourceforge.net/ARQ/`
[41] `http://joseki.sourceforge.net/`
[42] `http://www.franz.com/agraph/support/documentation/v4/text-index.html`
[43] `http://jena.sourceforge.net/ARQ/lucene-arq.html`
[44] `http://gnowsis.opendfki.de/wiki/LuceneSail`
[45] `http://download.oracle.com/docs/cd/E11882_01/server.112/e16508/cncptdev.htm#CNCPT1537`

groGraph also provides programmatic support for importing RDF triples. Other approaches allow to insert bulk data via SPARQL INSERT queries, like *ARC*, *Jena TDB*, *Semantics Platform*, and *Virtuoso*. RDF data can be loaded into *Kowari* and *Mulgara* using the iTQL `load` command[46]. Additionally, Mulgara allows for including external data into queries via Resolvers and Content Handlers that act as a query layer between the Mulgara transport and storage layers[47]. An HTTP-based approach is chosen by hosted RDF store systems such as the *Talis Platform*, where RDF data can be loaded to a store using `HTTP PUT` or `POST` operations. *Oracle 11g* provides multiple ways to load data into a database such as bulk load using a SQL*Loader, batch load using a Java client, or via an `SQL INSERT` statement[48].

(Selective) Replication *4store* and *5store* provide a replication on segment and storage node level. Selective replication on graph or RDF model level is not supported. *AllegroGraph* and *ARC* support replication on database-level where ARC provides a specific replication command[49] or allow for specifying a finer grained replication where only triples/quads from a custom, SPO(G)-compliant `SELECT` query are considered [1]. *Jena TDB* and *SDB* do not provide designated replication functionality; replication must be performed on the file system or database level, requiring corresponding locking mechanisms. *Boca* supports selective replication of RDF data, where data is selected either on the Named Graph level, or by defining triple patterns with placeholders for the subject, predicate, and object positions. The *Talis Platform* is Software as a Service wherefore replication is performed internally but it is possible for users to make snapshots of their stores and download the triples and binary content as a tar archive. *Oracle 11g* offers a rich set of replication functionality ranging from full database replicas, mass deployment and server-to-server replication, to materialized views. A rich set of database-related objects can be replicated (multi-level replication)[50]. *Semantics Platform* reuses the replication features of the underlying SQL Server; selective replication can be done on the relational model level. *OWLIM* supports the definition of replication clusters, where nodes can fulfill either master or non-master roles; however, no selective replication can be performed using this model [32]. *Sesame* does not provide out-of-the-box replication functionality at the moment. *Virtuoso* provides a range of options for replication, including snapshot replication (one-time replication of data chosen by a query) as well as (optionally bi-directional) transactional replication, allowing other instances to subscribe to updates in selected database tables.

Versioning No explicit versioning support can be found in *4store*, *5store*, *Kowari*, *Mulgara* [19], *ARC*, *OWLIM*, *Jena TDB/SDB*, *Redstore*, and *Virtuoso*. However, there are components available that can be deployed on top of an existing RDF storage solution, e.g., *Graph Versioning System (GVS)*[51]. *Boca* provides access to revisions of named graphs, therefore allowing clients to access historic versions of data. *AllegroGraph* implements a versioning framework on the database level but not on graph level[52]. The *Talis Platform* realizes versioning through an individual concept called *Changesets*[53] that allows for versioned updates (add and remove triples) of the contents stored in the triple store. *Oracle 11g* provides versioning support on RDF model level using a component called Workspace Manager. RDF graph versions are also reflected in queries on version-enabled models. New versions are created for changed data only. *Sesame* does not provide versioning functionality out-of-the-box; however, components like *SemVersion*[54] can bring in such functionality as an add-on. Finally, *Bigdata* allows the database administrator to configure a time interval for which historical data (i.e., deleted or modified RDF resources) will be retained. Within that interval, the database can be rolled back to any point and the corresponding data can be read.

[46] `http://kowari.sourceforge.net/oldsite/417.htm`
[47] `http://www.mulgara.org/trac/wiki/Resolvers`
[48] `http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11828/sdo_rdf_concepts.htm#insertedID7`
[49] `http://arc.semsol.org/docs/v2/store`
[50] `http://download.oracle.com/docs/cd/E11882_01/server.112/e10706/repoverview.htm#i15730`
[51] `http://tech.groups.yahoo.com/group/jena-dev/message/29826`
[52] `http://www.franz.com/agraph/support/documentation/v4/change-history.html`
[53] `http://n2.talis.com/wiki/ChangeSets`
[54] `http://semanticweb.org/wiki/SemVersion`

Provenance *4store*, *5store* and *ARC* use a quad-based model for recording provenance information where the fourth element contains the model or graph URI a triple belongs to. Since these stores initially lack inference support, provenance information regarding whether a statement was explicitly asserted or inferred is not recorded. A common strategy to distinguish inferred from explicitly asserted triple is to store them in separated models or database tables. This strategy is used by *Kowari*, *Mulgara*, *OWLIM*, *Jena TDB/SDB*, *Redstore*, *Semantics Platform*, *Sesame*, *Virtuoso*, and *Bigdata*. Bigdata additionally supports provenance by optionally assigning an identifier to each statement, which allows to make assertions about their provenance. *Oracle 11g* uses multiple tables to record provenance information regarding a stored model including graph IDs and asserted and inferred statements etc.[55].

Named Graphs The concept of named graphs is supported by many systems, although the names used to describe this feature differ. *4store*, *5store*, *AllegroGraph*[56], *Jena TDB/SDB*, *Semantics Platform*, *ARC*, *Sesame*, *Boca*, *Kowari*, *Mulgara*, *OWLIM*, *Virtuoso*, and *Bigdata* actually represent RDF triples as quads, where the fourth element describes the graph a triple belongs to. *YARS2* also uses a quad-based model but introduces the notion of *context* to track or record provenance information in the form of URLs referring to the original data sources. The *Talis Platform* does not provide full user-defined support for named graphs; although a store may contain multiple graphs, but operations for creating or deleting graphs are not offered by its API[57]. *Oracle 11g* uses a generic construct (specific schema) for resembling the concept of named graphs.

Inference RDFS inference is not directly supported in *4store*, but a side-project called *4sreasoner*[58] currently investigates how to include RDFS reasoning in 4store [40]. ARC can be extended via a plug-in architecture that allows reasoning to be implemented via triggers. *BigData* provides limited RDFS and OWL inference. The reasoning engine built in *AllegroGraph* is capable of performing RDFS++ and Prolog reasoning and allows for computing the full entailments of RDF/S and OWL predicates [25]. Temporal reasoning is supported too. *Kowari* and *Mulgara* support rule-based reasoning using different engines. *Mulgara* for instance uses the `Krule` rule reasoning engine[59] for applying a set of pre-defined entailment rules to a base set of RDF model statements. *Talis Platform* does not provide any form of reasoning[60]. Some stores decouple inferencing from their core functions and use external reasoners; *YARS2* uses the external Scalable Authoritative OWL Reasoner (SOAR) engine which incorporates a "*carefully crafted subset of OWL*" [31]. *Oracle 11g* supports reasoning over RDFS and OWL ontologies as well as the definition of custom reasoning rules. *Semantics Platform* provides the means to express custom reasoning rules. *OWLIM* provides means to perform rule-based reasoning over RDF data, whereas rules can either be defined manually using a custom rule language, or be chosen from a set of predefined rule collections (RDFS, OWL-Horst, OWL-Lite) [32]. *Jena* provides an extensible reasoning API, which allows to connect the store to any external reasoner, and provides built-in support for RDFS, OWL, and DAML reasoning[61]. This API can be used in conjunction with any Jena-compliant store, including *Jena SDB* and *Jena TDB*. *Sesame*'s native store provides only RDFS reasoning, however additional reasoning engines can be connected to a Sesame store through the *Sesame Storage and Inference Layer (SAIL)*. *Virtuoso* is equipped with a backward-chaining reasoner that provides support for a subset of RDFS and OWL semantics, including class hierarchies and property equivalence. *Bigdata* employs a mixed inference strategy, where load-time reasoning is applied to certain RDF Schema constructs, while other entailments are computed during query time [45].

Backup and Restore *4store* and *5store* support only manual or script-based backup and restore functionalities; restore is realized either via a Perl file method using the *4s-import* module or by importing

[55] `http://download.oracle.com/docs/cd/E11882_01/appdev.112/e11828/sdo_rdf_concepts.htm#insertedID4`
[56] AllegroGraph uses a 5-tuple model where the fifth element is a unique identifier assigned by the triple store.
[57] cf. `http://groups.google.com/group/n2-dev/browse_thread/thread/a4e5492bbdd1627f`
[58] `http://4sreasoner.ecs.soton.ac.uk/`
[59] `http://docs.mulgara.org/tutorial/krule.html`
[60] `http://weblog.clarkparsia.com/2010/09/23/the-rdf-database-market/`
[61] `http://jena.sourceforge.net/inference/`

previously created *tar balls* to the `/var/lib/4store` directory. *AllegroGraph* supports online full backups containing all data files and information required to restore a database to its backup state[62]. *Jena TDB* does not provide special backup functionality. Storage files can only be backed up on the file system level while no TDB instance is running. Similarly, for *Jena SDB* the backup/restore infrastructure of the underlying relational data base must be used. *Kowari* and *Mulgara* offer specific TQL terminal command support for creating online backups or restoring a database, where a complete snapshot of the triple store can be backed[63]. The *Talis Platform* supports database snapshots which can be created using a snapshot job and can be requested via its `HTTP URI`. *ARC* uses specific PHP commands for streamed store backups by creating a SPOG[64] document from all quads in the store [1]. *Oracle 11g* provides a rich set of backup and restore functionality realized through the Oracle Enterprise Manager Database Control that features multiple manual and automatic backup and recovery strategies[65]. To backup RDF data within the *Semantics Platform*, the underlying MS SQL Server features can be used. *OWLIM*, *Redstore*, *Sesame*, and *Bigdata* provide no special back and restore functionality; data can be secured on the file system or database level, though. *Virtuoso* provides comprehensive support for backups, allowing the administrator to take backups at any time while the database is operational.

5.3 Summary

The question of which RDF storage solution to choose cannot be answered clearly. Even when the requirements of a specific application are clear and well defined, there exists a plentitude of comparable solutions to be chosen from. We can identify a group of powerful database solutions, which are usually designed to handle large amounts of data in a reliable and safe manner. These systems usually provide support for concurrent access through a well-defined transaction model, import and export functionality, sophisticated query processing (including optimization) for different languages, reasoning and inference, backup and restore functionality, and support for replication and synchronization. Examples for this category are *4Store/5Store*, *AllegroGraph*, *Oracle 11g*, *Semantics Platform*, *Virtuoso*, and *Bigdata*. These stores are suited for data-intensive server applications and for enterprise warehouses. A special case is *Talis Platform*, which can be used as an off-site, cloud storage for RDF data.

On the other hand we can observe a number of lightweight implementations that are suitable to be included as RDF library within applications that need to process and store RDF data. Often these solutions provide only in-memory and file-based storage facilities, do not guarantee safe transactions, support a limited set of query languages (only SPARQL is supported by the vast majority of analyzed systems), and do not provide versioning, replication, or inference. Examples include *RedStore*, *YARS2*, and *Jena TDB*.

6 Quantitative Analysis

After having discussed the features and functional characteristics of various RDF storage solutions, we now focus the performance of a selected subset of these solutions w.r.t *load time* and *SPARQL query response time*. We conducted all our experiments in the context of the Europeana[66], which, as we explain in the following, influenced the applied methodology and the datasets and test queries used for the performance evaluation.

6.1 Background and Methodology

Europeana currently aggregates metadata of approximately 15 million items from around 1500 cultural institutions all over Europe. The metadata of these objects are currently represented in XML following

[62] `http://www.franz.com/agraph/support/documentation/v4/backup-and-restore.html`
[63] `http://www.mulgara.org/trac/wiki/BackupRestore`
[64] `http://www.wasab.dk/morten/blog/archives/2008/04/04/introducing-spog`
[65] `http://download.oracle.com/docs/cd/E11882_01/server.112/e10897/backrest.htm#CHDHEJBA`
[66] The European Digital Library: `http://www.europeana.eu/`

the Europeana Semantic Elements (ESE)[67] scheme and stored on the file system. A periodic batch process parses and indexes these metadata using the Apache Lucene-based Solr[68] engine. Since the ESE provides only limited means for expressing semantics, a new, semantically richer, RDF-based format, called the Europeana Data Model (EDM)[69] is currently being developed. As a consequence, appropriate solutions for persisting and retrieving RDF data are required

With this background, we included selected solutions from the previously presented RDF store inventory (see Figure 3), which fulfill the following properties: the store must be (i) under active development (latest release year in or after 2010), (ii) available as open source under a free license, also for production, (iii) installable as a local server on UNIX-based systems, and (iv) provide a SPARQL query interface that supports DESCRIBE queries in order to install Linked Data front-ends such as Pubby[70] on-top of a store. Further, we skipped those stores that were superseded by other stores in experiments conducted by others (see Figure 4). Therefore we included the following stores in our study:

- *Native stores*: 4Store v.1.0.4, Jena TDB v.0.8.9
- *DBMS-backed stores*: Jena SDB v.1.3.3 with MySQL RDB backend
- *Hybrid stores*: Sesame v.2.3.2, OpenLink Virtuoso Open-Source Edition 6.1.2

For conducting the experiments we developed a Ruby-based RDF store performance test framework (`https://github.com/behas/tripleval`), which defines the following generic evaluation procedure that is applied a selected set of RDF stores:

- Iteratively ingest triple blocks, where each block is defined by a given N-Triples file
- Execute the set of predefined queries after each block ingest using random query values extracted from the already ingested triples

This procedure allows us to measure the triple load and query response times at growing triple load levels. For each RDF store, we implemented a wrapper class, which implement the execution of triple-store specific command-line calls for triple ingest and SPARQL query.

All experiments were run three times on a 2x Quad-Core Xeon E5504 2.0 GHz machine, with 48 GB DDR3-1333 ECC RAM, running openSUSE Linux 11.1. We configured each RDF store to persist the ingested data on a RAID-01 cluster of 4 Seagate 600 GB SAS hard disks. We believe that the usage of such a high-end server application is justified in contexts such as Europeana where an investment into hardware of this scale should be possible. We configured all RDF stores — also those that could run in cluster configuration (e.g., 4Store) — to run in single-machine, single-disk mode. Furthermore, we installed all triple stores in standard configuration.

6.2 Dataset and Queries

We used the complete bibliographic metadata set currently available in Europeana for our performance experiments. Since the production metadata are currently available only in the Europeana Semantic Elements (ESE) XML format, we first had to map and convert these data from ESE XML to the RDF-based EDM model. The exact mapping specification is documented at `http://europeanalabs.eu/wiki/EDMPrototypingTask15` and the transformation scripts are available at `https://github.com/behas/ese2edm`. The resulting EDM RDF/XML records, as shown in Figure 6, were then transformed to the N-Triples format, merged into one large file containing 382,629,063, and then split into 38 N-Triples block files, each containing 10,000,000 triples and one N-Triples file containing the remaining 2,629,063 triples. The resulting EDM dataset is available for download upon request at: `http://europeana.mminf.univie.ac.at`.

[67] Europeana Semantic Elements schema. Available at: `http://www.europeana.eu/schemas/ese/`

[68] Apache Solr. Available at: `http://lucene.apache.org/solr/`

[69] `http://www.version1.europeana.eu/c/document_library/get_file?uuid=718a3828-6468-4e94-a9e7-7945c55eec65&groupId=10605`

[70] `http://www4.wiwiss.fu-berlin.de/pubby/`

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dct="http://purl.org/dc/terms/"
  xmlns:ens="http://www.europeana.eu/schemas/edm/" xmlns:ese="http://www.europeana.eu"
  xmlns:ore="http://www.openarchives.org/ore/terms/">

  <ens:PhysicalThing rdf:about="http://www.europeana.eu/resolve/record/00101/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>

  <ore:Aggregation rdf:about="http://id.europeana.eu/00101/aggregation/C400748B5B9DAE7A28181B9AC620F76E19F78940">
    <ore:aggregatedCHO
      rdf:resource="http://www.europeana.eu/resolve/record/00101/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>
    <ens:isShownBy
      rdf:resource="http://baimages.gulbenkian.pt/images/winlibimg.aspx?skey=&amp;doc=130466&amp;img=10866"/>
    <ens:object
      rdf:resource="http://baimages.gulbenkian.pt/images/winlibimg.aspx?skey=&amp;doc=130466&amp;img=10866&amp;res=30&amp;thb=1&amp;pag=1"/>
    <ens:provider>Funda o Calouste Gulbenkian - Portugal</ens:provider>
  </ore:Aggregation>

  <ore:Proxy rdf:about="http://id.europeana.eu/00101/proxy/C400748B5B9DAE7A28181B9AC620F76E19F78940">
    <ore:proxyFor rdf:resource="http://www.europeana.eu/resolve/record/00101/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>
    <ore:proxyIn rdf:resource="http://id.europeana.eu/00101/aggregation/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>
    <ens:type>TEXT</ens:type>
    <dc:description xml:lang="pt">Novamente emendado nesta ltima impress o</dc:description>
    <dc:title xml:lang="pt">Acto do Infante D. Pedro de Portugal, o qual andou as sete partidas do mundo</dc:title>
    <dc:format xml:lang="pt">29, [1] p.</dc:format>
    <dct:extent xml:lang="pt">(21 cm)</dct:extent>
    <dc:creator>Santo Estev o, Gomes de</dc:creator>
    <dc:contributor>Ribeiro, Ant nio lvares</dc:contributor>
    <dc:identifier>130466</dc:identifier>
    <dc:identifier>CALLNUMBER-TC 155</dc:identifier>
    <dc:language>pt</dc:language>
    <dct:issued>1790</dct:issued>
    <dc:publisher>Offic. de Antonio Alvarez Ribeiro</dc:publisher>
    <dc:publisher>Porto</dc:publisher>
    <dc:subject>Porto</dc:subject>
    <dc:type xml:lang="pt">material textual, impresso</dc:type>
    <dc:type xml:lang="en">language materials, printed</dc:type>
  </ore:Proxy>

  <ens:EuropeanaAggregation rdf:about="http://id.europeana.eu/europeana/aggregation/C400748B5B9DAE7A28181B9AC620F76E19F78940">
    <ore:aggregatedCHO rdf:resource="http://www.europeana.eu/resolve/record/00101/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>
    <ore:aggregates rdf:resource="http://id.europeana.eu/00101/aggregation/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>
    <dc:creator>Europeana</dc:creator>
    <ens:landingPage rdf:resource="http://www.europeana.eu/portal/record/00101/C400748B5B9DAE7A28181B9AC620F76E19F78940.html"/>
    <ens:isShownBy rdf:resource="http://baimages.gulbenkian.pt/images/winlibimg.aspx?skey=&amp;doc=130466&amp;img=10866"/>
    <ens:hasView rdf:resource="http://baimages.gulbenkian.pt/images/winlibimg.aspx?skey=&amp;doc=130466&amp;img=10866"/>
    <ens:country>portugal</ens:country>
    <ens:language>pt</ens:language>
  </ens:EuropeanaAggregation>

  <ore:Proxy rdf:about="http://id.europeana.eu/europeana/proxy/C400748B5B9DAE7A28181B9AC620F76E19F78940">
    <ore:proxyFor rdf:resource="http://www.europeana.eu/resolve/record/00101/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>
    <ore:proxyIn rdf:resource="http://id.europeana.eu/europeana/aggregation/C400748B5B9DAE7A28181B9AC620F76E19F78940"/>
    <ens:type>TEXT</ens:type>
  </ore:Proxy>

</rdf:RDF>
```

Figure 6: Sample EDM record.

In Europena RDF stores will mainly take the role of metadata repositories that are accessed by higher-level components to build up the Lucene-based search and retrieval index and to serve EDM records following the Linked Data principles. At the time of this writing it seems that RDF stores must provide good performance for two types of queries:

- SPARQL DESCRIBE queries for instances of `ens:EuropeanaAggregation` objects (e.g., `DESCRIBE <http://id.europeana.eu/europeana/aggregation/C400748B5B9DAE7A28181B9AC620F76E19F78940>`) This type of queries is frequently applied by Linked Data front-ends, which must deliver RDF data for URIs that are being de-referenced via their HTTP URIs.
- SPARQL SELECT queries for subjects, which have certain resources or labels as object. As a representative for this query type, we set up a query that retrieves all `ens:EuropeanaAggregation` subjects that have a certain `ens:landingPage` as object (e.g., `SELECT ?x WHERE ?x ens:landingPagemate <http://www.europeana.eu/portal/record/00101/C400748B5B9DAE7A28181B9AC620F76E19F78940.html>`).

To evaluate the response times of these queries our test harness caches a randomly selected subset of the ingested data, such as URIs of `ens:EuropeanaAggregation` objects and `ens:landingPage` objects and generates random queries without proceeding triple store query interactions, which should lead to unbiased query response results.

For all results we applied the same time-out strategies: triple load stops when ingesting a single N-Triples file (10,000,000 triples) takes more than 2 hours. SELECT and DESCRIBE queries have a timeout of 10 seconds.

6.3 Results

Figure 7 illustrates the cumulative load time for the complete Europeana EDM/RDF dump and shows significant differences in triple load performance. Ingesting the complete EDM dataset into 4Store takes 1.5 hours in total; doing the same with JenaTDB or Virtuoso takes longer, but is still in an acceptable time range. The load time of the MySQL-backed JenaSDB installation grows exponentially and is not suitable for datasets as large as the Europeana EDM dataset. With Sesame we encountered the problem that data ingestion is, to the best of our knowledge, only possible via HTTP, which resulted in a load time greater than two hours for the first N-Triples file and hence a time-out.

Figure 8 shows the performance of SPARQL DESCRIBE queries executed against different RDF stores at various triple load levels. We can observe that Virtuoso answers DESCRIBE queries within a second unless it performs system-internal tasks that delay the response time. Up to 200 million triples 4Store provides the fastest answers but the response times grow significantly with increasing triple load. JenaTDB achieves almost constant 2 second response times; JenaSDB behaves similar but is not applicable for triple loads larger than 100 million triples.

Figure 9 shows the performance of SPARQL queries that retrieve subjects by object labels, executed against different RDF stores at various load levels. We can observe that Virtuoso performs best on this type of queries and delivers answers in less than half a second. JenaSDB responses take 2 seconds in average but scale up to the Europeana triple size, which is not the case for JenaSDB.

These results show that certain RDF stores, notably OpenLink Virtuoso and 4Store, can hold the complete Europeana EDM data set. They can serve as repositories for metadata expressed in the Europeana Data Model (EDM) and allow to set up Linked Data front-ends on-top of them. It is however, yet unclear how these stores perform under heavy load and with increasing query frequency. Therefore, Linked Data services built with one of these stores must still be consider experimental and should not replace existing data storages or search and retrieval facilities.

7 Summary and Conclusions

In this report, we gave a detailed overview on existing RDF store solutions, analyzed their functional and non-functional features, summarized the outcomes of other, previously carried out studies, and conducted a

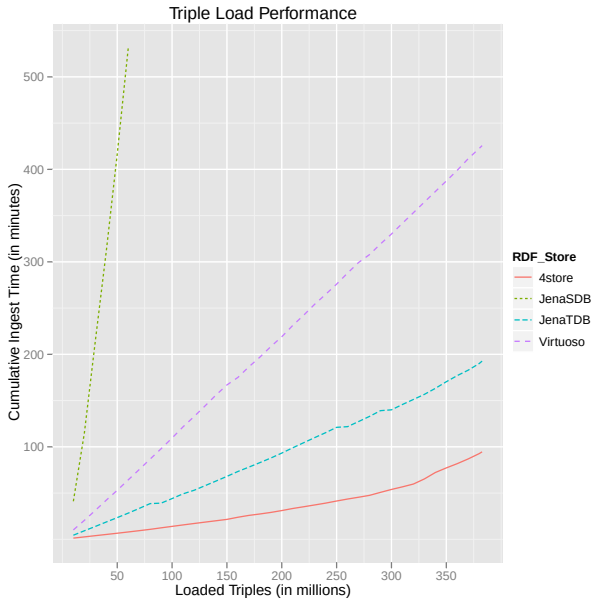

Figure 7: Cumulative Load Time.

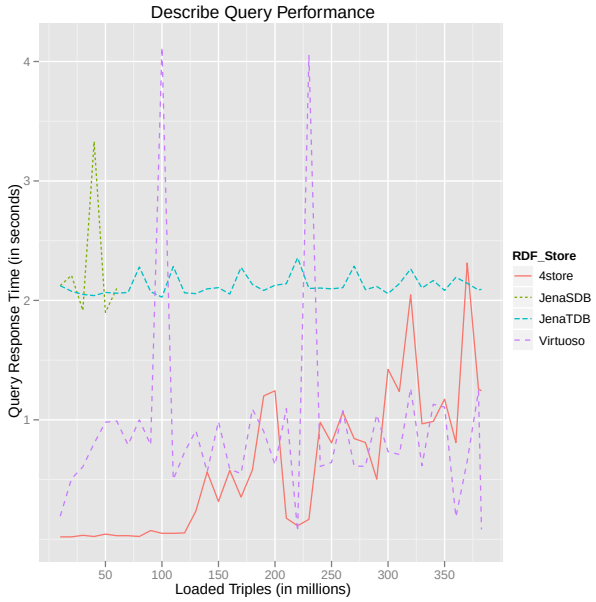

Figure 8: SPARQL DESCRIBE query performance.

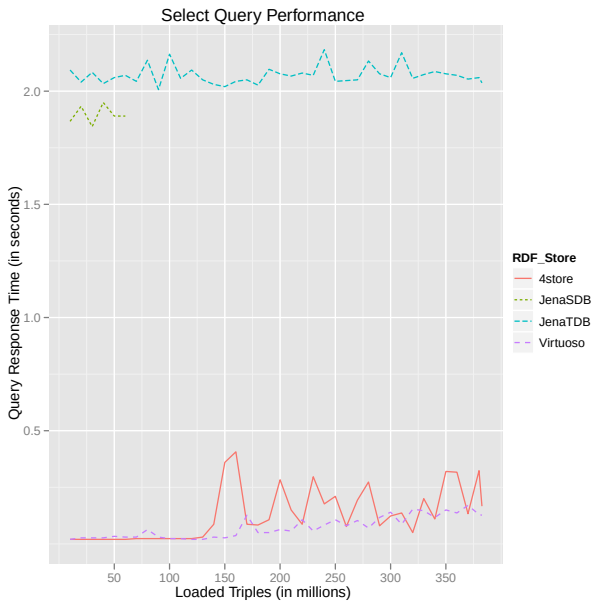

Figure 9: SPARQL SELECT query performance.

performance evaluation on a subset of existing RDF stores w.r.t to load and query time.

The results of this study show that native RDF solutions can deal with the Europeana data volume and answer SPARQL queries, which are relevant in the context of Europeana, in a time-range that is acceptable when RDF stores are solely used as Linked Data enabling metadata stores and not as a replacement for existing full-text search engines such as Apache Lucene.

Acknowledgements

This report presents work done for the best practice network Europeana*Connect*. Europeana*Connect* is funded by the European Commission within the area of Digital Libraries of the eContent*plus* Programme and is lead by the Austrian National Library. We want to thank Jan M. Stankovsky for the technical support in setting up the various RDF store instances.

References

[1] Arc developer documentation, 2010. Available at: `http://arc.semsol.org/docs/v2/store`.

[2] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Using the barton libraries dataset as an rdf benchmark. Technical Report MIT-CSAIL-TR-2007-036, MIT, 2007.

[3] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *The VLDB Journal*, 18(2):385–406, 2009.

[4] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-peer Content Distribution Technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[5] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify - light-weight linked data publication from relational databases. In *18th International World Wide Web Conference*, pages 621–621, April 2009.

[6] Sören Auer and Heinrich Herre. A Versioning and Evolution Framework for RDF Knowledge Bases. In Irina Virbitskaite and Andrei Voronkov, editors, *Ershov Memorial Conference*, volume 4378 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2006.

[7] James Bailey, François Bry, Tim Furche, and Sebastian Schaffert. Web and semantic web query languages: A survey. In Norbert Eisinger and Jan Maluszynski, editors, *Reasoning Web*, volume 3564 of *Lecture Notes in Computer Science*, pages 35–133. Springer, 2005.

[8] Schandl Bernhard. Tripcel: Exploring rdf graphs using the spreadsheet metaphor (poster and demo). In *8th International Semantic Web Conference (ISWC 2009)*, Washington, DC, USA, 10 2009.

[9] Philip A. Bernstein and Nathan Goodman. Concurrency Control in Distributed Database Systems. *ACM Comput. Surv.*, 13(2):185–221, 1981.

[10] Chris Bizer and Daniel Westphal. Developers guide to semantic web toolkits for different programming languages. Available at: `http://www4.wiwiss.fu-berlin.de/bizer/toolkits/`.

[11] Christian Bizer, Richard Cyganiak, and Olaf Hartig. Ng4j - named graphs api for jena, 2010. Available at: `http://www4.wiwiss.fu-berlin.de/bizer/ng4j/`.

[12] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3), 2009.

[13] Christian Bizer and Andreas Schultz. Benchmarking the performance of storage systems that expose sparql endpoints. In *In Proceedings of the ISWC Workshop on Scalable Semantic Web Knowledgebase*, 2008.

[14] J. Broekstra. Storage, querying and inferencing for semantic web languages. Phd-thesis, Vrije Universiteit - Amsterdam, 2005.

[15] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM.

[16] Jeremy J. Carroll and Patrick Stickler. Trix: Rdf triples in xml. Technical Report HPL-2004-56, HP Labs, May 2004.

[17] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends databases*, 1(4):379–474, 2009.

[18] Orri Erling and Ivan Mikhailov. RDF Support in the Virtuoso DBMS. In Sören Auer, Christian Bizer, Claudia Müller, and Anna V. Zhdanova, editors, *CSSW*, volume 113 of *LNI*, pages 59–68. GI, 2007.

[19] Sebastian Eulau. Verteilte speicherung von daten und metadaten in einem föderierten semantischen produktinformationssystem, December 2009.

[20] Paul Ford. A first look at the kowari triplestore, June 2004. Available at: `http://www.xml.com/pub/a/2004/06/23/kowari.html`.

[21] J. Gray. The Transaction Concept: Virtues and Limitations. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 144–154. IEEE Computer Society, 1981.

[22] Steve Harris, Nicholas Lamb, and Nigel Shadbol. 4store: The design and implementation of a clustered rdf store. In *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009), co-located with ISWC2009*, Washingtion DC, USA, October 25-29 2009. Available at: `http://4store.org/publications/harris-ssws09.pdf`.

[23] Bernhard Haslhofer and Bernhard Schandl. The OAI2LOD Server: Exposing OAI-PMH Metadata as Linked Data. In *International Workshop on Linked Data on the Web (LDOW2008)*, 2008.

[24] Alice Hertel, Jeen Broekstra, and Heiner Stuckenschmidt. RDF Storage and Retrieval Systems. On-line, 2008.

[25] Franz Inc. Allegrograph 4.0.5d introduction, 08 2010. Available at: `http://www.franz.com/agraph/support/documentation/v4/agraph-introduction.html`.

[26] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation 10 February 2004)*. World Wide Web Consortium, 2004.

[27] Ryan Lee. Scalability report on triple store applications. Technical report, Massachusetts Institute of Technology, 2004.

[28] Baolin Liu and Bo Hu. An Evaluation of RDF Storage Systems for Large Data Applications. In *SKG*, page 59. IEEE Computer Society, 2005.

[29] Zhihong Lu and Kathryn S. McKinley. Partial Replica Selection Based on Relevance for Information Retrieval. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 97–104, New York, NY, USA, 1999. ACM.

[30] Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. Benchmarking fulltext search performance of rdf stores. In *6th Annual European Semantic Web Conference (ESWC2009)*, pages 81–95, June 2009.

[31] Vassil Momtchev, Brahmanda Sapkota, Andreas Harth, Omair Shafiq, and Atanas Kiryakov. Storage performance evaluation. Deliverable, FP6 – 02734, www.tripcom.org/docs/del/D1.4.pdf, December 2009.

[32] Ontotext. BigOWLIM User Guide Version 3.3. Technical report, Ontotext AD, 2010. Available at `http://www.ontotext.com/owlim/BigOWLIMUserGuide_v3.3.pdf`.

[33] Ontotext. OWLIM Primer. Technical report, Ontotext AD, 2010. Available at `http://www.ontotext.com/owlim/OWLIM_primer.pdf`.

[34] Alisdair Owens. An investigation into improving rdf store performance, 2009.

[35] Prasanna Padmanabhan, Le Gruenwald, Anita Vallur, and Mohammed Atiquzzaman. A S of Data Replication Techniques for Mobile Ad hoc Network Databases. *The VLDB Journal*, 17(5):1143–1164, 2008.

[36] Sandro Reichert. A secure data repository for semantic federation of product information. In *iiWAS '09: Proceedings of the 11th International Conference on Information Integration and Web-based Applications and Services*, pages 745–749, New York, NY, USA, 2009. ACM.

[37] Rohloff, Dean, Emmons, Ryder, and Sumner. An Evaluation of Triple-Store Technologies for Large Data Stores. 2006.

[38] Satya Sanket Sahoo, Olivier Bodenreider, Pascal Hitzler, Amit P. Sheth, and Krishnaprasad Thirunarayan. Provenance context entity (pace): Scalable provenance tracking for scientific rdf data. In Michael Gertz and Bertram Ludäscher, editors, *SSDBM*, volume 6187 of *Lecture Notes in Computer Science*, pages 461–470. Springer, 2010.

[39] Yasushi Saito and Marc Shapiro. Optimistic Replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[40] Manuel Salvadores, Gianluca Correndo, Temitope Omitola, Nicholas Gibbins, Steve Harris, and Nigel Shadbolt. 4s-reasoner: Rdfs backward chained reasoning support in 4store. In *Web-scale Knowledge Representation, Retrieval, and Reasoning (Web-KR3)*, September 2010.

[41] Bernhard Schandl. TripFS: Exposing File Systems as Linked Data. In *Linked Open Data Triplification Challenge, Graz, Austria*, 2009.

[42] Michael Schmidt, Thomas Hornung, Norbert Küchlin, Georg Lausen, and Christoph Pinkel. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In *ISWC '08: Proceedings of the 7th International Conference on The Semantic Web*, pages 82–97, Berlin, Heidelberg, 2008. Springer-Verlag.

[43] Andy Seaborne. RDQL - A Query Language for RDF. W3c member submission, Hewlett Packard, January 2004. See `http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/`.

[44] Sang Hyuk Son. Replicated Data Management in Distributed Dtabase Systems. *SIGMOD Rec.*, 17(4):62–69, 1988.

[45] SYSTAP. Bigdata Architecture Whitepaper. Technical report, SYSTAP, LLC, 2009. Available at `http://www.bigdata.com/whitepapers/bigdata_whitepaper_10-13-2009_public.pdf`.

[46] W3C. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Semantic Web Activity - RDF Core Working Group, February 2004. Available at: `http://www.w3.org/TR/rdf-schema/`.

[47] W3C. *SPARQL Query Language for RDF*. W3C Semantic Web Activity – RDF Data Access Working Group, 2008.

[48] W3C. *OWL 2Web Ontology Language*. W3C Semantic Web Activity – OWL Working Group, 2009.

[49] David Wood. Kowari: A platform for semantic web storage and analysis. In *In XTech 2005 Conference*, pages 05–0402, 2005.

[50] Dimitris Zeginis, Yannis Tzitzikas, and Vassilis Christophides. On the Foundations of Computing Deltas between RDF Models. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon J B Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, pages 631–644, Berlin, Heidelberg, November 2007. Springer Verlag.