7 Storing the Semantic Web: Repositories

Atanas Kiryakov · Mariana Damova
Ontotext AD, Sofia, Bulgaria

John Domingue, Dieter Fensel & James A. Hendler (eds.), *Handbook of Semantic Web Technologies*,
DOI 10.1007/978-3-540-92913-0_7,

Abstract: Semantic repositories are database management systems, capable of handling structured data, taking into consideration their semantics. The Semantic Web represents the next-generation Web of Data, where information is published and interlinked in a way, which facilitates both humans and machines to exploit its structure and meaning. To foster the realization of the Semantic Web, the World Wide Web Consortium (W3C) developed a series of metadata, ontology, and query languages for it. Following the enthusiasm about the Semantic Web and the wide adoption of the related standards, today, most of the semantic repositories are database engines, which deal with data represented in RDF, support SPARQL queries, and can interpret schemas and ontologies represented in RDFS and OWL. Naturally, such engines take the role of Web servers of the Semantic Web.

This chapter starts with an introduction to semantic repositories and discussion on their links to several other technology trends, including relational databases, column-stores, and expert systems. As the most distinguishing quality of the semantic repositories is reasoning, an overview of the strategies for the integration of inference in the data management life cycle is presented. An overall view of the mechanics of the engines is provided from the perspective of a conceptual framework that reveals all their tasks and activities (e.g., storage and retrieval) along with the factors that impact their performance (e.g., data size and complexity). A review of several design issues, including distribution, serves as a basis for understanding the different implementation approaches and their implications on the performance of semantic repositories. Several of the most popular benchmarks and datasets, which are often used as measuring sticks for the performance of the engines, and few of the outstanding semantic repositories, are presented along with the best published evaluation results.

The advantages and the typical applications of semantic repositories are presented focusing on two usage scenarios: reasoning with and the management of linked data (a popular trend in the Semantic Web) and enterprise data integration. The chapter ends with some considerations regarding the future development of semantic repositories and design topics like adaptive indexing and interoperability patterns.

7.1 Introduction

The Semantic Web creates a wealth of data, where information is given well-defined meaning and computers are better able to work with it; a vast collection of structured data are published and interlinked together to form a Web of Data. The potential for increasing knowledge availability and the ability of machines to effectively work with it is enormous. Managing the data on the Web, however, represents a tremendous challenge, considering their size and complexity, the anticipated number of requests, and the desirable response times. This opens the story of semantic repositories (SR) – tools that combine characteristics of database management systems (DBMS) and inference engines to support efficient manipulation of Semantic Web data. Semantic repositories take the role of Web servers, providing access to the Web of Data.

The presence of such data management systems, able to hold, interpret, and serve requests and queries from multiple users against massive amounts of data, is an indispensable step toward the realization of the vision and the potential of the Semantic Web. They evolve dynamically, racing to extend human's and computer's capabilities to deal with structured data. The implementation of such engines requires advancement of the frontiers in two fields: databases and reasoning. Each new development allows loading more data, dealing with more comprehensive schemas and ontologies, and answering more complex queries in less time. As in mountain climbing, each new achievement uncovers new opportunities and challenges.

A story related to UNIPROT illustrates what it feels like to be a technology pioneer. UNIPROT is the most extensive and the most popular public database about protein-related information (see Sect. 7.5.2). Back in 2006, Ontotext established LifeSKIM – a small team to work on life science applications. One of the first tasks of the LifeSKIM people was to load UNIPROT in the OWLIM semantic repository (see Sect. 7.6.4). The first attempt ended up with a "stack overflow" error – the corresponding version of the engine was not prepared for a group of 3,000 concepts, related through the transitive `owl:sameAs` property. Once this problem was fixed, it became clear that heavy usage of `owl:sameAs` alignment can ruin the performance, as discussed in Sect. 7.2.1.4. The necessary optimizations in the indices were made; latter on those proved to be very useful for linked data management and data integration (see Sect. 7.8.1). It was finally possible to load UNIPROT and to perform materialization against the relevant fragment of OWL. The last surprise came when a person from the LifeSKIM team defined several sample queries and tried to make sense of the results. Some of them were definitely incorrect, and investigation was carried out to find the source of the problem. Finally, it appeared that there was a small bug in the UNIPROT schema, which remained unspotted by its developers and its numerous users, because no one before was able to interpret this aspect of its semantics.

The UNIPROT encounter with OWLIM (the semantic repository) is an example of how a dataset can drive an improvement in the engine and vice versa. Practically, semantic repositories can be seen as track-laying machines, which extend the reach of the data railways: Each previous step is only possible on top of the results of the previous one. These railways change the data-economy of entire domains and areas, by allowing larger volumes of more complex data to be handled at lower cost. This makes the topics around performance, capabilities, and development of semantic repositories both fascinating and intriguing.

This chapter deals with the foundations of semantic repositories and attempts to provide a roadmap toward their major characteristics, related design and performance issues, the state of the art in the field, and future directions. The major objectives are:

- To clarify the principles of operation of semantic repositories and the benefits of their usage
- To explain the facets of their performance, because their understanding of this is a key factor for the successful adoption of semantic repositories

The remainder of this section starts with discussion on the recent history and the "political economy" of the field (Sect. 7.1.1), the role of the semantic repositories and

their typical usage (Sect. 7.1.2), and continues with a quick introduction to several related subjects: RDF data models (Sect. 7.1.3) and linked data basics (Sect. 7.1.4).

Section 7.2 discusses reasoning within the semantic repositories: the strategies for the integration of inference in the data management life cycle, with their advantages and related problems, as well as, ontology languages and dialects suitable for inference in such scenarios. In Sect. 7.3, a conceptual framework for the understanding of all tasks and activities of a semantic repository is provided. This framework addresses also the factors that impact its performance and the different aspects or dimensions of this performance. Section 7.4 goes into a range of practical issues related to the design of today's semantic repositories, including analysis of typical server configurations and various approaches for the distribution of the repositories. Next, in Sect. 7.5, several of the most popular benchmarks and datasets are presented – those are often used as measuring sticks for the performance of the semantic repository engines. At this point, the floor is set to discuss a few of the most prominent semantic repositories (Sect. 7.6) and present an overview of the best published evaluation results (Sect. 7.7).

Section 7.8 outlines three typical applications of semantic repositories:

- FactForge: a search engine, serving as a gateway, facilitating the usage of the central datasets in the Linking Open Data cloud
- LinkedLifeData: a platform for semantic data integration in life sciences domain
- BBC's website for World Cup 2010, which demonstrates how semantic technologies can enable optimizations in the publishing process

Finally, a list of related resources is provided (Sect. 7.9) and discussion on a few of the key issues related to the future development of the semantic repositories (Sect. 7.10), along with design considerations (like adaptive indexing) and interoperability patterns, which are likely to be adopted in order to resolve some of the bottlenecks of today's semantic technology applications.

7.1.1 Inspiring Vision + Standards + Business Demands = Rapid Development

Semantic repositories are still in the initial phase of rapid upward development. Since 2004, every couple of years, the engines have been getting an order of magnitude faster and more scalable. Such development has been enabled by several factors:

- Standards – standardization efforts related to the Semantic Web, most notably RDF(S), OWL, and SPARQL, provided a solid ground for development and good minimal levels of interoperability.
- Benchmarks – the performance of a semantic repository is a multidimensional phenomenon; the performance with respect to different tasks depends on a range of factors and parameters. Making sustainable progress in the performance of the engines requires adequate measuring sticks and methods, namely, benchmarks and evaluation

methodologies. They can provide clear indication about the cost, the applicability, and the benefits of each new optimization, approach, or technique with respect to the different aspects of the performance.

- Hardware – while a $10,000 server was needed to load one billion statements in 2006, three years later, this task was achievable on a $2,000 workstation. Running semantic repositories on commodity hardware comparable with the environments on which relational databases are run is considered as a serious asset.
- Performance engineering – the understanding about the optimal server configurations for different tasks, types of data, and query loads is far better today than 5 years ago.
- Data integration demands – globalization and the consolidation of the business increased the demand for data integration approaches, which scale efficiently to tens and hundreds of data sources. In the life sciences, the availability of hundreds of public databases, efficient access to which can facilitate medical research and drug development, served as a huge stimulus for the adoption of semantic repositories.
- Linked data enthusiasm – the emergence of the Web of linked data (see Sect. 7.1.4) provoked interest in industry (and many governments) to use public data and to publish databases in RDF, according to the linked data principles [5].

In summary, the growing demand for the management of heterogeneous and dynamic structured data met with a technology stream that already offers robust tools. The latter are backed by a vision, community, and standards that ensure its steady development. The overheads related to the adoption of higher-level data management paradigm became bearable in the light of the increasing hardware capabilities.

7.1.2 Semantic Repositories = Inference Engines + Column-Stores

Semantic repositories are engines similar to database management systems (DBMS).Their major functionality is to support efficient storage, querying, and management of structured data. The major differences with the DBMS can be summarized as follows:

- They use ontologies as semantic schemas, which allows them to automatically reason about the data.
- They work with generic physical data models, which allows them to easily adopt updates and extensions in the schemas, that is, in the structure of the data.

Functionally, semantic repositories are essentially DBMS that can interpret the data. Based on the semantics of the schemas, they can infer implicit facts and consider them in the process of query evaluation.

As an illustration of this, *Fig. 7.1* provides an example of the representation of simple family relationships (the graph on the right) and the facts that can be inferred from those (indicated with dashed arcs) based on simple entailment rules (presented on the left). Two explicit facts were asserted in the repository (the solid orange arcs): "Ivan is

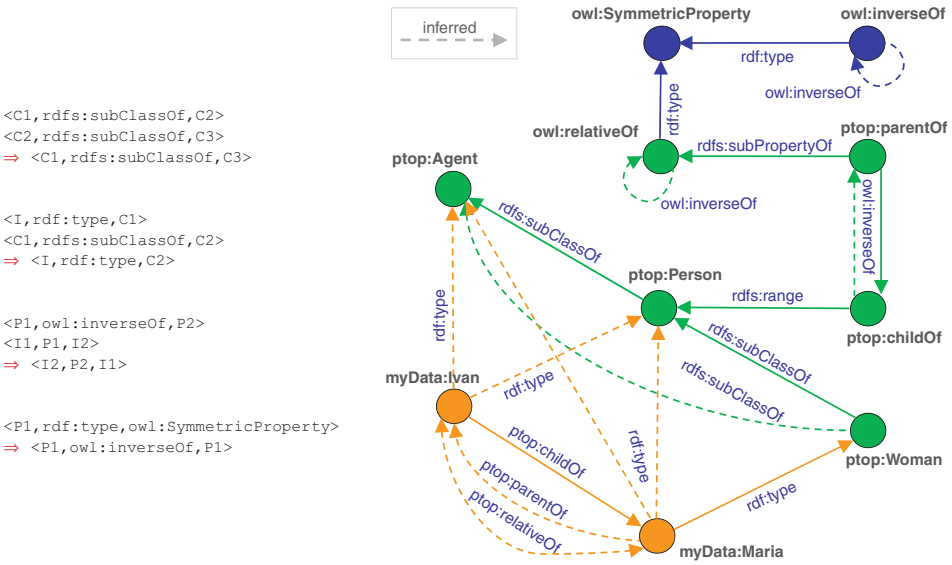

Fig. 7.1
Data representation and interpretation in semantic repositories

a child of Maria" and "Maria is a woman." The repository was able to infer several new facts (the dashed orange arcs), for example, that Maria is parent of Ivan, that they are relatives to each other, that Maria is a person and an agent. Those inferences were made on the basis of the specific data schema that was used. The interpretation of both the specific instance data and the schema was possible based on the semantics of the system primitives used (the part of the graph in blue), encoded in the entailment rules (presented on the left).

The most direct benefit of the ability of the semantic repository to interpret the data is that it can evaluate queries (or more generally retrieval requests) in a much "cleverer" and more flexible manner. For instance, in the above example, the repository will be able to return Ivan as a result of a request asking for all relatives of Maria (e.g., by the retrieval pattern "*Maria relativeOf ?x*"). In this case, the query pattern is more general than the explicit fact asserted (being a relative is a general case of being a child). Moreover, the relation in the query goes in the inverse direction (the fact was "*Ivan ?p Maria*," but it matches request for "*Maria ?p ?x*").

The above query example is an illustration that semantic repositories allow query variation with respect to the level of generality and the direction of the relation between two entities. More generally, the expression of the information need in the query no longer has to match the syntax of assertion of the data. This enables a whole new range of information access scenarios, where the person who formulates the query is not aware of the details of the schemas that were used for the encoding of the data. This is only possible in a DBMS where the engine understands (to some extend) the semantics of the data and the semantics of the query and can interpret them in order to find matches

despite syntax variations. This is probably the major difference between the RDF-based semantic repositories and XML-based DBMS. XML is designed to allow interoperability with respect to the syntax of the data, remaining ignorant to its semantics. For instance, in an XML schema, one cannot define inverse, transitive, and symmetric properties as it is possible with RDF (see *Fig. 7.1*). This means that the XML database has no chance to interpret the data and to provide a useful answer to the query in the example given above.

As already mentioned, another principal advantage of the RDF-based DBMS is that they use a generic physical model. The data are represented internally in graph-like data structures like the one depicted on *Fig. 7.1*. *Figure 7.2*, on the other hand, presents a comparison of the representation of the data in relational DBMS (RDBMS, on the left) and the RDF databases (on the right), where each arc from the RDF graph is represented as a triple: subject (the source node), predicate (the property, which determines the type of relation or the attribute), and object (the target node, or the value). A change in the schema (e.g., definition of a new property) requires no changes in the representation of the asserted facts in the RDF database, because it does not impose a structural change in the physical representation. In contrast, in the relational DBMS, the physical representation of the data schema is dependent on the schema. The data are stored in files, structured in accordance with the structure of the tables, essentially as sequences of bytes of equal length, each of which representing one row in the table as depicted on *Fig. 7.2* (on the left). Thus, a change in the schema (e.g., addition of a new column in a table) requires considerable re-arrangement of the data files.

Historically, over the last couple of decades, there were multiple attempts to implement "semantic databases" based on data understanding and interpretation. Those were labeled and promoted in very different ways, depending on their origin and the IT trends

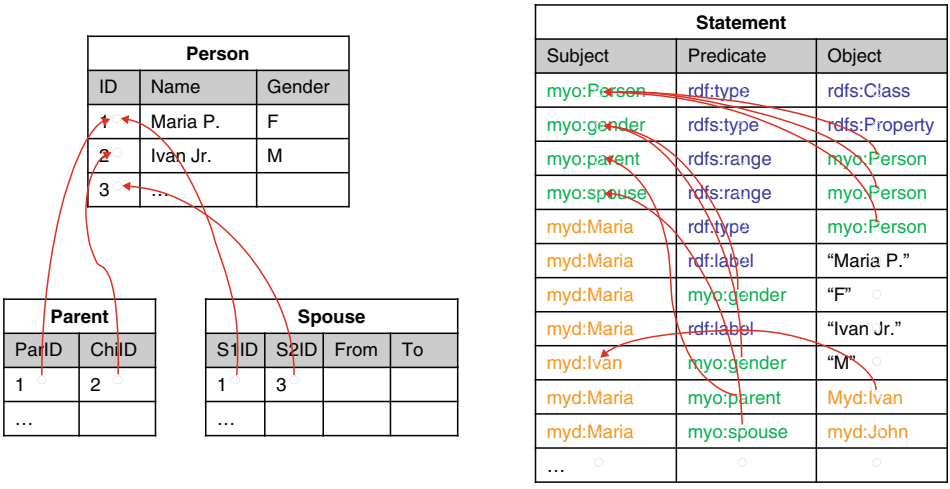

Fig. 7.2
Row stores versus RDF databases

at the time of their appearance. The most notable language for "semantic databases" in this line is probably known as Datalog – a limited version of Prolog, defined for usage in DBMS (such databases were referred to as *deductive databases*). Most of the scalable semantic repositories today support logical languages quite similar in spirit and expressive power to Datalog (see Sect. 7.2.2). Further, many of the expert systems and knowledge base management systems developed in the late 1980s and the early 1990s, as part of the Artificial Intelligence (AI) work, were also offering fairly similar functionality, packaged and promoted with plenty of high-level claims. Finally, over the last couple of decades, there have always been *reasoners* or *inference engines*. While these tools are mostly focused on logical inference, at the end of the day, their basic functionality is the same: One should be able to assert some facts and get answers based on interpretation of their semantics.

Over the last decade, the popularity of *column-stores* has been growing. The central idea is that, while in the relational database the information is stored and managed primarily by rows (those could be referred to as row-stores), in the column-stores, information is managed by columns. Typical representatives are Google's BigTable, [12], and the Vertica Database, [66]. The major advantage of this representation is the same as with the RDF databases: Changes in the schema are far easier to implement compared to RDBMS. Such representations are also far more efficient for the management of sparse data. Imagine a class of objects that can have 50 different attributes. In relational databases, the information about the instances of such class will be naturally modeled as a table of 51 columns (one for primary key and 50 for the attributes). Now, imagine that for each instance of the class there are values defined only for 10 of these attributes on average. In a typical RDBMS, this would result in a data file, which is 80% full with null values. In contrast, an RDF-based DBMS does not need to allocate space for missing attribute values – there will be simply no such records in the Statements table as shown in the example on *Fig. 7.2*.

In a nutshell, column-stores have considerable advantages against the RDBMS in two respects: dynamic data schema and sparse data. The principal advantage of the RDBMS is that the information is already grouped to a major extent, so, typically, the RDBMS needs to make less joins (i.e., to resolve less references to other records) during query evaluation. Obviously, both column- and row-stores have their advantages in different data management scenarios. Column-stores are good in situations where aggregates are computed over large number of similar data items, like in data warehouses. Row-stores are good in situations where relatively stable processes with predetermined structure are to be automated and managed.

While RDF databases and column-stores share a lot of design principles, a typical column-store differs from an RDF-based semantic repository in several ways:

- *Globally unique identifiers.* An important feature of RDF, as a data representation model, is that it is based on the notion of Unique Resource Identifiers (URI, [4]). All predicates and most of the subjects (see Sect. 7.1.3) are identified by a URI. In the examples provided above (see *Fig. 7.2*), the nodes are identified by URLs

(`myd:Maria` is a URL representation as QName, using namespace prefix, e.g., `myd:`). In contrast, most of the other DBMS use integer identifiers, which are unique only in the scope of the same type of elements in the same instance of the database.
- *Standard compliance*. While there are no well-established standards in the area of the column-stores, the RDF-based semantic repositories are highly interoperable between one another on the basis of a whole ecosystem of languages for schema definition, ontology definition, and querying.

Semantic repositories can be described as "*RDF-based column-stores with inference capabilities*."

As a wrap-up, provided is a list of the major characteristics and advantages of semantic repositories:

- *Easy integration of multiple data sources*: Once the schemas of these sources are semantically aligned, the inference capabilities of the engine support the interlinking and combination of the facts from the different sources.
- *Easy querying against rich, diverse, or unknown data schemas*: Inference is applied to match the semantics of the query to the semantics of the data, regardless of the vocabulary and the data modeling patterns used for encoding data.
- *Great analytical power*: One can count that semantics will be thoroughly applied even when this requires recursive inferences on multiple steps. In this way, semantic repositories can uncover facts, based on the interlinking of long chains of evidence – the vast majority of those facts would remain unspotted in a regular DBMS.
- *Efficient data interoperability*: Importing RDF data from one store to another is straightforward, based on the usage of globally unique identifiers.

The above qualities make semantic repositories an attractive choice for a range of data management tasks, that is, data integration, data warehousing, content management, metadata management, master data management (MDM), online analytical processing (OLAP), and business intelligence (BI). Over the last couple of years, these applications have been recognized and analyzed by many reputable analysts of information technology (see [52] and [67]).

7.1.3 RDF, SPARQL, and RDF-Based Data Representation Models

Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web [41]. Although it was designed to represent metadata about Web resources, RDF has much broader use as a generic data model for structured data management and reasoning. SPARQL, [53], is a query language for RDF data sources. Here we provide an overview of several augmentations of the basic RDF specification,

which are relevant to its usage as data representation model in semantic repositories and the support of SPARQL queries.

The atomic data element in RDF represents a statement about a resource or a blank node. Each statement is a triple of the format < Subject, Predicate, Object>, for example, <John, loves, Mary > or < Mary, hasBirthday, "14.11.1972">. RDF description can be seen as directed labeled graph, where each triple defines an edge directed from the subject to the object, which is labeled with the predicate. The nodes of the graph could be URI (unified resource identifiers, [4], e.g., an URL), blank node (auxiliary nodes), or XML literal. The predicates are always URIs. Literals are not allowed in subject position, that is, they cannot be the start of an edge in the graph. Intuitively, literals are used to describe resources identified by URIs, but there is no point in describing literals, because they represent primitive data values. A sample graph, which describes a Web page, created by a person called Adam, can be seen in *Fig. 7.3*. More on RDF can be found in Semantic Annotation and Retrieval: RDF.

SPARQL [53] is an SQL-like query language for RDF data, specified by the RDF Data Access Working Group of W3C. It differs from SQL in the following aspects:

- SPARQL does not contain specific Data Definition Language (DDL) provisions because the schemas are represented in both RDFS and OWL as standard RDF graphs, thus requiring no specific language to deal with them.
- SPARQL is not a Data Modification Language (DML), that is, one cannot insert, delete, and update RDF graphs using SPARQL. The major reason for this is that there is still no consensus on the optimal DML design for RDF.

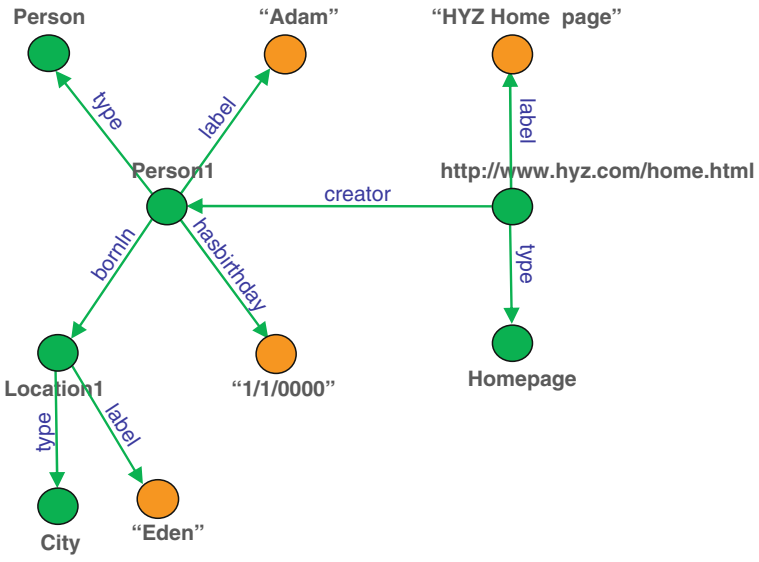

Fig. 7.3
RDF graph describing Adam and his home page

SPARQL supports four types of queries:

- SELECT queries – return *n*-tuples of results just like the SELECT queries in SQL.
- DESCRIBE queries – return an RDF graph. The resulting graph describes the resources, which match the query constraints. Usually, a description of a resource is considered an RDF-molecule, forming the immediate neighborhood of an URI.
- ASK queries – provide positive or negative answer indicating whether or not the query pattern can be satisfied.
- CONSTRUCT queries – return an RDF graph constructed by means of the substitution of the variables in the graph template and combining the triples into a single RDF graph by set union. More on SPARQL can be found in Querying the Semantic Web: SPARQL.

Named graph, [11], is an RDF graph with assigned name in the form of a URI reference. In an extended RDF model, one can deal with multiple named graphs and describe the graphs, making statements about them, putting their URIs in subject position. While the original definition of named graphs leaves plenty of room for interpretation, a more concrete definition is provided in the specification of SPARQL, where queries are evaluated against datasets, composed from multiple RDF graphs. In SPARQL, [53], *RDF Dataset* is defined as

```
{ G, (<U1>, G1), (<U2>, G2), ... (<Un>, Gn) }
```

where `G` and each `Gi` are RDF graphs, and each `<Ui>` is a distinct IRI (internationalized URI). The pairs `(<Ui>, Gi)` are called *named graphs*, where `<Ui>` is the name of graph `Gi`. `G` is called *default graph* – it contains all triples, which belong to the dataset, but not to any specific named graph. The notion of default graph is not present in [41]. Intuitively, a *dataset* integrates several RDF graphs in such a way that each graph can be distinguished, manipulated, and addressed separately. In a nutshell, in the RDF data model, extended with named graphs, statements can be part of named graphs, which can be used to model provenance or other contextual meta-information: Named graphs are also referred to as *contexts* in some systems.

Formally, a dataset can be represented as an RDF multi-graph, which, in its turn, can be represented as a set of quadruples of the following type: `<S,P,O,G>`. The first three elements of the quadruple, `<S,P,O>`, represent an RDF statement; the fourth element, `G`, represents the name of the named graph.

The SPARQL specification does not provide sufficient formal grounds for the semantics of the named graphs in order to determine the possible behavior of a semantic repository that supports such an extended RDF model. As SPARQL supports no data modification, it is unclear what should be the formal consequences of adding or removing a statement from a named graph. Counting statements in a SPARQL dataset is also not specified. To fill this gap, the specification of the second generation of the ORDI framework, [43] defined these aspects of the semantics of named graphs and introduced a new notion, namely triplesets.

A *tripleset*, as introduced in [43], is a mechanism to deal with parts of datasets or to group some of the statements in a dataset. An RDF dataset with named graphs and

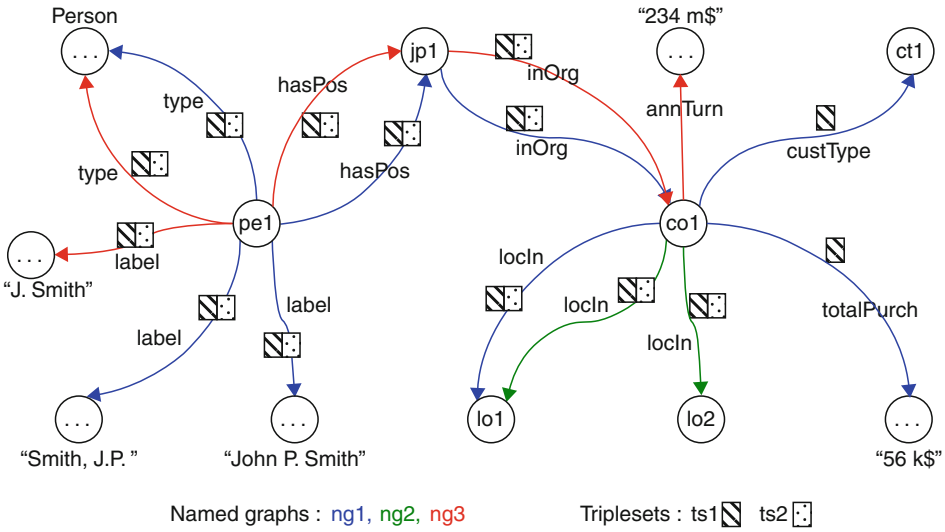

Fig. 7.4
RDF graph with named graphs and triplesets

triplesets is depicted in *Fig. 7.4*. The difference between named graphs and datasets can be explained as follows:

- Named graphs "own" the statements; for example, each statement belongs to a specific named graph of the default graph. When a statement is added or deleted from a named graph, a new arc appears or disappears from the multi-graph, which represents the datasets; the count of the arcs increases or decreases, respectively.
- Triplesets are tags on the statements. When a statement is associated with a tripleset, this can be seen as an association operation, which does not add a new arc in the graph. When statement is removed from a tripleset, that is, it is no longer a member of this group of statements, it does not disappear from the dataset, it is just being un-tagged or disassociated.

  Given the above extensions, the atomic entity of the tripleset model is a quintuple:

```
< S, P, O, G, {TS1,...,TSn} >
```

where `G` is the named graph and `{TS1, ... TSn}` is a set of identifiers of the triplesets to which the contextualized statement `< S, P, O, G >` is associated. In other words, each statement (from each graph) can be member of multiple triplesets. Formally, the extension of a tripleset is an RDF multi-graph, a subset of the set of all quadruples in the dataset. Triplesets are named, that is, each tripleset is associated with an URI. It is worth noting that the above quintuple is provided for the sake of formal specification of the semantics of the extended RDF data model. Semantic repositories can (and most of them do) implement alternative data representation and indexing structures, while supporting the same semantics.

The need for the enhancement of the RDF data model with triplesets is a result from the clear specification of the semantics of the named graphs. Named graphs are used most often for tracking of provenance, for example, when multiple graphs from different sources are merged or referenced (e.g., when dealing with linked data, see Sects. 7.1.4 and 7.8). In such a scenario, strong "ownership" semantics should be enforced for the named graphs so that updates of the contents of specific named graphs can have real impact on the contents of the dataset. Once named graphs are given such semantics, there is a need for a mechanism which allows for dealing with metadata about the contents of an integrated dataset. Triplesets are defined as a weaker mechanism to group quadruples (statements form a dataset) and assign metadata with them. Moreover, since the triplesets allow for the designation or tagging of parts of a dataset, they are especially useful when selecting parts of the dataset, for example, in the course of multistage processing, where intermediate results should be passed from one component to another. Triplesets are supported by BigOWLIM and the storage infrastructure of Freebase (see Sect. 7.8.1.1); they are also a standard feature of the LarKC data layer (see Sect. 7.5.1 in [31]). A more concise description of the tripleset mechanism is provided in [35].

7.1.4 Linked Data

The notion of "linked data" (linked data principles and applications are presented in greater detail in Semantic Annotation and Retrieval: Web of Data; this section provides only a brief introduction to their major principles as linked data bring specific requirements for semantic repositories) is defined by Tim Berners-Lee, [4], as RDF graphs, published on the WWW so that one can explore them across servers by following the links in the graph in a manner similar to the way the HTML Web is navigated. It is viewed as a method for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF. "Linked data" are constituted by publishing and interlinking open data sources, following four principles. These are:

1. Using URIs as names for things
2. Using HTTP URIs, so that people can look up those names
3. Providing useful information when someone looks up a URI
4. Including links to other URI, so people can discover more things

In fact, most of the RDF datasets fulfill principles 1, 2, and 4 by design. The piece of novelty in the design principles above concerns the requirement for enabling Semantic Web browsers to load HTTP descriptions of RDF resources based on their URIs. To this end, data publishers should make sure that:

- The "physical" addresses of the published pieces of data are the same as the "logical" addresses, used as RDF identifiers (URIs).
- Upon receiving an HTTP request, the server should return an RDF-molecule, that is, the set of triples that describe the resource.

Linking Open Data (LOD, [68]) is a W3C SWEO community project aiming to extend the Web by publishing open datasets as RDF and by creating RDF links between data items from different data sources. Linked Open Data provides sets of referencable, semantically interlinked resources with defined meaning. The central dataset of the LOD is DBPedia – an RDF extract of the Wikipedia open encyclopedia (DBPedia is discussed in Sect. 7.5.3). Because of the many mappings between other LOD datasets and DBPedia, the latter serves as a sort of a hub in the LOD graph assuring a certain level of connectivity. LOD is rapidly growing – as of September 2010, it contains 203 datasets, with total volume of 25 billion statements, interlinked with 142 million statements as illustrated on *Fig. 7.5*.

Although not related to semantics, the linked data concept turns into an enabling factor for the realization of the Semantic Web as a global Web of structured data around the Linking Open Data initiative. Still, querying and reasoning with linked data raises various challenges related to the very scale and nature of such data. A specific approach for the management of linked data, named "reason-able views," is presented in Sect. 7.8.1 and Sect. 7.8.1.1 provides an overview of few of the central LOD datasets.

More generally, the management and publishing of linked data creates major usage scenarios for semantic repositories, which bring a range of specific requirements, such as:

- Dealing with a massive number of different predicates with no proper definition – there are about hundred thousand predicates in DBPedia.
- Optimizations in the reasoning with owl:sameAs-equivalence (see Sect. 7.2.1.4) – in linked data, many objects have multiple different identifiers across different datasets or even within a single dataset.
- Novel query methods need to be developed as the standard structured query languages and engines assume schema knowledge at the time of query specification. In the linked data scenario, such assumptions are nonrealistic; methods of the type of the RDF Search developed in BigOWLIM and used in FactForge are required (see Sects. 7.6.4 and 7.8.1.1 respectively).

7.2 Reasoning in the Semantic Repositories

A major distinctive feature of the semantic repositories, versus most of the other database management systems (DBMS), is that they manage the data taking into consideration their semantics. They can interpret the semantics of the schemas and the data loaded into them and deliver answers, based on this interpretation. In the simplest scenario, a request using the pattern `"?x fellowCitizenOf Frank"` will return Orri as a result, if `"Frank fellowCitizenOf Orri"` was asserted and `fellowCitizenOf` is defined to be a symmetric relationship.

The ability of the semantic repositories to interpret the semantics of the data delivers one major advantage. The syntax of the query is no longer required to match the syntax of

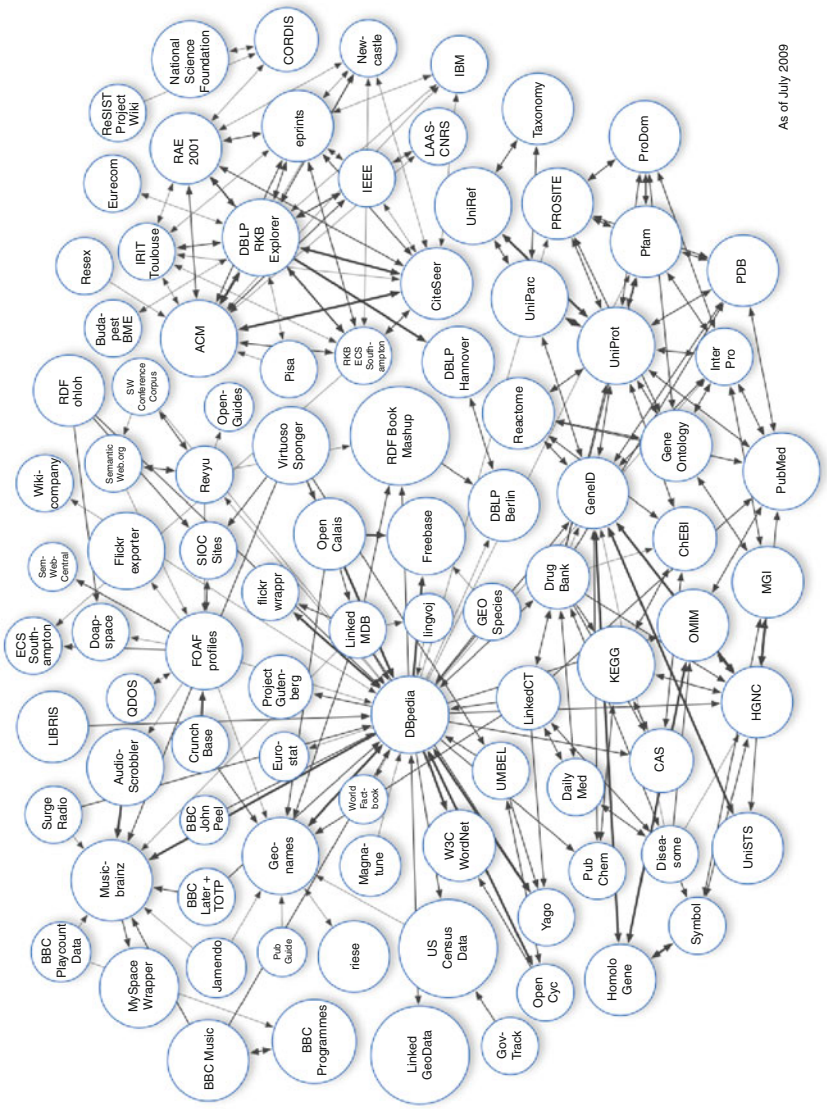

Fig. 7.5
Map of the datasets in Linking Open Data (LOD) project, [68]

the assertion. In the example above, the user will receive one and the same results, disregarding the direction in which the symmetric relationship was asserted. A slightly more complicated scenario is presented in *Fig. 7.1*, where the request is formulated through a more general relationship (relative), but the repository is capable of delivering results based on assertion of a more specific one (child). More generally, the engine takes care to interpret the semantics of the query and the semantics of the data in order to deliver all correct results.

The ability to abstract the query syntax from the data syntax bears important advantages in data access scenarios where one has to deal with complex relationships or with schema diversity. *Figure 7.1* presented a simple example of enhanced analytical power – one can obtain results at the desired level of generality, even if the underlying data are far more specific. As long as the semantic repositories can interpret the semantics in a recursive fashion, one can enjoy interpretations of the data, which combine results from previous interpretations and explicit assertions. In other words, depending on the data patterns and the semantics, one can retrieve facts, which are the results of multiple steps of interpretation, and by this way one can uncover relationships, which would otherwise remain hidden.

In a data integration scenario, often similar facts are encoded in different ways across different data sources. Imagine a case when one data source deals with information about the locations of airports encoded as `"Stansted airportOf London"`, while another data source associates airports to cities in a more fine grained manner, using patterns like `"London hasMainAirport Heathrow"` and `"London hasAlternativeAirport Gatwick."` An approach to integrate these data using RDFS and OWL would be to define a new property `hasAirport`, and to define it to be inverse property of `airportOf` and super-property of both `hasMainAirport` and `hasAlternativeAirport`. Given such semantic vocabulary alignment, a semantic repository will be able to return all the three airports as a match for the retrieval pattern `"London hasAirport ?x."` This way data and schema interpretation facilitate data integration.

Although all the examples of data or query interpretation can be achieved when working with other types of DBMS also, RDF-based semantic repositories provide the most efficient and standard compliant mechanism to deliver such limited intelligence in a robust, reliable, and manageable manner.

7.2.1 Lightweight Inference Integration

Semantic repositories are expected to demonstrate performance and scalability comparable to those of the mature database management systems. They should be able to deal with billions of facts, to handle online updates while at the same time processing vast query loads. This puts heavy constraints on both the worst-case and average-case complexity of the reasoning algorithms to be used. Further, such repositories shall be used and operated by engineers and administrators with no deep understanding of reasoning or

mathematical logic. This means that the inference techniques used should allow inferences to be traced and "debugged" by a typical database administrator, given one week of RDF (S) and OWL training. Altogether, these factors limit the applicability of non-tractable languages and techniques like satisfiability checking in semantic repositories. As seen in Sect. 7.6, most of the semantic repositories (which offer inference at all) support reasoning based on Datalog-style rule entailment.

7.2.1.1 Rule-Based Entailment Example and Relations to Query Evaluation

The following example is meant to facilitate intuitive understanding of the mechanisms for Datalog-style rule entailment in RDF repositories (a proper definition of RDF-based rule-entailment formalisms is provided in Sect. 7.1.1) and their similarities to the structured query evaluation mechanisms. In a typical language of this type, rules are defined through premises and consequences. One or more RDF triple patterns, involving variables, constitute the premises of the rule and several others represent the consequences. For variable bindings, which satisfy the premises, the reasoner can infer the consequences. For instance, given the rule:

```
<I, rdf:type, C1 > AND < C1, rdfs:subClassOf, C2> => < I, rdf:type, C2>
```

where `I`, `C1`, and `C2` are variables, if the following statements are already in the repository

```
<myData:Maria, rdf:type, ptop:Woman>
<ptop:Woman, rdfs:subClassOf, ptop:Person>
```

it can infer the following statement

```
<myData:Maria, rdf:type, ptop:Person>
```

It is important to be considered that rule-based reasoning of the type presented above is computationally very similar to the evaluation of structured queries in languages like SQL and SPARQL (see Sect. 7.1.3). For instance, the consequences of the above rule are the same as the results of the following query (The definitions of the namespace prefixes `rdf` and `rdfs` are omitted above for the sake of better readability.):

```
CONSTRUCT { ?I rdf:type ?C2 }
WHERE { ?I rdf:type ?C1 . ?C1 rdfs:subClassOf ?C2 }
```

The latter has several implications. As a start, it indicates that reasoning can be implemented by means of query evaluation. It is also the case that most of the performance considerations concerning query evaluation are also relevant to rule-based reasoning – such is the case with distribution approaches discussed in Sect. 7.4.2.

7.2.1.2 Reasoning Strategies: Forward- and Backward-Chaining

The main strategies for the implementation of rule-based inference are as follows. (More information on rule-based semantics can be found in KR and Reasoning on the Semantic Web: RIF, which introduces the Rule Interchange Format and provides discussion on interoperability across multiple rule-based systems.)

- Forward-chaining: to start from the known facts (the explicit statements) and to derive valid implications. The goals of such reasoning can vary: to compute the *inferred closure*; to answer a particular query; to infer a particular sort of knowledge (e.g., the class taxonomy).
- Backward-chaining: to start from a particular fact or a query and to verify it or to find all possible solutions. In a nutshell, the reasoner decomposes or transforms the query into simpler (or alternative) requests that can be matched directly to explicit facts available in the KB or can be proven through further recursive transformations. The most popular example for backward-chaining is the unification mechanism in Prolog.

In the context of forward-chaining, *inferred closure* (also known as *deductive closure*) is the extension of a knowledge base (or an RDF graph) with all the implicit facts (or statements) that could be inferred from it, based on the enforced semantics.

In relational database management systems, *materialized views* represent an approach where the results of the query, which define the view, are cached in a sort of temporary table. Generally, in inference engines, *materialization* is the process of making some reasoning results explicit. In both cases, it is a matter of storing the results of data processing so that these are available at a later stage. In relation to semantic repositories, materialization is used to refer to a range of techniques similar to those referred above, which are, however, diverse and not standardized. Below is provided a definition for materialization, as a reasoning strategy for semantic repositories, considering that such a strategy should be transparent for the clients of the repository. In other words, following the "separation of concerns" principle, the clients of the repository should not experience functional differences in the case of a change of the strategy. Given the same sequence of update transactions and queries, the repository should return the same results for the queries disregarding the reasoning strategy, which is implemented. Still the clients can (and are likely to) experience change in some nonfunctional parameters, for example, the speed of specific operations.

Materialization is a strategy for reasoning in semantic repositories where the inferred closure of the contents of the repository is derived through forward-chaining and maintained in a form that allows its efficient usage. In practice, this means that the inferred closure of large volumes of data should be persisted and indexed in order to enable efficient query evaluation or retrieval. The repository should take care to keep the inferred closure up-to-date after each update of its contents, that is, at the end of each update transaction.

7.2.1.3 The Advantages and Applicability of the Different Strategies

The principal advantages and disadvantages of materialization can be summarized as follows:

- The loading of new facts gets slower because the repository is extending the inferred closure after each transaction. In fact, all the reasoning is performed during loading.
- The deletion of facts is also slow, because the repository should remove from the inferred closure all the facts, which cannot be inferred any longer.
- The maintenance of the inferred closure usually requires considerable additional space (RAM, disk, or both, depending on the implementation).
- Query and retrieval are fast, because no deduction, satisfiability checking, or other sorts of reasoning are required.

The advantages and disadvantages of the backward-chaining-based interpretation of the data in semantic repositories are also well known:

- The loading and modification of the data are faster, compared to repositories using materialization, because no time and space is lost for the computation and maintenance of the inferred closure of the data.
- Query evaluation is slower because extensive query rewriting (reformulation and expansion) has to be performed. As a result, a potentially much larger number of lookups in the indices are required, as compared to standard query evaluation.

The choice of the most appropriate and efficient reasoning strategy requires balancing between loading and query performance, considering several factors, related to the data, its semantics, and the typical usage scenarios and loads:

- When the data are updated very intensively, there will be relatively high costs for the maintenance of the inferred closure, so materialization becomes less attractive.
- In the case of challenging query loads, backward-chaining becomes inappropriate, as long as it increases the time for query evaluation considerably.
- If materialization requires time and space, which are hard to be secured, it should be avoided.
- Whenever low response time has to be guaranteed, backward-chaining should be avoided because it leads to recursive query evaluation, where performance cannot be easily estimated and managed.

The selection of the appropriate reasoning strategy for the specific application is very important as long as it can change the performance of some operations several times; see for instance, the variation in the loading performance in dependence of the materialization complexity in Sect. 7.7.2. One should note, however, that the reasoning strategy generally impacts the so-called average-case complexity. The worst-case reasoning complexity, even for relatively simple ontology languages, is unbearable for semantic

repositories, independently of the selected strategy. Suppose, for instance, a dataset where each of the one million citizens of Amsterdam is linked to one of the other citizens with a `fellowCitizen` relationship. Suppose also that the latter is defined to be transitive and symmetric, as it naturally is. The inferred closure of such dataset will contain one trillion facts; thus, total materialization is impractical. A query checking the pattern `"AmsCitizen fellowCitizen ?x,"` in a system using backward-chaining, will have to make one million steps of recursion, which is also practically unfeasible. Avoiding such cases requires through ontology and schema design and, more generally, concise data modeling. Architects and database administrators should be aware of such dangers and take care to control the complexity. In this respect, semantic repositories are not very different from the mainstream DBMS, which also expose unbearable query evaluation performance in cases of poor data or query modeling – no one can guarantee the good performance of an SQL database if indices are not properly defined or if a query specifies unconstrained Cartesian products on large tables.

Total materialization is adopted as a reasoning strategy in a number of popular Semantic Web repositories, including Sesame, Jena, DAML DB, ORACLE, and BigOWLIM (please, refer to the corresponding subsections of Sect. 7.6 for details). Probably, the most important advantage of the usage of materialization in semantic repositories is that all data are available at query time, which makes possible RDBMS-like query optimization techniques. The query evaluation engine can use statistics to make guesses about the cost of evaluation of a particular constraint and the cardinality of its results; such guesses allow DBMS to reorder constraints in order to build an efficient query evaluation plan. Such optimization techniques are far more complex in the case of deductive query evaluation.

7.2.1.4 Hybrid Strategies and Dynamic Materialization

A range of different techniques have been invented to improve the efficiency of the reasoning strategies for specific types of data and usage scenarios.

There are cases where hybrid strategies, which involve *partial materialization* and partial backward-chaining for specific data patterns, deliver the best overall performance. One such schema, related to the efficient handling of the semantics of `owl:sameAs`, is described in Sect. 7.4.1. The lightweight version of OWLIM (SwiftOWLIM ver. 2.9, see Sect. 7.6.4) implements a partial materialization schema where the inference related to transitive, inverse, and symmetric properties is performed during forward-chaining; however, a specific materialization strategy is applied to avoid the inflation of the indices of the repository.

There are repositories, which use backward-chaining, but do caching of the intermediate results, which effectively means that they perform partial materialization on demand. Such a strategy, named *dynamic materialization*, is implemented in AllegroGraph (see Sect. 7.6.2). The advantage, compared to the full materialization, is that the inferred closure is materialized incrementally. Further, in many application

scenarios, a significant fraction of the implicitly inferable facts will never be accessed during the handling of retrieval queries – in the case of dynamic materialization, those will never be materialized. The advantage compared to backward-chaining is that the same conclusions do not need to be inferred multiple times. The principal disadvantages of the dynamic materialization are that (1) it still slows down the query processing, (2) query optimization, based on cardinality statistics, is still hardly applicable, and (3) the cached partial materialization still requires maintenance upon modification of the repository contents.

7.2.1.5 The Honey and the Sting of owl:sameAs

`owl:sameAs` is a system predicate in OWL, declaring that two different URIs denote one and the same resource. Most often it is used to align the different identifiers of the same real-world entity used in different data sources. In FactForge (see Sect. 7.8.1.1), one and the same entity has different URIs in the different linked data LOD datasets (See Sect. 7.1.4) where it appears. For instance, in DBPedia, the URI of the city of Vienna is http://dbpedia.org/page/Vienna, while in Geonames, it is http://sws.geonames.org/2761369/. DBpedia contains the statement

```
(S1)  dbpedia:Vienna owl:sameAs geonames:2761369
```

which declares that the two URIs are equivalent. `owl:sameAs` is probably the most important OWL predicate when it comes to merging data from different data sources.

Following the specification of OWL (To be more specific, this is the semantics defined in OWL 2 RL, but also in the other dialects discussed in Sect. 7.1.1.), whenever two URIs `U1` and `U2` are declared equivalent, all statements that involve `U1` and are true will also be inferable and retrievable, with `U2`, replacing `U1` at the same position. For instance, in Geonames, the city of Vienna is defined as part of http://www.geonames.org/2761367/ (the first-order administrative division in Austria with the same name), which, in turn, is part of Austria (http://www.geonames.org/2782113):

```
(S2)  geonames:2761369 gno:parentFeature geonames:2761367
(S3)  geonames:2761367 gno:parentFeature geonames:2782113
```

As long as `gno:parentFeature` is a transitive relationship, in the course of the initial forward-chaining, it will be derived that the city of Vienna is also part of Austria:

```
(S4)  geonames:2761369 gno:parentFeature geonames:2782113
```

Based on the semantics of `owl:sameAs`, from (S1), it should be inferred that statements (S2) and (S4) also hold for Vienna when it is referred with its DBpedia URI:

```
(S5)  dbpedia:Vienna gno:parentFeature geonames:2761367
(S6)  dbpedia:Vienna gno:parentFeature geonames:2782113
```

These are true statements and when querying RDF data, no matter which one of the equivalent URIs is used in the explicit statements, the same results will be returned. When one considers that Austria, too, has an equivalent URI in DBpedia.

```
(S7)  geonames:2782113 owl:sameAs dbpedia:Austria
```

it should also infer that:

```
(S8)  dbpedia:Vienna gno:parentFeature dbpedia:Austria
(S9)  geonames:2761369 gno:parentFeature dbpedia:Austria
(S10)  geonames:2761367 gno:parentFeature dbpedia:Austria
```

In the above example, there are two alignment statements (S1 and S7), two statements carrying specific factual knowledge (S2 and S3), one statement inferred due to a transitive property (S4), and seven statements inferred as a result of `owl:sameAs` alignment (S5, S7, S8, S9, S10, and the inverse statements of S1 and S7, which are not given above due to space limitations). As seen, inference without `owl:sameAs` inflated the dataset by 25% (one new statement on top of four explicit), while `owl:sameAs` related inference increased the dataset by 175% (seven new statements). Considering that Vienna has a URI also in UMBEL, which is also declared equivalent to the one in DBpedia, the addition of one more explicit statement for this alignment will cause inference of four new implicit statements (duplicates of S1, S5, S6, and S8). Although this is a small example, it provides a good indication about the performance implications of using `owl:sameAs` alignment in LOD. Also, because `owl:sameAs` is a transitive, reflexive, and symmetric relationship, a set of N equivalent URIs N^2 `owl:sameAs` statements will be generated for each pair of URIs. Thus, although `owl:sameAs` is useful for interlinking RDF datasets, its semantics cause considerable inflation of the number of implicit facts that should be considered during inference and query evaluation (either through forward- or backward-chaining).

To overcome this problem, BigOWLIM and ORACLE (see Sects. 7.6.4 and 7.6.7) handle `owl:sameAs` in a specific manner. In their indices, each set of equivalent URIs (the equivalence class with respect to `owl:sameAs`) is represented by a single super-node. This way, the repository does not inflate the indices and, at the same time, retains the ability to enumerate all statements that should be inferred using the equivalence upon retrieval request (e.g., during inference or query evaluation).

7.2.1.6 Truth Maintenance and Smooth Invalidation

Normally, repositories, which use materialization support dialects of OWL, which allow for monotonic inference (see Sect. 7.2.2). Upon the addition of new statements, they can incrementally extend the inferred closure with new facts, which can be entailed from the new data. The deletion of statements requires the repository to remove from the inferred closure statements, which are no longer inferable. The classical approach to

implement this is known from the expert systems as a *truth maintenance system* (TMS), which keeps meta-information about which fact can be inferred from which other facts. While such systems allow for the tracing of the inference dependencies and the invalidation of the statements which are no longer supported, the overheads associated with the maintenance of the TMS information itself appear quite high for most of the scenarios of usage of semantic repositories. This is the reason why most of the repositories which use materialization do not implement a TMS – upon deletion, they have to delete the entire inferred closure and compute it again. Scenarios where deletion can be avoided or postponed have no problem with such strategy. However, there are plenty of scenarios where deletion should be performed in real time.

Smooth invalidation is technique implemented in BigOWLIM (see Sect. 7.6.4), which allows for the efficient maintenance of the inferred closure upon deletion. It is based on an algorithm which performs a sequence of backward- and forward-chaining iterations to figure out what part of the deductive closure is no longer supported, without using truth maintenance information. The complexity of this algorithm is comparable to the complexity of the plain forward-chaining. In other words, the update of the inferred closure after removal of some statements requires time and computational resource comparable to those needed for the update of the inferred closure after the addition of the same statements.

One should note that the removal of key statements from the schema may lead to the invalidation of a large fraction of the inferred closure – in such cases the computation of the inferred closure from scratch can be the more efficient option. Still, in most of the data management scenarios, non-monotonic schema changes represent a tiny fraction of all update transactions and usually can be postponed and implemented in a time slot when full re-inference is feasible. This way, for a very large range of applications, forward-chaining, combined with optimizations (like those discussed in Sects. 7.2.1.3 and 7.2.1.4) and smooth invalidation, represents the optimal reasoning strategy, which delivers good performance through the entire life cycle of the data. A BigOWLIM instance implementing this strategy was used behind the BBC's World Cup 2010 website, presented in Sect. 7.8.2.

7.2.2 OWL Dialects Suitable for Scalable Inference

In order to match the expectations for the next-generation global Web of data, the Semantic Web requires a scalable high-performance storage and reasoning infrastructure. One challenge toward building such an infrastructure is the expressivity of its schema and ontology definition standards RDFS and OWL. RDFS, [10], is the schema language for RDF, which allows for the definitions of subsumption hierarchies of classes and properties; the latter being binary relationships defined with their domains and ranges. While RDFS is generally a fairly simple knowledge representation language, implementing semantic repositories which support its semantics and provide performance and scalability comparable to those of relational database management systems (RDBMS) is very challenging.

The semantics of RDFS is based on Logical Programming (LP) – a declarative programming paradigm, in which the program specifies a computation by giving the properties of a correct answer. The LP languages like PROLOG emphasize the logical properties of a computation, using logic and proof procedures to define and resolve problems. Most logic programming is based on Horn-clause logic with negation-as-failure to store the information and rule entailment to solve problems. Datalog is a query and rule language, a simplified version of PROLOG, meant to enable the efficient implementation of deductive databases. The semantics of RDFS is defined by means of rule-entailment formalism, which is a simplification of Datalog.

OWL [13] is an ontology language which supports more comprehensive logical descriptions of the schema elements, for instance: transitive, symmetric, and inverse properties; unions and intersections of classes; and property restrictions (for a detailed introduction into OWL one can refer to ❯ KR and Reasoning on the Semantic Web: OWL). The first version of the OWL specification, which was published as a W3C standard in year 2004 has three dialects: OWL Lite, OWL DL, and OWL Full. They range in their levels of expressivity. OWL Lite is a subset of OWL DL, and OWL DL is a subset of OWL Full. The OWL language is based on description logics (DL, [3]). The reasoning procedures of DLs are decision procedures that are aimed to always terminate – in mathematical logic terms this means that DLs are decidable. Compared to other logical languages, DLs are relatively inexpressive. Still reasoning with DLs is based on satisfiability checking, which means that in order to prove or to reject a specific statement, a DL reasoner needs to check whether it is possible or not to build a model of the world that satisfies a "theory" which includes this statement or its negation. For instance, suppose that there is a semantic repository which contains one billion statements and a client makes a query, checking whether a specific resource is an instance of a specific class. In order to validate this, with respect to the semantics of OWL DL, a repository should add to its current contents the statement that the resource is not an instance of the class and check whether the new state of the repository is consistent. It is clear that such semantics is impractical to implement for large volumes of data. Even the simplest dialect of OWL, OWL Lite is a DL formalism which does not support algorithms enabling efficient inference and query answering over reasonably large knowledge bases.

Logic programming and description logics support semantics and data interpretation capabilities of a different nature: LP uses rules to infer new knowledge, whereas DL employ descriptive classification mechanisms. None of these is more powerful or expressive than the other one – there are meaning aspects, which can be expressed in each one of them, which cannot be expressed in a language from the other paradigm. As a result, the semantics of OWL Lite and DL are incompatible with that of RDFS. (The issues related to the interoperability and layering of the Semantic Web languages are also discussed in ❯ Introduction to the Semantic Web Technologies.) Although OWL was meant to be layered on top of RDFS in the Semantic Web specification stack, there is no "backward compatibility." In practical terms, this means that it may be impossible to "upgrade" to OWL an application, which uses RDFS schemas, without replacing them with OWL

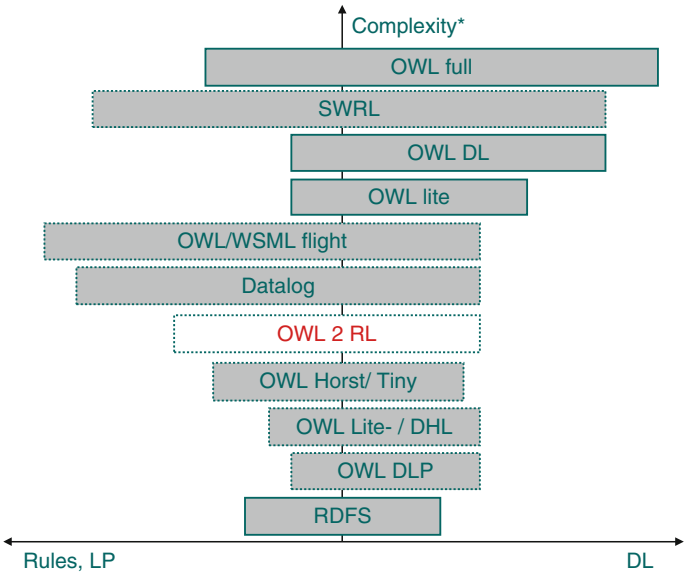

Fig. 7.6
Diagram of expressivity of OWL dialects

ontologies. The latter may require considerable changes in the semantics of the classes and the properties and in the data modeling principles used in the application.

To bridge the gap of expressivity, compatibility, and logical decidability and reach the goals of scalable inference, other dialects of OWL have been created which lie between RDF(S) and OWL Lite. *Figure 7.6* presents a simplified map of the expressivity or complexity of a number of these OWL-related languages together with their bias toward description logic (DL) and Logical Programming (LP) based semantics. The diagram provides a very rough idea about the expressivity of the languages, based on the complexity of entailment algorithms for them. A direct comparison between the different languages is impossible in many of the cases. For instance, Datalog is not simpler than OWL DL, it just allows for a different type of complexity.

OWL DLP is a nonstandard dialect, offering a promising compromise between expressive power, efficient reasoning, and compatibility. It is defined in [21] as the intersection of the expressivity of OWL DL and Logic Programming (LP). In fact, OWL DLP is defined as the most expressive sub-language of OWL DL, which can be mapped to Datalog. OWL DLP is simpler than OWL Lite. The alignment of its semantics to the one of RDFS is easier, as compared to the Lite and DL dialects. Still, this can only be achieved through the enforcement of some additional modeling constraints and transformations. A broad collection of information related to OWL DLP can be found in [49]. DLP has certain advantages:

- There is freedom to use either DL or LP (and associated tools and methodologies) for modeling purposes, depending on the modeler's experience and preferences.

- From an implementation perspective, either DL reasoners or deductive rule systems can be used. Thus it is possible to model using one paradigm, for example, a DL-biased ontology editor, and to use a reasoning engine based on the other paradigm, for example, a semantic repository based on rules.

These features of DLP provide extra flexibility and ensure interoperability with a variety of tools. Experience with using OWL has shown that existing ontologies frequently use only very few constructs outside the DLP language.

In [62] ter Horst defines RDFS extensions toward rule support and describes a fragment of OWL, more expressive than OWL DLP. He introduces the notion of R-entailment of one (target) RDF graph from another (source) RDF graph on the basis of a set of entailment rules R. R-entailment is more general than the D-entailment used by Hayes, [26], in defining the standard RDFS semantics. Each rule has a set of premises, which conjunctively define the body of the rule. The premises are "extended" RDF statements, where variables can take any of the three positions. The head of the rule comprises one or more consequences, each of which is, again, an extended RDF statement. The consequences may not contain free variables, that is, which are not used in the body of the rule. The consequences may contain blank nodes.

The extension of the R-entailment (as compared to the D-entailment) is that it "operates" on top of the so-called generalized RDF graphs, where blank nodes can appear as predicates. R-entailment rules without premises are used to declare axiomatic statements. Rules without consequences are used to imply inconsistency.

This extension of RDFS became popular as "OWL Horst." As outlined in [62], this language has a number of important characteristics:

- It is a proper (backward-compatible) extension of RDFS. In contrast to OWL DLP, it puts no constraints on the RDFS semantics. The widely discussed meta-classes (classes as instances of other classes) are not disallowed in OWL Horst.
- Unlike the DL-based rule languages, like SWRL, [28] and [46], R-entailment provides a formalism for rule extensions without DL-related constraints;
- Its complexity is lower than the one of SWRL and other approaches combining DL ontologies with rules; see Sect. 7.5 of [62].

OWL Horst is supported by OWLIM and ORACLE (presented in Sect. 7.6), which makes it the OWL dialect that has the largest industry support. An official OWL dialect with the same properties emerged recently under the name OWL 2 RL. The latter is one of the tractable profiles (dialects) defined in the specification of OWL 2, [45] – the next version of the OWL language that is currently in process of standardization. OWL 2 RL is designed with the objective to be the most expressive OWL dialect, which allows for efficient reasoning with large volumes of data in rule-based systems. OWL 2 RL was inspired by OWL Horst – its semantics is defined with the rule language equivalent to R-entailment. However, OWL 2 RL is considerably more expressive than OWL Horst. Support for OWL 2 RL is provided by several reasoning engines, including OWLIM and ORACLE. More details on the OWL 2 profiles can be found in KR and Reasoning on the Semantic Web: OWL.

Recent research reported in [58] evaluates the level of completeness of the inference supported by few inference engines (namely, HAWK) and semantic repositories: IBM's Minerva, Sesame (Sect. 7.6.8), and OWLIM (Sect. 7.6.4). It demonstrates that although OWLIM supports sufficient reasoning to answer the LUBM (LUBM benchmark is introduced in Sect. 7.5.1) queries correctly, it is still not complete with respect to the semantics of the data and the queries.

7.3 Semantic Repository Tasks, Performance Factors, and Dimensions

Measuring and benchmarking the performance of semantic repositories is an important aspect in allowing the engineers to understand and use them efficiently. As discussed in Sect. 7.1.2, semantic repositories are RDF databases that may or may not provide lightweight inference support. Their benchmarking is a complicated exercise, which requires a clear conceptualization and structuring. This section provides a conceptual framework for benchmarking of semantic repositories, as further development of [32]. The framework takes into consideration the tasks executed by the semantic repository, examines the performance factors and the performance dimensions of the measurements, and introduces the concept of full-cycle benchmarking.

A wide range of links to resources related to benchmarking RDF repositories can be found on the page on "RDF Store Benchmarking" in the ESW wiki, maintained by W3C, at http://esw.w3.org/topic/RdfStoreBenchmarking.

7.3.1 Tasks

The major tasks and activities toward which the performance of semantic repositories needs to be benchmarked are:

- Data loading, including parsing, persistence, and indexing of both instance data and ontologies
- Query evaluation, including query preparation, optimization, and fetching
- Data modification, which may involve changes to the ontologies and the schemas

Inference is not a first-level activity in a semantic repository. Depending on the implementation, it can affect the performance of the other activities, for instance, when inference is performed during loading.

Modifications to the data and/or schemas (e.g., updating and deleting values or changing class definitions) represent another important class of the tasks performed against a semantic repository. Most of the contemporary RDF-related benchmark suites do not cover modification tasks, because their specifics, complexity, and importance can change considerably across applications and usage patterns.

7.3.2 Performance Factors

The performance of data loading depends on several factors:

- *Materialization* – whether and to what extent forward-chaining is performed at load time, including the complexity of the forward-chaining (see Sect. 7.2.1)
- *Data model complexity* – support for extended RDF data models (see Sect. 7.1.3), for instance, including named graphs, is computationally more "expensive" as compared to the simple triple model
- *Indexing specifics* – repositories may or may not create a variety of different indices in dependence of the datasets loaded, the foreseen usage patterns, hardware constraints, etc.
- *Data access and location* – where the data is imported from, for instance, local files, loaded from the network, etc.

Several factors affect the time and memory space, or more generally, the computing resources, required for query evaluation:

- *Deduction* – whether and to what extent backward-chaining is involved, whether it is recursive, etc. (see Sect. 7.2.1)
- *Size of the result-set* – fetching large result-sets can take considerable time
- *Query complexity* – the number of constraints (e.g., triple-pattern joins), the semantics of the query (e.g., negation- and disjunction-related clauses), the usage of operators that are hard to support through indexing (e.g., LIKE)
- *Number of clients* – number of simultaneous client requests
- *Quality of results* – what is the quality of the results required in modalities where incomplete answers are requested

Transaction size and level of isolation may also have serious impact on the performance of both loading and query evaluation. Although many semantic repositories provide some sort of transaction isolation, it is usually less comprehensive than the corresponding mechanisms in the mature RDBMS. Furthermore, transaction management in large-scale systems is usually carefully designed and tuned for each specific setup.

7.3.3 Performance Dimensions

The performance dimensions of a semantic repository comprise parameters of a specific task or scenario, which affect its speed, and the size of the loaded datasets. There are several parameters affecting the speed of a semantic repository:

- *Scale* – the size of the repository in terms of number of RDF triples (more generally, facts or atomic assertions).

- *Schema and data complexity* – the complexity of the ontology/logical language, the specific ontology (or schema), and the dataset. A highly interconnected dataset, with long chains of transitive properties, can appear far more challenging for reasoning compared to another dataset, even when both are encoded against one and the same ontology; sparse versus dense datasets; presence and size of literals; number of predicates used; usage of `owl:sameAs`, and other alignment primitives.
- *Hardware and software setup* – the performance can vary considerably depending on the version and configuration of the compiler or the virtual machine, the operating system, the configuration of the engine itself, and the hardware configuration, of course.

The size of a dataset loaded in a semantic repository is measured in different ways. These are related to the types of statements taken into consideration. The following measures should be considered:

- *Number of inserted statements* (NIS): How many statements have been inserted in the repository.
- *Number of stored statements* (NSS): How many different statements have been stored and indexed by the repository. For engines using forward-chaining and materialization, the volume of the data to be indexed includes the inferred triples. For instance, the RDF/OWL representation of WordNet expands after materialization from 1.9 million (NIS) statements to 7.1 million (NSS). In the opposite direction, NSS can be smaller than NIS, when one and the same statement is inserted multiple times.
- *Number of retrievable statements* (NRS): How many different statements can be retrieved from the repository. This number can be different from NSS when the repository supports some sort of backward-chaining. For instance, BigOWLIM performs `owl:sameAs` optimization, which reduces considerably the NSS in the repository.

7.3.4 Full-Cycle Benchmarking

The performances of a specific setup of a given semantic repository on loading datasets and query evaluation are interdependent. More comprehensive indexing and forward-chaining take time during loading and respectively make the loading performance worse in order to facilitate faster query processing. In fact, if query evaluation performance is not to be considered, loading of RDF could be as fast as the file-system could be in storing the input files locally with no overheads to maintain indices.

Here the notion of *full-cycle benchmarking* is introduced – an evaluation experiment, which provides a complete picture of the performance of a repository with respect to the full "life cycle" of the data within the engine. At the high-level, this means the measuring and publication of data for both loading and query evaluation performance within a single experiment or benchmark run. Thus, full-cycle benchmarking requires loading

performance data to be matched with query evaluation data. A full-cycle run on LUBM, [22], or similar benchmark usually involves the following activities:

1. Loading input RDF files from the storage system
2. Parsing the RDF files
3. Indexing and storing the triples
4. Forward-chaining and materialization (optional)
5. Query parsing
6. Query optimization
   a. Query rewriting (optional)
7. Query evaluation, involving
   a. Backward-chaining (optional)
   b. Fetching of the results

While different repositories can employ different indexing, reasoning, and query evaluation strategies, the seven activities listed above should always be handled in a cycle that includes data loading and query evaluation.

To provide correct answers to queries, a semantic repository should consider the semantics of the data. For instance, it is impossible to deliver correct results to the queries of LUBM without some form of reasoning. As discussed in Sect. 7.2.1, different reasoning strategies can be adopted. Inference can take place either during loading (step 4 above) or during query evaluation (steps 6a and/or 7a above).

Full-cycle benchmarking provides an adequate picture on the performance of the engine and its utility in real-world applications. It shows the implications of the different design choices and implementation approaches. This notion needs further extension to include data modification.

7.4 Performance Considerations and Distribution Approaches

This section covers several issues related to the implementation of efficient and scalable semantic repositories. It includes a discussion on the principal advantages and disadvantages of some distribution approaches along with practical aspects such as the current state of the server hardware and the typical sizes of the datasets used today. The latter provides a good perspective toward the economic efficiency of the different distribution approaches.

7.4.1 Performance Engineering and Hardware Trends

As stated earlier, semantic repositories represent a sort of database management systems (DBMS) and have performance requirements similar to those of the relational DBMS and the NoSQL solutions, [47], used as back-ends for some of the largest websites. The basic

requirement is fast random access to relatively small blocks of data (few kilobytes), retrieved from large volumes of data – the indices maintained by the engine to enable the fast retrieval of data records based on specific constraints or retrieval patterns. Thus, the major factors for the performance of a server, when running a semantic repository, are the amount of the available RAM and the random seek speed of the hard drives. In an ideal situation, all the data can be cached in the RAM, which provides 3 orders of magnitude faster random access time than enterprise class hard drives (few microseconds versus few milliseconds).

A sample server configuration is provided below that, considering the above high-level requirements and their experience, is appropriate for handling serious query loads against datasets comparable to the volume of several of the central linked data datasets (see Sect. 7.1.4). While the needs of each usage scenario are different, this configuration could be used as a starting point for the estimation of the hardware requirements of a specific application.

7.4.1.1 Usage Scenario: 1 Million Queries per Day Against 10 Billion Statements

The scenario described here assumes that an enterprise is interested to provide, through its website, some sort of public information access service, which exposes proprietary content and data, which is annotated and interlinked with several linked data datasets. This could be the example of a media company, annotating its news with respect to a dataset like FactForge, described in Sect. 7.8.1.1, or a pharmaceutical company interlinking its internal data and clinical trial reports to resource like LinkedLifeData, Sect. 7.8.1.2. Such a scenario can be described as follows:

- Data size: between one and ten billion statements.
- Semantics: OWL 2 RL lightweight reasoning.
- Query loads: about one million queries per day; 10–20 queries per second. This load is comparable to the peak load at BBC's World Cup 2010 website (see Sect. 7.8.2).
- Update rate: about a hundred small updates per hour. Small updates can be described like transactions, which add or delete up to 1,000 statements, without changing the schemas and the ontologies. (Changes in the schemas are possible in such scenarios, and they can be adopted far quicker in semantic repositories as compared to relational databases. Still the complexity of such updates varies considerably and is hard for quantification.) Such updates are cases like updating the descriptions of several related entities or the metadata of several articles or pictures.

7.4.1.2 The $10,000 Database Server Landmark

Considering the scenario described in the previous section using the benchmarking data and experience, provided below is a sample description of a server configuration which,

given the current situation on the hardware market, combines good efficiency with low cost per transaction per second. The server configuration with an assembly cost below $10,000 is:

- 2 × Xeon 5500 series CPUs, each with four cores with hyper-threading
- 48–72 GB of RAM
- 8 × 120 GB SSD MLC drives with appropriate RAID controller, setup in RAID-0; those are to be used as fast storage for the indices of the semantic repository
- 2 × 2 TB SATA HDD set up in mirror (RAID-1), to be used for complementary data and content and for backup of the data placed on the SSD drives

The above configuration represents a commodity databases server, with the specificity that it is loaded with as much RAM as the configuration allows. This amount of memory would allow most of the engines to keep in memory several critical types of information (e.g., URL-to-internal-ID dictionaries) for datasets of a size up to 10–20 billions of statements. The solid state drives (SSD) and the hard disk drives (HDD) should be carefully selected to deliver best performance and reliability for this class of products.

More powerful database servers with four CPUs and 128 to 144 GB of RAM can be purchased for approximately $20,000–30,000. While the price of such servers is disproportionally higher, compared to their capacity, they are still very likely to offer a more efficient solution for data scalability issues, compared to distribution approaches involving data partitioning (see Sect. 7.4.2.1). Such machines can handle up to 40–50 billion explicit statements and query loads approaching one thousand queries per minute.

One should consider that the above configurations represent an efficient solution in the year 2010. As the hardware evolves rapidly, they should be considered only as a reference point to estimate the required hardware for specific loads.

7.4.2 Distributed Semantic Repositories

The restriction to a single computer system limits the overall scalability and performance parameters of a semantic repository. Database management distribution is usually considered to address one or more of the following goals:

1. To handle efficiently larger volumes of data (The size of the data, managed by a DBMS, and the average time for the execution of specific operations with the data are always in dependency – thus the first point above can be seen as a generalization of the following ones. Data scalability issues can always be avoided by adoption of a DBMS or a DBMS configuration, which has lower hardware requirements, trading scale for efficiency. Still, the notion of "data scalability" appears here, because often change of the DBMS configuration or further lowering the hardware requirements is impractical.)
2. To speed up the data loading and indexing and to improve the performance for updates

3. To lower the query evaluation time for complex queries (e.g., analytical Business Intelligence reports)
4. To better handle concurrent query loads and large numbers of users
5. To ensure failover, for example, to surmount failure of one or more nodes and repositories

The general approaches for the distribution of database management systems are:

- *Data partitioning*, where the information stored and accessed by the system is spread across multiple machines, so that none of them contains the entire dataset
- *Data replication*, where the entire dataset resides on each of the machines

The remainder of this section provides discussion on the different approaches, their advantages and disadvantages, and appropriateness with respect to different scenarios and goals.

7.4.2.1 Data Partitioning

While data partitioning looks as the more promising schema, it is also the one which is most problematic to implement. In general, it enables the management of larger volumes of data and provides more space for in-memory data structures. Each node can apply more efficient caching and optimization with respect to the fraction of the data that it deals with. Data partitioning with redundancy also allows for failover support. Still, the major issue is that query evaluation against distributed data requires intensive communication between the nodes for exchanging intermediate results; the most common variety of such communication is known as "remote join." Query optimization schemas, which consider the communication costs, are far harder to implement, which triggers less-optimal query evaluation plans and larger overall numbers of index lookups. In large number of scenarios, these effects neutralize the gains from the additional computing power gained from several machines. As outlined in Sect. 7.2.1.1, the same concerns are the application of rule-based reasoning in repositories using data partitioning.

Two approaches to data partitioning appear in database systems from the established distributed DBMS research: horizontal and vertical partitioning. The horizontal data partitioning approach partitions a dataset across several repository nodes where no schema limitations apply to any of the nodes. A vertical partitioning approach would assign different parts of the data schema to different repository nodes so that later on requests for any type of data would be redirected to the respective repository node. This approach can be further extended and types of data that usually appear "close" together can be placed within a single node (when possible). In principle, such clustering would allow for whole sub-queries to be executed within a single node. It would therefore avoid the transfer of intermediate results between the repository nodes and the central query processing node only to complete the query. Overall, one should consider that data partitioning comes together with problems which are not trivial to solve and imply

compromises in the engineering design. The CAP theorem [9], summarizes the results of significant amount of theoretical and experimental work in distributed database systems by stating the trade-offs between consistency, availability, and partition-tolerance. Hybrid schemes are possible for datasets that are both horizontally and vertically too large to fit in a single computational node. Such a partitioning scheme, however, would complicate considerably the query processing subsystem, which would have to take into account splits and joins in both dimensions while planning the query execution.

7.4.2.2 Data Replication

Data replication is a traditional approach for boosting the read performance of a DBMS at the cost of redundancy and write propagation complexity. In a classic scenario, several slave nodes are assigned incoming read requests by a central master node that performs any sort of load balancing (e.g., round robin) to distribute the load evenly across the slaves. Writes are executed on the master node, and updates are propagated to the slaves in the background. Such a setup is very appropriate in situations when a lot of read requests occur while write requests are rare or clustered together in large batches (for example, if a large dataset is initially loaded in the repository). In such situations, the resource-intensive replication procedure will not be necessary most of the time, while a theoretically linear scalability will take place on the reading side.

In an alternative replication implementation, write operations are propagated to all the slave nodes which operate as independent repositories. Such schema is implemented in BigOWLIM (see Sect. 7.6.4) and presented on *Fig. 7.7*. In this case, the total loading and modification performance of the cluster is equal to that of the fastest slave node. The data scalability of the cluster is bounded by those of the weakest slave node. Such design, however, simplifies the role of the mater node and the communication within the cluster, bringing principal advantages with respect to the resilience of the cluster.

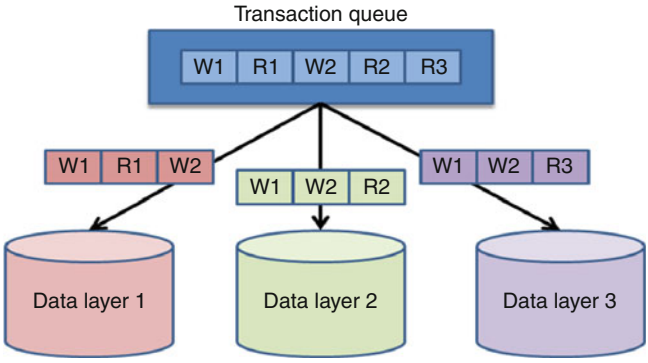

Fig. 7.7
Data replication with propagation of write operations

In case of failure of one or more instances, the performance degradation is graceful and the cluster will be fully operational even when there is only one instance working.

Using replication excludes the remote join problem, as each node can evaluate the query without exchanging information with the other nodes. As a result, there are no query performance issues, which facilitates a better handling of concurrent user requests. Load balancing and failover are easy to implement. The cluster can scale horizontally to handle virtually unlimited query loads, as the query evaluation capacity is a sum of the capacities of all the slave nodes in the cluster. Still, replication does not deliver better scalability with respect to larger datasets and the usage of resources is not optimal because all the loading and modifications need to be performed on all nodes.

7.4.2.3 Advantages of the Different Distribution Approaches

As an overview of the two major distribution approaches we can summarize that:

- Data partitioning improves data scalability; however, in most of the cases it hampers the query evaluation performance due to high communication overheads. It can improve loading performance if there is no materialization involved (see Sect. 7.2.1.2).
- Data replication allows for a better handling of concurrent query loads and failover. It is neutral with respect to loading and inference performance.

None of these approaches provide a principal advantage for the evaluation of complex queries. Under data replication, one of the nodes can be off-loaded from concurrent queries, which would allow faster execution of a complex query. An approach known as "federated join" can in theory improve the performance of such queries in very specific data partitioning scenarios, where the communication costs can be minimized.

As already mentioned, a direct consequence of the CAP theorem [9] is that data consistency might have to be sacrificed if availability and partitioning are required. One might argue that data inconsistency could be a serious issue in certain types of real-world production-level applications. An eventually consistent and highly scalable distributed system could be very well suited for the semantic repository of an incomplete reasoning system, where result completeness is not a requirement by design.

7.4.3 Multi-Core Parallelism

In addition to the various multi-node data distribution approaches, parallelization at thread level within a single computer system can also be considered. This is the approach of multi-core parallelism. These days even commodity desktop hardware is equipped with dual or even quad-core CPUs that are better utilized by multi-threaded implementations whenever these are appropriate. The cost-efficient DB servers (as the one presented in Sect. 7.4.1.2) come with 8–12 CPU cores, each of which is an autonomous computing device, while all cores share the same RAM. A multi-threaded semantic repository can benefit from parallel execution using multiple CPUs for computation, as with the

repositories distributed across several machines, but without the communication costs and overheads of the distributed approach, in particular without the remote join problem. One should consider that efficient multi-threading with respect to loading and modification requires nontrivial locking and synchronization, whereas multi-threaded handling of read-only operations (like query evaluation) is straightforward. One possible candidate for multi-threading is the "federated join" approach in which the outer-most loop of the query evaluation routine (i.e., the one traversing the outer-most triple pattern of the query) is split among several threads that operate over equal parts of the repository. This is another case that would be appropriate in scenarios when reads occur a lot and writes are rare.

Both Virtuoso and BigOWLIM support multi-threading and can efficiently use multi-core CPU as it becomes evident from the results of the BSBM benchmark presented in Sect. 7.2.1. As anticipated, multi-threaded query evaluation works without extra effort. The query throughput grows until the number of simultaneous users reaches the hardware support for parallelism. However, the usage of multi-threading for loading and inference requires special configuration. Local data partitioning is applied to minimize the interlocking between multiple threads used for loading and inference.

7.5 Popular Benchmarks and Datasets

The data stored in the semantic repositories form datasets. Their most distinctive features are their size and their complexity with respect to reasoning and querying. Benchmarks can be created through the combination of datasets and predefined sets of queries, which along with the datasets are commonly used as measuring sticks for evaluating the performance of the semantic repositories. A few of the most popular are presented here.

7.5.1 Lehigh University Benchmark (LUBM) and UOBM

The most popular benchmark for semantic repositories with support for RDF and OWL is Lehigh University Benchmark (LUBM), defined in [22]. The purpose of the benchmark is to measure the performance of storing and querying of large amounts of data that are created for realistic Semantic Web systems. It employs synthetically generated datasets using a fixed OWL ontology of university organizations, lecturers, teachers, students, and courses. The complexity of the language constructs used is between OWL Horst and OWL DLP [17], [50]. Fourteen queries are defined that are used to check the query evaluation correctness and speed of repositories that have loaded a given dataset. The biggest standard dataset is LUBM(50) (i.e., it contains synthetic data for 50 universities). Its size is 6.8 million explicit statements, distributed in 1,000 RDF/XML files with total size of 600 megabytes. For the purposes of scalability measurements, many groups have used the LUBM generator to create bigger datasets.

This is the dataset which is adapted practically by all major semantic repository vendors. Through the years, it played a considerable role in the development of the

field. Many of the scalable reasoning records were generated on the basis of LUBM – in 2009, 5 years after LUBM was published, BigOWLIM loaded a dataset of 90,000 universities. The major critiques against LUBM are as follows:

- The RDF graphs generated are very easy for partitioning and could provide misleading positive results for some query evaluation, reasoning, and distribution approaches that otherwise may not perform well on top of real-world datasets such as UNIPROT and DBPedia (described in the next sections).
- Some of the queries (most notably Q2 and Q9) return results, which are proportional to the size of the dataset, which for very large datasets is impractical and introduces distortion of the query evaluation results.

UOBM [40], is a benchmark that puts on test scalable ABox (instance data) reasoning against a relatively inexpressive DL ontology. UOBM is a further development of the LUBM benchmark; it uses the same evaluation framework (i.e., Java libraries), but provides alternative ontology, knowledge base, and queries, which allow for:

- More comprehensive coverage and usage of the OWL Lite and OWL DL semantics.
- Additional connections across the datasets for different universities – this modification assures higher level of connectivity in the RDF graph, which provides a more realistic and interesting test case.

The UOBM benchmark includes two distinct datasets for OWL Lite and OWL DL, each of which includes data collections for one, five, or ten universities, named respectively: Lite-1, Lite-5, Lite-10, DL-1, DL-5, and DL-10. No dataset generator for UOBM is publicly available at present. The Lite version of the test contains 13 queries; the DL version adds two more. Both the Lite and the DL variants of UOBM require considerably more complex reasoning to be performed by a semantic repository in order to provide correct answers to the queries.

7.5.2 UniProt

UniProt (Universal Protein Resource, http://www.uniprot.org) is the world's most comprehensive and most popular dataset of information on proteins, created by joining several other resources (Swiss-Prot, TrEMBL, and PIR). UniProt RDF, [59], is an RDF representation of the dataset; it is based on an OWL ontology, expressed in a sub-language of OWL Lite, that is more expressive than OWL Horst, but still tractable (see Sect. 7.2.2 for information on the different OWL dialects). It represents one of the largest datasets distributed in RDF and OWL. Processing UniProt is often used as benchmark for scalability and reasoning capabilities of semantic repositories. Still, very few of the repositories are capable of loading UniProt and perform materialization on top of it or to interpret its semantics otherwise.

The size of the RDF representation of UniProt is above 1.5 billion unique explicit statements. The RDF graph defined in the UniProt ontology is highly interconnected,

which has significant impact on the loading and reasoning speed of the semantic repositories. Today UniProt is one of the central datasets in the biomedical part of the Linking Open Data (LOD) initiative [38], [39].

The RDF representation of UniProt was used for benchmarking Jena and AllegroGraph (see Sects. 7.6.6 and 7.6.2). It is also part of the Pathway and Interaction Knowledge Base (PIKB), [2], and the LinkedLifeData service presented in Sect. 7.8.1.2.

7.5.3 DBPedia and Other Linked Datasets

DBPedia (More information about DBPedia is provided by its developers in Semantic Annotation and Retrieval: Web of Data) is a dataset represented in RDF the Infobox of Wikipedia together with other information related to or derived from Wikipedia. It serves as a connectivity hub of the Linked Open Data (LOD) initiative, [39]. The diversity of the information represented in DBPedia and the fact that it represents encyclopedic knowledge make it an excellent resource for benchmarking semantic repositories. DBPedia version 3.3 includes 362 million unique statements without the `owl:sameAs` links.

FactForge (Sect. 7.8.1.1) is probably the largest and most heterogeneous body of general factual knowledge that was ever used for logical inference. It integrates in a single repository several of the most central datasets of Linking Open Data (LOD) cloud [39]. When used for benchmarking of semantic repositories, it has several advantages, compared to the straightforward usage of DBPedia:

- It is more diverse, as it represents data and data modeling patterns from several other datasets as well. One should consider for instance that DBPedia and Geonames are datasets of a very different nature. A repository, which performs well on Geonames could show poor performance when dealing with DBPedia for various reasons; one of them being that Geonames uses few tens of different predicates, whereas DBPedia uses around a hundred thousand predicates.
- The data in FactForge can be used for inference, because they were cleaned up and prepared to allow this. On the other hand, reasoning with the raw DBPedia data is inappropriate as it results in the inference of a very large number of false and useless statements; the latter happens mostly due to problems with the category hierarchy discussed in [34].

7.5.4 Berlin SPARQL Benchmark (BSBM)

Berlin SPARQL Benchmark, [6], evaluates the performance of query engines in an eCommerce use case: searching products and navigating through related information. Randomized "query mixes" (of 25 queries each) are evaluated continuously through a client application that communicates with the repository through a SPARQL endpoint.

The benchmark enables evaluation with respect to the changing sizes of the dataset and differing numbers of simultaneous clients.

Although created for the benchmarking of SPARQL engines, the design of BSBM favors relational databases and other raw-store-based implementations, as long as they deal with a single fixed data schema and uniform dense data representation. Generally, RDF databases are designed to deal efficiently with diverse data, integrated from multiple data sources, encoded against different data schema, resulting in sparse data tables in relational databases (see Sect. 7.1.2).

BSBM supports the benchmarking of relational engines, as there is an SQL-based version of the dataset and the queries. One should note that the semantics of some of the queries in the SQL version is simpler than those of their SPARQL equivalents, that is, the SQL versions are less powerful and return different results. The evaluation results published in [7] prove that relational databases are really more suitable for such loads – considering that the results are not truly comparable, it is still amazing that the relational engines are one order of magnitude faster.

Finally, unlike LUBM (Sect. 7.5.1), BSBM does not require any inference – an engine can deliver correct results of the queries without any interpretation of the semantics of the data. Still, the benchmark is useful for evaluation of the SPARQL support of the engines and their efficiency in handling multi-client loads.

7.6 Semantic Repository Engines

Several of today's outstanding semantic repositories are presented in alphabetical order in this section with discussion on their specific advantages and features.

7.6.1 3store, 4store, 5store

3store, 4store, and 5store represent a family of RDF database engines developed by Garlik (http://www.garlik.com/) – a company dealing with online identity and personal information protection in UK. 4store (http://www.4store.org) is an open-source RDF storage engine designed to run in cluster setup of up to 32 nodes. 4store implements a distribution schema based on data partitioning (see Sect. 7.4.2.1); more details about it are available in [23]. While 4store does not offer inference capabilities, [57] presents the 4s-reasoner, which implements RDF reasoning through backward-chaining on top of 4store.

It is implemented in ANSI C99 and is available for UNIX-based operating systems only, including Linux and MacOS. 4store is advertised with the fact that at Garlik it is "holding and running queries over databases of 15GT, supporting a Web application used by thousands of people." The RDF repositories created with 4store are managed through a set of command line utilities that allow the importing of RDF data in various formats,

querying using SPARQL, backup, and other maintenance tasks. The 4store distribution also includes a stand-alone SPARQL HTTP protocol server as a standard interface to the 4store repository.

As the names suggest, 3store is the predecessor and 5store is the successor of 4store. According to the website of 5store (http://4store.org/trac/wiki/5store), it offers the same functionality and interfaces as 4store, but features a new architecture which allows for the handling of larger clusters and better performance. There is no public information about benchmarking 4store or 5store with respect to popular benchmark datasets; there are also no publications about independent performance evaluations of 4store and 5store.

7.6.2 AllegroGraph

AllegroGraph RDFStore, [1, 18], is an RDF database with support for "SPARQL, RDFS++, and Prolog reasoning from Java applications." In addition to the "basic" DBMS functionality, it offers specific support for geo-spatial data handling and social network analysis.

AllegroGraph does not perform reasoning and materialization during loading. The so-called RDFS++ reasoning can be switched on during query processing. The semantics supported in this way is comparable to OWL Horst ([17]). RDFS++ reasoning allows AllegroGraph to deliver correct results on LUBM benchmark. Versions 3.2 and 4.0 of AllegroGraph support the so-called RDFS++ dynamic materialization, [20], which seems to require building some extra indices on-the-fly, whenever those are required for query evaluation. This approach helps AllegroGraph report excellent results, [19], on query evaluation in LUBM(8000).

One of the major developments in AllegroGraph 3.0 is the "federation" that enables the grouping multiple stores (running locally or on a remote machine) within a single virtual store. Federation has the potential to considerably speed up the loading process, as the work is effectively distributed across multiple stores. The distribution approach implemented in AllegroGraph is based on data partitioning (see Sect. 7.4.2.1).

7.6.3 BigData

BigData (http://www.systap.com/bigdata.htm) is an open-source distributed B + Tree database, designed to accommodate large RDF repositories and scale horizontally on commodity hardware. Scaling is achieved by index partitioning across the nodes of the cluster. As the data evolve, partitions of the indices might get split, moved, or joined transparently and asynchronously to the client. A centralized metadata index takes care of locating the index partitions in the cluster. As discussed in [60] and [61], BigData is

designed to support various modalities of transactional isolation through multi-version concurrency control (MVCC, [55]) and aims to support load balancing internally.

BigData integrates with the Sesame 2.0 platform (see Sect. 7.6.8) to achieve SPARQL support and therefore relies on it for query parsing but implements query optimization based on the fast key-range counts supported natively by the B + Tree architecture.

BigData supports a sort of RDFS + inferencing backed by a hybrid approach of materializing the entailments of some rules (forward-chaining) at load time and using backward-chaining for others at query time. Other features of BigData include statement-level provenance and full-text indexing.

7.6.4 BigOWLIM

OWLIM is a semantic repository implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database (see Sect. 7.6.8). OWLIM is based on TRREE – a native RDF rule-entailment engine. The standard inference strategy is forward-chaining and materialization. The supported semantics can be configured through rule-set definition and selection. The most expressive predefined rule-set is OWL 2 RL, [45]. Custom rule-sets allow tuning for optimal performance and expressiveness.

The two major varieties of OWLIM are SwiftOWLIM and BigOWLIM. In SwiftOWLIM, reasoning and query evaluation are performed in memory, while, at the same time, a reliable persistence strategy assures data preservation, consistency, and integrity. BigOWLIM is the "enterprise" variety that deals with data and indices directly from disk or other file storage, which allows it to scale more comprehensively. Further, BigOWLIM's indices are specially designed to allow efficient query evaluation against huge volumes of data. SwiftOWLIM can manage millions of explicit statements on desktop hardware. Given an entry-level server, BigOWLIM can handle billions of statements and serve multiple simultaneous user sessions.

The most advanced features of version BigOWLIM 3.3, released in June 2010 and used in BBC's World Cup 2010 website (see Sect. 7.8.2), can be summarized as follows:

- Pure Java implementation compliant with Sesame (see Sect. 7.6.8). The latter brings interoperability benefits and support for all popular RDF syntaxes and query languages, including SPARQL
- Clustering support brings resilience, failover, and horizontally scalable parallel query processing (see Sect. 7.4.2.2)
- Optimized handling of `owl:sameAs` (see Sect. 7.2.1.5), which delivers considerable improvements in performance and usability when huge volumes of data from multiple sources are integrated
- Full-text search, based on either Lucene or proprietary techniques
- High-performance retraction of statements and their inferences (see Sect. 7.2.1.6)

- Logical consistency checking mechanisms, as those supported in OWL 2 RL and OWL Horst (see Sect. 7.2.2)
- RDF rank, similar to Google's PageRank, can be calculated for the nodes in an RDF graph and used for ordering query results by relevance
- A notification mechanism, to allow clients to react to updates in the data stream

A feature of BigOWLIM 3.3 which deserves special attention is the so-called RDF Search, which provides a novel method for the schema-agnostic retrieval of data from RDF datasets. The main rationale behind RDF search is to allow one to search in an RDF graph by keywords and get usable results (stand-alone literals are not useful in many cases). Technically, it involves the full-text indexing of the URIs in the RDF graph with respect to their "text molecules" – text snippet achieved by means of the concatenation of text from all the nodes in the RDF-molecule of the corresponding URI. The result is list of URIs, ranked with a metric combining the standard full-text search Vector Space Model and the RDFRank. In FactForge (see Sect. 7.8.1.1), each of the URIs in the result list is presented with human-readable labels and text snippets.

7.6.5 DAML DB, Asio Parliament

DAML DB is an older name of the engine of BBN Technologies, which currently appears as part of Parliament (http://www.bbn.com/technology/knowledge/parliament) – an open-source knowledge base management system that implements a high-performance storage engine, compatible with the RDF and OWL standards. It is usually combined with query processing frameworks, such as Sesame (Sect. 7.6.8) or Jena (Sect. 7.6.6), to implement a complete data management solution with support for SPARQL query language.

Although there are no recent evaluation results published, DAML DB still seems to be one of the very few systems that can demonstrate a full-cycle benchmark results (see Sect. 7.3.4) on test like LUBM(8000).

7.6.6 Jena TDB

TDB (http://jena.hpl.hp.com/wiki/TDB) is an open-source RDF storage layer for the Jena Semantic Web Java framework (http://jena.sourceforge.net/). It is implemented purely in Java and can be accessed through Jena APIs or through several separately provided command line scripts. Paired with Jena, TDB provides a single-machine, non-transactional RDF storage and query environment that can be accessed over the SPARQL protocol when running inside the Jena-based Joseki HTTP server. SPARQL queries over TDB are made possible through Jena's ARQ SPARQL query engine.

A TDB repository can be operated by 32-bit and 64-bit JVM without any format migration being necessary (a direct consequence of Java's architecture independent types).

However, in 64-bit mode, TDB uses memory-mapped files to access its repository binary representation (data and indices). This is reported to contribute to a much better performance compared to the 32-bit case where data caching is handled by the TDB engine itself. Another benefit from relying on the operating system to do the file caching is the dynamically determined amount of memory used by the engine for disk caches.

TDB is equipped with different SPARQL query optimizers (e.g., fixed and statistical) that aim to optimize SPARQL queries on a per-repository basis. The fixed optimizer only considers the number of variables in a query's triple pattern in its reordering decision. The statistical optimizer relies on rules that approximate the number of results to be expected from a triple pattern (those rules could be either written manually or generated by the engine). TDB interprets RDF simple types (e.g., `xsd:integer`), which makes it possible to optimize SPARQL range filters.

7.6.7 ORACLE

ORACLE offers RDF support as part of the Spatial option of its DBMS since version 10 g R2. As presented in [69], in version 11 g R2, this support is improved in several ways, including:

- Support for the OWL Prime dialect, which is comparable with the owl-max semantics of OWLIM, [50]. The Pellet DL reasoner is integrated for T-Box (schema level) inference.
- The efficiency of RDF loading and inference is considerably improved.

ORACLE supports RDF models with named graphs, that is, it can be seen as a quadruple store. The semantics to be used for inference is defined in terms of rule-bases, which essentially represent sets of entailment rules. Inference is implemented in terms of forward-chaining and materialization – the results are stored in rule indices. The initiative regarding inference is given to the user, who should take care of it explicitly:

- Force inference, that is, generation of the rule indices.
- Invalidate the rule indices when necessary, that is, after statements are deleted.

The latest results from benchmarking ORACLE 11 g are published in [65]. One should consider summing up the times for the different steps. For instance, the times reported for LUBM(8000) are as follows:

- Bulk-load: 30 h. 43 min.
- Loading into staging table: 11 h. 32 min. (when proper correctness checks are performed by the RDF parser).
- Inference (OWL Prime): 11 h. Multi-threaded inference runs 3.3 times faster as compared to single-threaded one on a quad-core CPU.

Two important circumstances should be acknowledged:

- The LUBM(8000) results for ORACLE are measured on desktop computer, while most of the other results are measured on servers. In fact, the configuration used for inference is comparable to the workstation used for benchmarking BigOWLIM on LUBM(8000).
- ORACLE is also the most "economic" with respect to the RAM usage – the above results reflect the inference run with 4 GB of RAM, but results measured on 2 GB systems suggest graceful degradation of the performance when less memory is available.
- The results reported for loading and for inference come from different machines; there are no public results for loading times at the same CPU where ORACLE achieves its best inference time.

7.6.8 Sesame

Sesame is one of the most popular semantic repositories that supports RDF(S) and all the major syntaxes and query languages related to it. Sesame is ranked by multiple independent evaluations (e.g., [53]) as the most efficient RDF repository framework. Several engines rely on the Sesame RDF database framework (http://www.openrdf.org). Semantic repositories like OWLIM and BigData (see Sects. 7.6.3 and 7.6.4) use Sesame as a library, taking advantage of its APIs for storage and querying, as well as the support for a wide variety of query languages (e.g., SPARQL and SeRQL) and RDF syntaxes (e.g., RDF/XML, N3, Turtle). Other engines like Virtuoso and AllegroGraph use it just for the sake of interoperability.

7.6.9 Virtuoso

OpenLink Virtuoso (http://www.openlinksw.com/virtuoso/) is a "universal server" offering diverse data and metadata management facilities, for example, XML management, RDBMS integration, full-text indexing, etc. The core engine of Virtuoso is a relational database engine with numerous RDF-oriented adaptations in datatypes, index layout, and query optimization.

Virtuoso does not have built-in materialization of entailment during loading. Instead, it supports the semantics related to subclasses, sub-properties, and owl:sameAs at runtime, through backward-chaining on the basis of a specified RDF schema. Thus, the user is not required to rewrite queries. Materialization is possible by writing SPARUL statements to generate implied triples.

Virtuoso uses bitmap indices with key compression to address the issue of space efficiency, and samples the database at query optimization time by keeping the data in memory to address the issue of speed of processing. It is designed with partitioning, specified at the index level – a hash partitioning where the hash picks a logical partition out of a space

of n logical partitions, where n is a number several times larger than the expected maximum machine count. Each logical partition is then assigned to a physical machine. When loading RDF data, the database translates between IRI's and literals and their internal identifiers. It then inserts the resulting quad in each index. An RDF query consists of a single key lookups grouped in nested loop joins with occasional bitmap intersections. The basic query is a pipeline of steps where most steps are individually partitioned operations. The partitioned pipe function may return a set of follow-up functions and their arguments, which get partitioned and dispatched in turn. In Virtuoso, each logical partition is allocated on multiple nodes. At query time, a randomly selected partition is used to answer the query if the data are not local. At update time, all copies are updated in the same transaction. This is replicated on all nodes. When loading data at a rate of 40Ktriples/s, the network traffic is 170 messages/s and the aggregate throughput is 10 MB/s with no real network congestion. Scale can be increased without hitting a network bottleneck. Thus Virtuoso can run complex queries with a reasonable number of messages, about 1620/34 = 47 messages per query. As data are becoming a utility, the objective of Virtuoso and semantic repositories in general is to provide for the rapid deployment of arbitrary scale RDF and other database systems for the clouds. This involves automatic partitioning and re-partitioning and adapting query planning cost models to data that contain increasing inference.

The developers of Virtuoso report in [14] the results of various "distributions of labor" between materialization and query expansion. As expected, it appears that extensive materialization can save a lot of computations during querying, while the same query completeness is achieved. In principle, this type of inference can be implemented on top of any DBMS – the complexity of the inference depends on the complexity of the query language supported. Here the semantics is not enforced by the engine, but it is rather a matter of handcrafted queries that may or may not correctly reflect the semantics of the ontology language primitives.

The data provided in [14] leave several questions open:

- The queries of LUBM are modified, so it is not clear: (1) how the query evaluation performance compares with that of other engines benchmarked with the original LUBM queries and (2) whether the query results are truly complete and whether the implemented inference mechanisms are correct and sufficient for a full LUBM run.
- No performance data are provided for entailment and materialization, that is, there are no indications about the performance and efficiency of the various inference configurations that were put on test.
- Implementing the semantics of transitive properties via query-based entailment and materialization, as presented in [9], requires recursive query evaluation. However, there are no comments on the implementation of such behavior in Virtuoso.

Version 6.0 is the latest generation of the Virtuoso engine, which supports distributed RDF database management. The results (http://www.openlinksw.com/dataspace/vdb/weblog/vdb%27s%20BLOG%20%5B136%5D/1563) from the loading of LUBM(8000) in a cluster of eight instances running on a single dual-CPU, eight-core, server represent the fastest load on this benchmark on a server worth less than $10,000.

7.7 State of the Art in Performance and Scalability

This section provides a discussion on the current state of the art with respect to scalability and the different aspects of the performance (outlined in Sect. 7.3). Reference [33] presents an overview and comparison of the most relevant publicly available benchmark results from several of the most prominent semantic repositories. Such a comparison is not presented here because in the year 2010, the rate of publication of new results is already quite high, which means that any specific results will be out of date soon. Further, the results being published are very hard to compare in a reliable manner for several reasons: Most of the vendors do not publish sufficient information about the benchmarking experiments; most of the information published does not comply with the full-cycle benchmarking principles (presented in Sect. 7.3.4); the hardware used by the different vendors is very different in terms of both price and performance. Instead, the remainder of this section generalizes some trends and refers to reports on independent evaluations and comparisons of semantic repositories, such as [63] and [8].

7.7.1 Data Loading Performance

The current publications at the "Large Triple Stores" page (a wiki page maintained by W3C at http://esw.w3.org/LargeTripleStores, where vendors report on their scalability achievements) suggest that there are several repositories which can load RDF datasets of a size between 10 and 20 billion statements. Most of the vendors report such results using servers similar to the one presented in Sect. 7.4.1.2. There are however certain differences:

- According to a post at the "Large Triple Store" (no further information is provided elsewhere), a Virtuoso (See Sect. 7.6.9 for more information about the engine) cluster of eight servers has loaded 15 billion statements of the LOD Cloud Cache (http://lod.openlinksw.com/) service. No average loading speed for the entire dataset is provided, here is a quote "*Latest bulk load added ~3 Billion triples in ~3 h — roughly 275Ktps (Kilotriples-per-second) — with partial parallelization of load.*" Materialization is not performed
- According to [33], BigOWLIM (See Sect. 7.6.4 for more information about the engine) uses a single server to load the 12 billion statements of LUBM(90,000), perform materialization, and index 20 billion statements. Two speeds should be considered here because of the materialization: 12,000 statements/s. is the speed of loading explicit statements; 18,000 statements/s. is the speed of indexing statements
- Again according to a post at the "Large Triple Store" with no further information provided elsewhere, 4Store (See Sect. 7.6.1 for more information about the engine) ". . . is running with 15B triples in a production cluster". While no proper reference is provided for the average loading speed, values in the range of 140,000 statements per second are indicated. 4Store does not perform materialization.

To summarize, systems which do not perform materialization and feature data partitioning demonstrate speeds in the range of few hundred thousand statements per second. Still, to understand the importance of these achievements, one should be able to analyze also query evaluation correctness and performance. This is important in order to distinguish really comprehensive implementations, on the one end of the spectrum, from a trivially partitioned store (TPS) like the following one, on the other end. Suppose that TPS is a distributed repository which is implemented as a cluster of N nodes, each of which being a stand-alone semantic repository. New statements are given a hash code and distributed to one of the nodes; materialization is not supported. All low-level read requests are broadcasted to all the nodes, and the results are federated at the master node. Such a TPS will exhibit perfect horizontal scalability with respect to the volumes of data and the speed of loading; given a hash function which distributes the statements evenly, the loading speed of the TPS will be close to the sum of the speeds of all the nodes. However, the query evaluation speed of the TPS will be very poor due to the high communication costs.

Under full-cycle benchmarking conditions, the best loading speeds for datasets in the range of one billion statements are in the range of 100,000 statements per second; such is the case of AllegroGraph's multi-threaded loads, [19]. As expected, and as observed in the results comparison in [33], materialization with respect to OWL Horst-like languages, even for the synthetic datasets of LUBM (See Sect. 7.5.1), brings the loading speed down to the levels of a few tens of thousands of statements. This is the case even with multi-threaded loading and inference implementations as the one of ORACLE, [69].

Probably the most interesting and useful performance analysis results are presented in [63] where several of the outstanding repositories are evaluated with respect to a real-world scenario. The repository is populated with: an ontology; a factual knowledge base of about seven million statements; and eight million statements of metadata about a collection of Press Association, containing five million images. Next 15 queries relevant to a content management and Web publishing are evaluated. The scenario requires interpretation of the ontology with respect to OWL Horst semantics. The test setup makes no assumptions whether reasoning is implemented through forward-chaining and materialization during loading or through backward-chaining during query evaluation. AllegroGraph loads the data about ten times faster than the engines which perform materialization (Sesame, Jena TDB, and BigOWLIM); as expected, this has implications on the query evaluation times presented in the next section.

7.7.2 Query Evaluation

Loading, combined with query evaluation, can deliver full-cycle benchmarking (Sect. 7.3.4), i.e., an adequate picture of the performance of the engine. Below we provide publicly available results about query evaluation performance of the engines presented in Sect. 7.6 based on two benchmarks: BSBM (Sect. 7.5.4) and LUBM (Sect. 7.5.1). We will start, however, with the result of a query evaluation benchmarking

a real-world scenario of Press Association, reported in [63] and introduced in Sect. 7.7.1. While AllegroGraph was much faster in the loading of the data, it fails to answer two of the queries and its average query evaluation time is 50 times slower than that of BigOWLIM (8.2 s. versus 0.16 s.) This is a clear demonstration of the trade-offs between the two main reasoning strategies (see Sect. 7.2.1.2).

7.7.2.1 BSBM Results

Recent results from evaluation of few of the most popular engines with the BSBM benchmark (see Sect. 7.5.4) are published in [7] and [8]. The former includes: relational database engines (running the SQL version of the benchmark), relation-to-RDF wrappers, and native RDF engines. As long as BSBM is biased toward relational databases, it is not a surprise that the relational engines (MS SQL and Virtuoso SQL) perform far better than the native RDF stores and the relational-to-RDF wrappers are somewhere in the middle (*Fig. 7.8*).

Figure 7.9 provides a comparison between results from the internal evaluation of BigOWLIM 3.1 and results for other systems from [7] regarding query evaluation with a growing number of simultaneous clients (1, 4, 8) against 25-million statement version of the BSBM dataset. Unfortunately, there are no public BSBM results on the evaluation of engines, other than OWLIM, with more than four clients for datasets larger than 25 M. On the other hand, for the most mature engines, the indices of the 25 M dataset fit into the main memory of any machine with more than 4 GB of RAM. Still, the results demonstrate the degree to which the engines can parallelize query processing in a way which utilizes the hardware more efficiently.

Given a larger number of simultaneous clients, BigOWLIM and Virtuoso can deliver considerably larger overall throughput, almost proportional to the number of CPU cores of the system. BigOWLIM's results were acquired on a desktop (osol) with a quad-core CPU with hyper-threading support; Virtuoso's results were acquired on a desktop quad-core CPU without hyper-threading.

Fig. 7.8
BSBM query results: Multiple clients

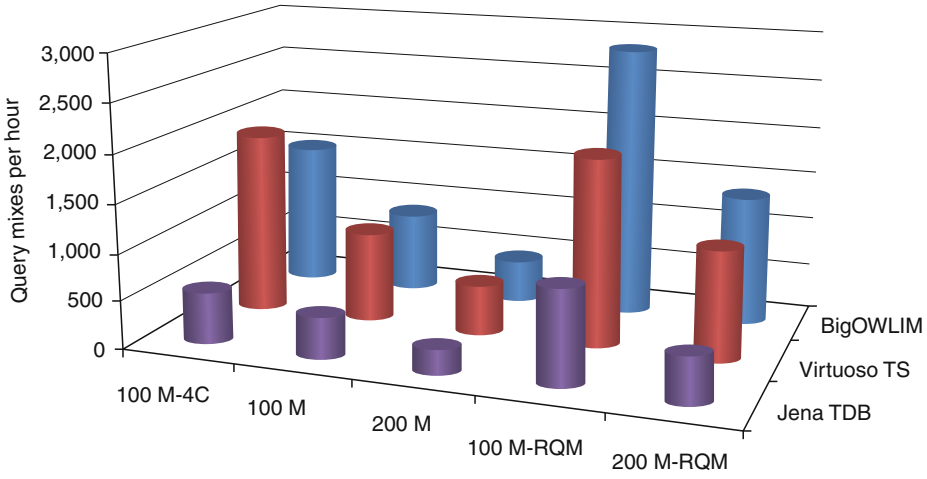

Fig. 7.9
BSBM query results: Large datasets and reduced Query mix

In subsequent experiments, [8], the authors of BSBM evaluated the performance of BigOWLIM, Jena TDB, and the native RDF Virtuoso engine. Excerpts of these results are presented in *Fig. 7.9*. The experiments included two scales: 100 M and 200 M. Apart from the standard runs, there are also results for two variants: running with four clients (indicated on the diagram with "4C") and running a "reduced query mix" (indicated with RQM). The reduction of the query mix was made by removing a couple of queries, which are not suitable for benchmarks at scales above 25 M; at large scale, they consume above 80% of the query time for the entire mix. Analysis of these results follows:

- As already observed on *Fig. 7.9*, Virtuoso and BigOWLIM are efficient in parallelizing the query evaluation.
- Both Virtuoso and BigOWLIM can handle about 10 nontrivial queries per second against 200 M dataset, even running in single-client mode on desktop machine worth less than $1,000.

7.7.2.2 LUBM Results

Although there are plenty of data on loading performance and the scalability of LUBM, in a very few cases, query evaluation has been benchmarked within the same environment that was used for dataset loading.

The table below presents the results of query evaluation of the LUBM benchmark with 8,000 universities. Two sets of results are provided for BigOWLIM 3.1 – for server (onap) and desktop (osol). The results of AllegroGraph 3.2, [19], were acquired on a server comparable to the one used for BigOWLIM. The semantic repository engines in question are described in Sect. 7.6, and specifications of the benchmark environments are given in Sect. 7.7.1.

Query performance LUBM(8000), one billion statements

Query No	# of results	BigOWLIM 3.1 (onap)		BigOWLIM 3.1 (osol)		AllegroGraph 3.2	
		Time (msec)	QTPR (msec)	Time (msec)	QTPR (msec)	Time (msec)	QTPR (msec)
1	4	23	5.8	52	13.0	4	1.0
2	2,528	251,992	99.7	873,197	345.4	54,964	21.7
3	6	26	4.3	25	4.2	2	0.3
4	34	8	0.2	28	0.8	14	0.4
5	719	38	0.1	29	0.0	37	0.1
6	83,557,706	84,333	0.0	174,777	0.0	104,794	0.0
7	67	38	0.6	71	1.1	9	0.1
8	7,790	113	0.0	132	0.0	178	0.0
9	2,178,420	523,215	0.2	1,460,862	0.7	117,303	0.1
10	4	27	6.8	24	6.0	5	1.3
11	224	4	0.0	23	0.1	8	0.0
12	15	6	0.4	33	2.2	14	0.9
13	37,118	1,662	0.0	121,980	3.3	47	0.0
14	63,400,587	63,998	0.0	157,568	0.0	27,990	0.0
Average		66,106	8.4	199,200	26.9	21,812	1.9

Considering that some of the LUBM queries return a very large number of results for large datasets, the query-time-per-result (QTPR) metric is introduced, where the query evaluation time is normalized with respect to the size of the result-set. Average QTPR represents a better overall score of the query performance on LUBM because the performance of queries with large result-sets does not dominate the score as in the case of using average query evaluation time. QTPR appearing as 0.0 indicates a value below 0.1 milliseconds.

As expected, given a large dataset, the query performance of the server is considerably better than that of the desktop. This should most probably be attributed to the better storage system and data buses between the CPUs and the main memory.

The query performance reported for AllegroGraph looks very good, especially if one considers that, based on the vendor information, "dynamic materialization" is performed during query evaluation. The major gain in terms of average QTPR and evaluation time comes from queries #2 and #9 – probably the two most challenging in the benchmark.

This is the state of the art in performance and scalability of semantic repositories. Dataset loading and query evaluation times are observed, considering the hardware configurations and the overall test environment.

7.8 Typical Applications

The applications of semantic repositories are presented in this section to illustrate their strengths across different domains and scenarios. The following table compares the most relevant characteristics of these applications.

Comparison of sample semantic repository applications

	FactForge	LinkedLifeData	BBC's World Cup website
Scope	General	Domain-specific	Domain- and application-specific
Closed-world assumption	–	+	+
Data management approach	Integration with minimal intervention	Extensive data restructuring	Well-maintained schema and data, extended using entity extraction
Update rate	Once in a few months	Once in a few months	Hundreds of updates per hour

7.8.1 Reason-Able Views to the Web of Linked Data

Reason-able views represent an approach for reasoning and the management of linked data from the Linking Open Data (LOD) cloud (see Sect. 7.1.4). *Reason-able view (RAV)* is an assembly of independent datasets, which can be used as a single body of knowledge with respect to reasoning and query evaluation.

Reason-able views are necessary as reasoning with linked data is problematic with respect to different dimensions. For example, reasoning with the Web of linked data is not feasible because of the clash between the mainstream reasoning techniques and the WWW-like nature of the data such as Linking Open Data (LOD). The major obstacles for this lay in several reasons:

- Most of the popular reasoners implement sound and complete inference under "closed-world assumption." Such setups are irrelevant in an environment, like the Web of Data and the WWW, where the knowledge is incomplete by design and logical consistency is not guaranteed.
- In addition to that, the complexity of reasoning even with the simplest knowledge representation language based on description logics (DL – one of the most popular paradigms nowadays) OWL Lite is prohibitatively high when applied to Linking Open Data (LOD). The great complexity of the algorithms for basic reasoning tasks indicates that they are unfeasible for application to large-scale knowledge bases and datasets.
- Further, the LOD cloud contains datasets that are not suitable for reasoning. Some of them are derived by the means of text-mining and include incorrect information, due

to the intrinsic limitations of the accuracy of the extraction techniques. Such inaccuracies are probably not a serious problem for human readers, but they can lead to significant noise and inconsistencies by automated reasoning.
- Finally, data publishers use OWL vocabulary with no account for its formal semantics. This results in long cycles in many category hierarchies.

Reasoning is practically unfeasible with distributed data as well. It is actually possible, but is usually far slower than reasoning with local data. The fundamental reason for that is related to the "remote join" problem. The remote join problem pertains to DBMS (Database Management Systems). The remote join is a method capable of executing a join across two DBMSs. The remote join problem is solved by creating a join view at the remote database, and a local synonym for this view, then retrieving through this local view. Distributed joins have caused performance problems throughout the history of distributed database support in general. That is why they cause problems in the context of reasoning with the Linking Open Data (LOD) cloud as well. In addition to this, this speed of access and service availability in distributed data can be a major issue regarding reasoning and manipulating the Web of Data in distributed environments.

Thus, the key ideas around the *reason-able views* approach focus on the following aspects:

- The grouping of the selected datasets and ontologies in a compound dataset, which becomes a single body of knowledge – *integrated dataset* – with respect to reasoning and query evaluation.
- Loading of the compound dataset in a single semantic repository in order to make query evaluation and reasoning practically feasible. It can be considered as an index, which caches parts of the Linking Open Data (LOD) cloud and provides access to the datasets included as Web search engines index WWW pages and facilitate their usage.
- The performance of inference with respect to tractable OWL dialects. Given all public results, only OWL Horst (see Sect. 7.2.2)–like languages seem to be suitable for reasoning with data in the range of billions of statements.

Complying with all these aspects makes reasoning with the Linking Open Data (LOD) feasible. This is because a basic level of consistency of the data is being guaranteed, along with a guaranteed service availability because the compound dataset is loaded into a single semantic repository. This allows for the easier exploration and querying of unseen data and ensures a lower cost of entry.

The constitution of reason-able views obeys special selection criteria for the datasets, for example, the datasets must allow inference and deliver meaningful results under the semantics determined for the view. Further, it is necessary that the datasets are easy to define and isolate, for example, they must be clearly distinguishable from other datasets. In many cases, additional manipulations on the datasets like cleanups are required. The datasets must allow easy and cheap cleanup manipulations that can be performed on them in an automated or semiautomated fashion. Ultimately, the datasets must be more or less static, able to function in predictable way, opposite to database wrappers which

implement complex mappings to be reusable in unplanned contexts, such as Web-based applications or federated systems, where RDF is generated in answer to retrieval requests.

There are two implementations of the concept for linked data reason-able views:

- FactForge, in red on *Fig. 7.10* (see Sect. 7.8.1.1)
- Linked Life Data (LLD) and PIKB Pathway and Interaction Knowledge Base, (see Sect. 7.8.1.2)

Although similar in spirit and based on identical principles, these two applications are very different use cases. FactForge is a collection of a vast amount of heterogeneous general purpose data, whereas LLD – PIKB is a domain-specific warehouse.

FactForge and LLD – PIKB are presented in greater detail in the following sections. While both of them can be seen as reason-able views to the Web of linked data (a notion introduced in Sect. 7.8.1), sharing one and the same search, exploration, and querying facilities, they are in fact quite different in terms of the data management approach, assumptions, and users.

7.8.1.1 FactForge: The Upper-Level Knowledge Base

FactForge is a reason-able view to the Web of linked data, an assembly of some of the central LOD datasets, which have been selected and refined in order to:

- Serve as a useful index and entry point to the LOD cloud
- Present a good use case for large-scale reasoning and data integration

It includes DBPedia, Geonames, UMBEL, Freebase, WordNet, CIA World Factbook, Lingvoj datasets and Dublin Core (DC), SKOS (Simple Knowledge Organization System), RSS, FOAF schemas.

The datasets of FactForge are loaded into BigOWLIM, where forward-chaining and materialization are performed. BigOWLIM uses internally a rule language that supports R-entailment – "owl-max," which extends OWL Horst, [60], to deliver expressiveness very similar to OWL 2 RL, [41]. The standard reasoning behavior of OWLIM is to update the deductive closure upon committing a transaction to the repository. Consistency checking is performed, applying the checking rules after adding all new statements and updating the deductive closure. Inconsistencies are reported accordingly. FactForge is loaded with OWLIM's "partialRDFS" option enabled. This excludes rules supporting some of the features from RDFS and OWL, thus avoiding inferring and indexing three "trivial" statements for each URI in the repository.

Additionally, the loading of FactForge benefits from a specific feature of the BigTRREE engine (see Sect. 7.6.4) that enables the engine to handle efficiently `owl:sameAs` statements in order to avoid their over-generation (see Sect. 7.2.1.5). The results of the loading of FactForge can be summarized as follows:

- Number of inserted statements (NIS): 1.4 billion
- Number of stored statements (NSS), including the implicit ones: 2.2 billion
- Number of retrievable statements (NRS): 10 billion

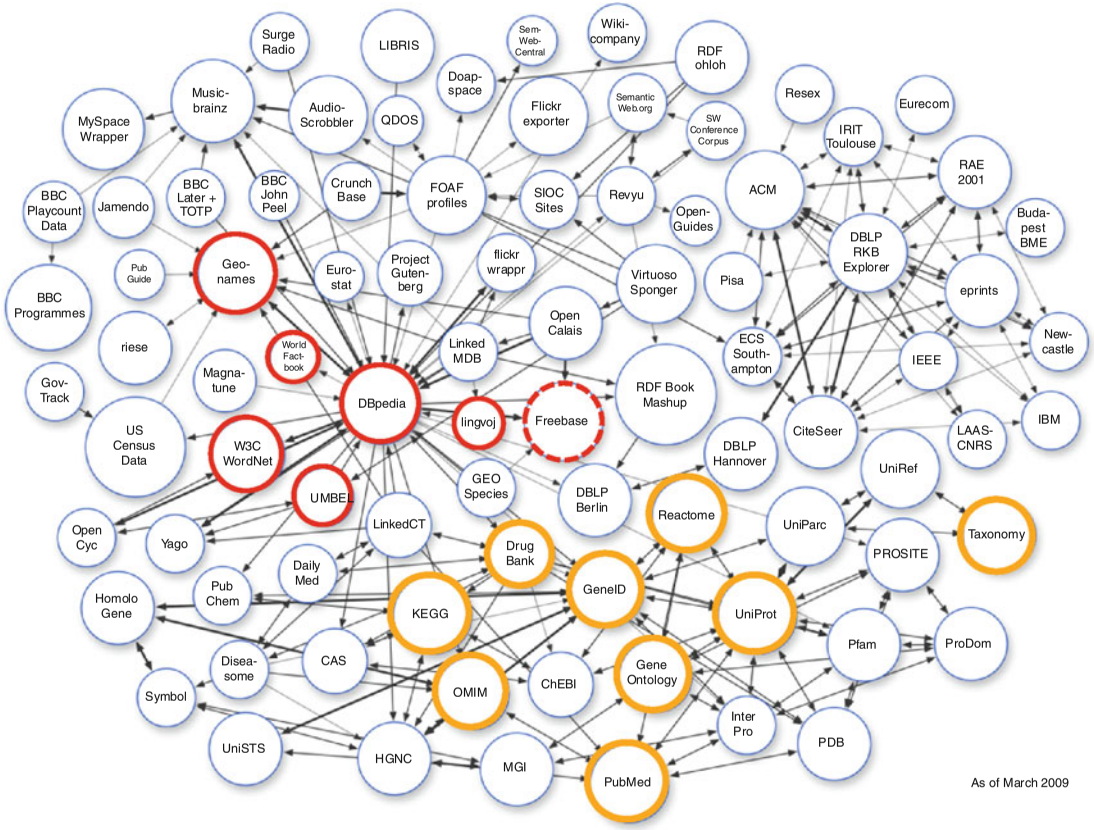

Fig. 7.10

Linking open data datasets included in reason-able views

The larger number of retrievable statements is a result of the `owl:sameAs` optimization discussed above. The optimization has "compressed" 7.8 billion statements, reducing the size of the indices five times. There are seven different "retrievable" statements against a single explicit statement asserted. The version of FactForge launched in May 2010 includes version 3.3 of DBPedia and a version of Geonames downloaded in April 2010.

FactForge is available as a free public service at http://factforge.net, offering the following access facilities:

- Incremental URI auto-suggest
- One-node-at-a-time exploration through Forest and Tabulator linked data browsers
- RDF Search: retrieve ranked list of URIs by keywords (see Sect. 7.6.4)
- SPARQL end-point

7.8.1.2 Linkedlifedata: 25 Biomedical Databases in a Box

Linked Life Data – Pathway and Interaction Knowledge Base (LLD – PIKB, http://linkedlifedata.com) is the largest known reason-able view of the Web of Data (4,179,999,703 triples). It assembles a large fraction of the life science–related datasets in LOD, and includes about 20 databases, as described in [41]. Linked Life Data is a data integration platform that realizes a massive RDF warehouse solution extended with inference and semantic annotations support. It integrates semantically, molecular information and realizes its linking to the public data cloud (LOD). The well-known data sources PubMed, UMLS, Entrez-Gene, and OBO Foundry have been transformed into RDF, using, in most of the cases, SKOS schema to represent the triples [29].

LLD – PIKB contains about 2,735,148,325 explicit triples, which are complemented by another 1,444,851,378 implicit statements inferred from them.

The data integration process is linking between related resources in the disconnected datasets. It takes place automatically according to predefined alignment patterns, mapping rules. They are presented in Sect. 7.8.1.2. For example, Namespace mapping identifies common parts of the URI and states that the same resource is referred to within two namespaces. Or Reference node identifies that a URI and a node with properties corresponding to parts of the URI refer to the same resource. Thus, LLD has an innovative way of handling pathways. It selects resources based on the metadata describing a resource, which interacts with the resource to be selected, whereas conventional approaches select elements based on the metadata describing them directly. Three types of mappings are supported: `exactMatch`, `closeMatch`, and related. `exactMatch` means that both entities have the same semantics, for example, they are both genes or proteins, and the mappings are made based on the existing stable unique identifiers. `closeMatch` means that both entities have the same semantics, for example, they are both gene or protein, but their mappings are made based on nonstable identifiers, for example, gene names. Related means that the two mapped entities can have different semantics, for example, one of them could be a gene, and one a protein.

A fourth special way of mapping individuals is through semantic annotation. Semantic annotation is used for assigning links between the entities, recognized by arbitrary information extraction algorithms, and their semantic descriptions. This process produces metadata providing class and/or instance information about the entities. The semantic annotation links identified entities in a given LLD dataset with their mentions in documents, for example, PubMed documents, and stores this information back into the semantic repository. Furthermore, the semantic annotation in LLD is capable of identifying and resolving all alternative names of a given element or concept to the same resource. The results demonstrate efficient search capabilities over highly heterogeneous and loosely coupled domain-specific data.

LLD - PIKB is used as a domain-specific reporting tool for generating new information insights. The system facilitates the mining of concealed relations among data by interlinking information from multiple heterogeneous sources and by providing a more holistic view over a particular scientific problem.

Linked data already gained popularity as a platform for data integration and analysis in the life science and health care domain. Reference [41] reports on recent developments in the Linked Life Data (LLD) platform and the Pathway and Interaction Knowledge Base (PIKB) dataset. The main objective set for the system is to facilitate the mining of concealed relations among data. Information is mined in 15 different data sources from five different biomedical domains. More than 20 completed data sources are interconnected, thus aiding in the understanding of research problems by linking unrelated data from heterogeneous knowledge domains. The collection of domain knowledge has been optimized by the use of instance alignment patterns that restore missing information relationships in the public linked data cloud (LOD). Additionally, information from unstructured texts has been matched with semantic annotations to the linked data instances from the knowledge base. This work addresses the reality that researchers in life sciences require different views over one and the same data. To understand the "bigger picture" of a research problem, the scientists need to link visually unrelated data from heterogeneous knowledge domains, while usually the analysis is limited based on the accessible overview of the data. Semantic Web technology has a place as a promising technology for reducing the complexity of combining data from multiple sources and resolving classical integration problems related to information accessibility. RDF technologies are applied as the "semantic glue" in these processes. Linked Life Data (LLD) is a data integration platform that realizes a massive RDF warehouse solution extended with inference and semantic annotations support. It implements the RDF representation of the PubMed, UMLS, Entrez-Gene, and OBO Foundry data sources, based on the SKOS scheme, and uses linked data principles. Integration patterns have been identified to interconnect related resources in RDF database representations. Semantic Annotation has been used for assigning links between the entities, recognized by arbitrary information extraction algorithm, and their semantic descriptions. This allows knowledge acquisition based on the extraction of more complex dependencies like the analysis of relationships between entities, event, and situation descriptions. The data in Pathway and Interaction knowledge Base (PIKB) has more than 2,217 billion statements, loaded in 40 h on

a standard server configuration with an average loading and inference speed varying between 5,000 and 60,000 statements per second, depending on the complexity of the loaded dataset. Continuous updates are maintained, and all post processing activities are automated. The LLD platform demonstrates efficient search over highly heterogeneous and loosely coupled data. It is capable of executing queries that cover information from seven different sources in a timely fashion. The platform and the PIKB dataset are used as domain-specific reporting tools for generating new information insights. The Web front-end provides three paths to access the data: a Web form for issuing SPARQL queries, a browser for exploring resources, and full-text search in the graph containing the searched literal with matched resources.

7.8.2 Publishing of Content on the Web, Based on Semantic Metadata

The BigOWLIM semantic repository (see Sect. 7.6.4) was successfully integrated into the high-performance Semantic Web publishing stack powering the BBC's 2010 World Cup website. This use case is presented as an example of an application of semantic repository technology in the publishing industry, which is remarkable in two ways: It provides evidence for the advantages of the technology compared to relational databases, and it proves that such engines are already mature enough to handle high loads in critical applications.

The following information was stored in the repository backing BBC's World Cup website:

- Ontologies, both domain-specific ones (about sport and particularly football) and general (e.g., the FOAF schema)
- Factual knowledge, for example, information about specific teams, players, games, etc.
- Metadata about describing the content produced and published by the BBC by means of references to the ontologies and the entities described in the factual knowledge part

The metadata part was updated constantly to reflect at real time the stream of new content (articles and other media assets) relevant to the World Cup, which was the subject of publishing at the BBC's website. As described below, the main function of the repository was to provide selections of artifacts relevant to specific concepts – these selections were used to dynamically generate Web pages on the subject. Reasoning helped the matching between the content metadata and the subjects to take into consideration the semantics of all the data.

BigOWLIM was set up to perform materialization against a customized variant of the OWL Horst rule-set. As the updates of the repository required deletions and needed to happen in real time, the "smooth invalidation" feature of BigOWLIM (see Sect. 7.2.1.6) was critical for the performance of the overall solution. Further, the Replication Cluster feature (see Sects. 7.4.2 and 7.6.4) of BigOWLIM enabled horizontal scaling with respect to the query loads and failover.

Due to confidentiality constraints, the use case will be described through citations of a couple of blog posts from the technical team at BBC that provide an insight into the

business case for the deployment of semantic technologies in their World Cup website, the technical architecture of the publishing stack, the strategic importance of the project's success, and the plans for the further usage of semantic technology and linked data within the BBC.

In [45], John O'Donovan, *Chief Technical Architect, Journalism and Knowledge, BBC Future Media & Technology*, discusses the business benefits of the implemented semantic solution:

- *"The World Cup site is our first major statement on how we think this (the Semantic Web) can work for mass market media and a showcase for the benefits it brings. . . . Though we have been using RDF and linked data on some other sites (. . .) we believe this is the first large scale, mass media site to be using concept extraction, RDF and a Triple store to deliver content."*

  ". . . we are not publishing pages, but publishing content as assets which are then organized by the metadata dynamically into pages, but could be re-organized into any format we want much more easily than we could before. . . . There is also a change in editorial workflow for creating content and managing the site. This changes from publishing stories and index pages, to one where you publish content and check the suggested tags are correct. The index pages are published automatically. This process is what assures us of the highest quality output, but still saves large amounts of time in managing the site and makes it possible for us to efficiently run so many pages for the World Cup."

  "As more content has Linked Data principles applied to it . . . the vision of a Semantic Web moves closer. Importantly, what we have been able to show with the World Cup, is that the technology behind this is ready to deliver large scale products."

  "This is more than just a technical exercise – we have delivered real benefits back to the business as well as establishing a future model for more dynamic publishing which we think will allow us to make best use of our content and also use Linked Data to more accurately share this content and link out to other sites and content*, a key goal for the BBC. We look forward to seeing the use of* Linked Data *grow as we move towards a more Semantic Web."*

In a following post [51], Jem Rayfield, Senior Technical Architect, BBC News and Knowledge, provides more information on the technical architecture of the high-performance publishing stack and the related data flows and data modeling:

- *"The* World Cup 2010 website *is a significant step change in the way that content is published. . . . As you navigate through the site it becomes apparent that this is a far deeper and richer use of content than can be achieved through traditional CMS-driven publishing solutions."*

  "The site features 700-plus team, group and player pages, which are powered by a high-performance dynamic semantic publishing framework. This framework facilitates the publication of automated metadata-driven web pages that are light-touch, requiring minimal journalistic management, as they automatically aggregate and render links to relevant stories."

  "The foundation of these dynamic aggregations is a rich ontological domain model. The ontology describes entity existence, groups and relationships between the things/concepts that describe the World Cup. For example, "Frank Lampard" is part of the "England Squad" and the

"England Squad" competes in "Group C" of the "FIFA World Cup 2010". The ontology also describes journalist-authored assets (stories, blogs, profiles, images, video and statistics) and enables them to be associated to concepts within the domain model . . ."

"A RDF triplestore *(ref. BigOWLIM) and* SPARQL *approach was chosen over and above traditional relational database technologies due to the requirements for interpretation of metadata with respect to an ontological domain model. The high level goal is that the domain ontology allows for intelligent mapping of journalist assets to concepts and queries. The chosen triple store provides reasoning following the forward-chaining model and thus implied inferred statements are automatically derived from the explicitly applied journalist metadata concepts."*

"This inference capability makes both the journalist tagging and the triple store powered SPARQL queries simpler and indeed quicker than a traditional SQL approach. Dynamic aggregations based on inferred statements increase the quality and breadth of content across the site. The RDF triple approach also facilitates agile modeling, whereas traditional relational schema modeling is less flexible and also increases query complexity."

"Our triple store is deployed multi-data center in a resilient, clustered, performant and horizontally scalable fashion, allowing future expansion for additional ontologies and indeed linked open data (LOD) sets. . . . The triple store is abstracted via a JAVA/Spring/CXF JSR 311 compliant REST service. . . . The API is designed as a generic facade onto the triple store allowing RDF data to be re-purposed and re-used pan BBC. This service orchestrates SPARQL queries and ensures that results are dynamically cached with a low 'time-to-live' (TTL) (1 minute) expiry cross data center using memcached."

"This dynamic semantic publishing architecture has been serving millions of page requests a day throughout the World Cup with continually changing OWL reasoned semantic RDF data. The platform currently serves an average of a million SPARQL queries a day with a peak RDF transaction rate of 100s of player statistics per minute. . . ."

"The development of this new high-performance dynamic semantic publishing stack is a great innovation for the BBC as we are the first to use this technology on such a high-profile site. It also puts us at the cutting edge of development for the next phase of the Internet, Web 3.0."

7.9 Related Resources

This section provides references to publications related to semantic repositories, their functionality, design, and performance.

Kiryakov [32] aims to define criteria for the validation of the performance of semantic repositories and in particular for the data layer of the LarKC platform for web-scale reasoning. It introduces the first version of the *conceptual framework for semantic repository tasks and performance aspects* (presented in Sect. 7.3 here) and provides an overview of the state-of-the-art repositories. Finally, he defines *performance and scalability targets* for the development of the semantic repositories over a period of 3 years.

In [15], the developers of the Virtuoso engine (see Sect. 7.6.8) argue that web-scale RDF management requires manipulations like joining, aggregation, and the filtering of data together with inference and on-the-fly schema mapping. Further, they present some of the research and experiments on the usage of various indexing techniques and distribution schemas. The paper motivates the usage of *bitmap indices* with key compression (to improve space efficiency) and *data sampling* at query optimization time by keeping the data in memory (to address the issue of the speed of processing). The *data partitioning* schema used in Virtuoso is presented along with the basic IRI encoding and indexing mechanisms. Useful observations and statistics about the *actual computational cost* of some atomic operations and the network traffic are also presented. [16] provides more recent insights of Orri Earling on the *trends* of development of the semantic repositories and argues that in order for the latter to become a widely adopted data management technology, they should match the efficiency of the relational DBMS in dealing with regular data. As an approach to achieve such efficiency, it proposes *column-based compression* techniques and reports evaluation results of an early implementation of such technique.

As discussed in Sect. 7.4.2, approaches based on distribution via data partitioning have some intrinsic limitations when it comes to full-cycle data management (see Sect. 7.3.4). Still, in specific scenarios and for specific tasks, distribution can deliver amazing results. In the scope of the LarKC project, a group at the VU Amsterdam developed the WebPIE (The distributed materialization approach implemented in WebPIE is presented in greater detail in KR and Reasoning on the Semantic Web: Web-scale Reasoning) system, which implements a MapReduce (MapReduce is a framework for the parallel and distributed processing of batch jobs on a large number of compute nodes.) based *distributed materialization*. As reported in [61], WebPIE demonstrates extremely scalable reasoning with respect to the semantics of RDFS and OWL Horst. The success of WebPIE is based on several optimizations, related to the specificity of the entailment rules, which defined the semantics of the above-mentioned languages, and assumptions about the specific data loading discipline. These optimizations allow WebPIE to decrease the number of inference jobs to be performed by the cluster, and thus the time required for closure computation. Given real-world datasets like UNIPROT and FactForge (see Sects. 7.5.2 and 7.5.3) of size close to 1 billion statements, it delivers OWL Horst reasoning speeds in the range of 70,000 explicit statements per second on a 64-node cluster. The experiments with the synthetic datasets of the LUBM benchmark (see Sect. 7.5.1) demonstrate speeds in the range of 500,000 statements per second for a dataset of 100 billion explicit statements. As already discussed in Sect. 7.5.1, this proves that LUBM datasets are relatively easy to reason with; still, at present, this experiment demonstrates the highest speed and scalability officially reported.

WebPIE builds on top of the experience with MARVIN [48], a system, which implements *incomplete distributed materialization*. MARVIN is based on the approach of divide-conquer-swap, for example, peers autonomously partition the problem in some manner, each operate on some subproblem to find partial solutions, and then repartition their part and swap it with another peer; all peers keep repartitioning, solving, and swapping to

find all solutions. MARVIN guarantees eventual completeness of the inference process and produces its results gradually. Initial partitioning is accomplished by reading some random data from disk, and subsequent partitioning is dictated by step swap. The conquer phase is performed by an off-the-shelf reasoner which computes the closure of the portion of the data available at a specific step in a single node. The problem of making distributed reasoning scalable and load-balanced is addressed with the SpeedDate approach that is also used in newer developments around WebPIE, [35], which count on the so-called elastic clustering to deal with *skewed term frequency distributions*.

In [28] Adam Jacobs examines some facets of data organization and processing, relating to the design of the data structure for efficient querying and analysis, and not just for storing. He argues for adopting *sequential access to data*, because what makes the big data big is the repeated observations over time and space. In other words, large datasets have *inherent temporal or spatial dimensions*, and in order to achieve acceptable performance for highly order-dependent queries on truly large data, one should turn to a data representation model that recognizes the concept of inherent ordering of data down to the implementation level. Further, the paper presents a discussion on several issues related to the interplay between the constraints and the capabilities of the current hardware infrastructure and few of the most popular approaches for high-performance data management, for example, "everything in memory" and distributed computation.

The YARS repository was the first one to demonstrate an efficient implementation of *large-scale RDF management via data partitioning*. The system was proven in scale-up experiments on seven billion synthetically generated statements over 16 Index Managers on 16 machines with 100 queries with a randomly chosen resource joined with one or two quad patterns. The basic design principles behind YARS2, along with algorithms and evaluation data, are presented in [23]. While YARS is not provided at present as a stand-alone product, the results and the principles discussed in this paper are used in several of the prominent engines discussed in Sect. 7.6. Further, YARS was developed as part of SWSE – an end-to-end *Semantic Web search engine* that uses a graph data model to enable interactive query answering over structured and interlinked data collected from many disparate sources on the Web. In many aspects, SWSE is similar to the FactForge linked data service, presented in Sect. 7.8.1.1. The major difference is that the RDF data indexed in SWSE are crawled from the Web, while FactForge is loaded with specific datasets, which are preprocessed and cleaned up in order to provide some level of guarantee about the consistency of the data and inferences on top of it; it is also the case that FactForge supports SPARQL queries, while SWSE only supports keyword search. What is common between the two systems is that in both projects *ranking and information retrieval* (More information about search and retrieval methods appropriate for the retrieval of relevant information from large RDF dataset can be found in Semantic Technology Adoption: A Business Perspective: The Cases of Swoogle and Watson, which presents two other semantic Web search projects. Sindice (http://sindice.com/) is another Semantic Web search engine. The similarity between Sindice and SWSE is that they both offer partial support for structured queries: Users are allowed to make one-step attribute-value constraints, which allow efficient implementation without expensive DBMS-alike

join operations.) methods for semi-structured RDF data represent a key feature. A detailed and up-to-date description of SWSE can be found in [26], while [24] presents a brief overview of the key points and the recent advances in both SWSE and YARS.

This 2004 paper [22] presents evaluation work of knowledge-based systems in large OWL applications to help in the selection of the most appropriate system for a given task. It provides the rationale for and describes the LUBM (Lehigh University Benchmark) – today's most popular *semantic repository benchmark*, discussed in Sect. 7.5.1. This is first-of-a-kind experiment with respect to the scale of data it was designed for, the open and comprehensive design, and the clear documentation of the benchmark framework, which allowed many repository vendors and users to adopt it for the testing of their tools and environments.

Over the last couple of years, semantic technologies have started getting real attention from the *business* (The adoption of semantic technologies by the industry is discussed in Semantic Technology Adoption: A Business Perspective). As discussed in Sect. 7.1.1, RDF-based semantic repositories are seen as an alternative technology, which can replace relational DBMS in many environments in order to improve the efficiency of data integration and heterogeneous data management. RDF-based data management is being promoted by respected *mainstream analysts and consultants* such as PriceWaterhouseCoopers [49], and Gartner [63]. According to [49], among the most critical business problems today are the information gaps, especially in the areas of customer needs and business risk. What is actually missing is more context that explains what the data mean. The Semantic Web directly engages with the meaning and context of a business – its semantics, and the linked data approach ensures access to comprehensive data and a greater sharing of internal data. Data federation for web-scale many-to-many sharing with easier connections to more sources, combined with the ability to be reused by others, is the preferred data model for accomplishing this. It is argued that business data integration must be rethought as data linking, a decentralized, federated approach that uses ontology-mediated links to have those data at their sources. The goal is to create an information mediation layer that lets business staff explore what-if scenarios, assess strategies and risks, and gain insight from the messy reality of the world inside and outside the company. The linked data approach is contrasted with the data warehouse approach, which is deemed as nonflexible and outdated, unable to meet the actual business needs.

According to [63], Master Data Management (MDM) is one possible path to bridge the gap of the adoption of semantic technologies in the enterprise. It is considered as a semantic-oriented discipline focusing on sustaining a "single view" of critical enterprise information (referred as "master data"), which seems like an argument for mapping semantics across different systems and data stores. The authors believe that the link between semantics, Semantic Web, and MDM will increase in the future.

7.10 Future Issues

Semantic repositories are database management systems, based on RDF as data representations model. They combine important features of several other types of tools: reasoning

capabilities, like those of the inference engines; capabilities to handle sparse data and evolving data schemas like those of the column-stores; the robustness of the relational DBMS. After 10 years of development, semantic repositories now start seeing adoption in real-world applications, which can be explained by two reasons: The tools passed some threshold of maturity, and the market finally started understanding and appreciating their unique value proposition.

Facing real usage and actual end-user requirements from wide range of applications generates new requirements for semantic repositories. Here follows a list of directions for future development:

- Web-scale and –style incomplete reasoning, similar to those developed in the LarKC project
- Content-based retrieval modalities, like the RDF Search employed in FactForge (Sect. 7.8.1.1)
- Extensible architectures, which allow for efficiently handling specific, filtering and lookup criteria, for instance, geo-spatial constraints, full-text search, and social network analysis.

7.11 Cross-References

- KR and Reasoning on the Semantic Web: RIF
- KR and Reasoning on the Semantic Web: OWL
- KR and Reasoning on the Semantic Web: Web-scale Reasoning
- Semantic Annotation and Retrieval: Web of Data
- Semantic Technology Adoption: A Business Perspective
- Semantic Web Search Engines

References

1. Aasman, J.: AllegroGraph 4.0 – industry's first real time RDF store. Presentation at Semantic Technologies Conference (SemTech 2009), San Jose (2009)
2. Andersson, B., Momtchev, V.: LarKC requirements summary and data repository. LarKC project deliverable D7a.1.1. http://www.larkc.eu/wp-content/uploads/2008/01/larkc_d7a-11_requirements-summary-and-data-repository_m6.pdf (2008)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Scheider, P.: The Description Logic Handbook. Theory, Implementation, Applications. Cambridge University Press, Cambridge (2003)
4. Berners-Lee, T., Fielding R., Masinter L.: Uniform Resource Identifier (URI): generic syntax. Network Working Group, Request for comments: 3986. http://tools.ietf.org/html/rfc3986 (2005). Accessed Jan 2005
5. Berners-Lee, T.: Design issues: linked data. http://www.w3.org/DesignIssues/LinkedData.html (2006)
6. Bizer, Ch., Schultz, A.: Benchmarking the performance of storage systems that expose SPARQL endpoints. In: Proceedings of the Fourth International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS 2008), Karlsruhe (2008)

7. Bizer, Ch., Schultz, A.: Berlin SPARQL benchmark results. Document version 1.2. http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/index.html (2009). Accessed 23 Mar 2009
8. Bizer, Ch., Schultz, A.: BSBM results for Virtuoso, Jena TDB, BigOWLIM. http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V5/index.html (2009). Accessed 30 Nov 2009
9. Brewer, E.: Towards robust distributed systems. Keynote at Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC 2000), Portland. http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf (2000)
10. Brickley, D., Guha, R.V. (eds.): Resource Description Framework (RDF) schemas. W3C Recommendation. http://www.w3.org/TR/rdf-schema/ (2004). Accessed 10 Feb 2004
11. Carroll, J.J., Bizer, B., Hayes, P., Stickler, P.: Named graphs, provenance and trust. In: Proceedings of the 14th International conference on World Wide Web (WWW 2005), Chiba. http://www2005.org/cdrom/docs/p613.pdf (2005)
12. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. In: Proceedings of the Seventh Symposium on Operating Systems Design and Implementation (OSDI 2006), Seattle. http://labs.google.com/papers/bigtable.html (2006)
13. Bechofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: In: Dean, M., Schreiber, G. (eds.), OWL web ontology language reference, W3C Recommendation. http://www.w3.org/TR/owl-ref/ (Feb 2004)
14. Erling, O.: LUBM and Virtuoso. OpenLink software product blog. http://www.openlinksw.com/dataspace/oerling/weblog/Orri%20Erling's%20Blog/1562 (2009). Accessed 24 July 2009
15. Erling, O., Mikhailov, I.: Towards web-scale RDF. http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSArticleWebScaleRDF (2009)
16. Erling, O.: Directions and challenges for Semdata. In: Proceedings of the Semantic Data Management (SemData 2010) Workshop at the 36th International Conference on Very Large Data Bases (VLDB 2010), Singapore (2010)
17. Fischer, F., Keller, U., Kiryakov, A., Huang, Z., Momtchev, V., Simperl, E.: Initial knowledge representation formalism. LarKC project deliverable D1.1.3. http://www.larkc.eu/wp-content/uploads/2008/01/larkc_d113-initial-knowledge-representation-formalism_m7.pdf (2008)
18. Franz Inc.: AllegroGraph RDFStore 4.0. AllegroGraph product information. http://www.franz.com/agraph/allegrograph/ (2010). Accessed 22 Aug 2010
19. Franz Inc.: AllegroGraph RDFStore version 4.0 LUBM benchmark results. http://www.franz.com/agraph/allegrograph/agraph_bench_lubm.lhtml (2010). Accessed 22 Aug 2010
20. Franz Inc.: RDFS++ dynamic materialization. http://www.franz.com/agraph/allegrograph/dynamic-materialization.lhtml (2010). Accessed 22 Aug 2010
21. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Proceedings of the 12th Internationl Conference on world Wide Web (WWW 2003), Budapest (2003)
22. Guo, Y., Pan, Z., Heflin, J.: An evaluation of knowledge base systems for large OWL datasets. J. Web Semant. **3**(2), 158–182 (2004)
23. Harris, S., Lamb, N., Shadbolt, N.: 4store: the design and implementation of a clustered RDF store. In: Proceedings of the Fifth International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2009) at the Eighth International Semantic Web Conference (ISWC 2009), Chantilly. Lecture Notes in Computer Science, vol. 5823. Springer, Berlin (2009)
24. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: a federated repository for querying graph structured data from the web. In: Proceedings of the Sixth International Semantic Web Conference (ISWC 2007), Busan. Lecture Notes in Computer Science, vol. 4825, pp. 211–224. Springer, Berlin (2007)
25. Harth, A: SWSE/YARS@SemData Sofia 2010. SemData roundtable, Sofia. http://semdata.org/sites/default/files/harth-swse-sofia-2010.pdf (2010). Accessed 12 Mar 2010
26. Hayes, P.: RDF semantics, W3C Recommendation. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ (Feb 2004)
27. Hogan, A., Harth, A., Umbrich, J., Kinsella, S., Polleres, A., Decker, S.: Searching and browsing

linked data with SWSE: the semantic web search engine. DERI technical report 2010-07-23. http://www.deri.ie/fileadmin/documents/DERI-TR-2010-07-23.pdf (2010)
28. Horrocks, I., Patel-Schneider, P.F., Bechhofer, S., Tsarkov, D.: OWL rules: a proposal and prototype implementation. J. Web Semant. **3**, 23–40 (2005)
29. Jacobs, A.: The pathologies of big data. Commun. ACM **52**(8), 36–44 (2009)
30. Jupp, S., Bechhofer, S., Kostkova, P., Stevens, R., Yesilada, Y.: Document navigation: ontologies or knowledge organisation systems? In: Proceedings of the Seventh International Workshop on Network Tools and Applications in Biology (NETTAB 2007), Pisa. A Semantic Web for Bioinformatics: Goals, Tools, Systems, Applications (2007)
31. Kerrigan, M., Bradesko, L., Fortuna B.: Rapid prototype of the LarKC. LarKC project deliverable D5.2.1. http://www.larkc.eu/wp-content/uploads/2008/01/larkc_d521_rapid-prototype-of-the-larkc.pdf (2009)
32. Kiryakov, A.: Measurable targets for scalable reasoning. LarKC project deliverable D5.5.1, formerly titled "Definition of validation goals for the prototyping phase". http://www.larkc.eu/wp-content/uploads/2008/07/larkc_d551.pdf (2008)
33. Kiryakov, A., Tashev, Z., Ognyanoff, D., Velkov, R., Momtchev, V., Balev, B., Peikov, I.: Validation goals and metrics for the LarKC platform. LarKC project deliverable D5.5.2. http://www.larkc.eu/deliverables/ (2009)
34. Kiryakov, A., Ognyanoff, D., Velkov, R., Tashev, Z., Peikov, I.: LDSR: materialized reasonable view to the web of linked data. In: Proceedings of the Third International Symposium on Rules, Applications and Interoperability (RuleML 2009), Las Vegas. Lecture Notes in Computer Science, vol. 5858. Springer, Berlin (2009)
35. Kiryakov, A., Momtchev, V.: Triplesets: tagging and grouping in RDF datasets. W3C Workshop "RDF Next Steps", Stanford (June 2010)
36. Kiryakov, A., Bishop, B., Ognyanoff, D., Peikov, D., Tashev, Z., Velkov, R.: The features of BigOWLIM that enabled the BBC's World Cup website. In: Proceedings of the Semantic Data Management (SemData 2010) Workshop at the 36th International Conference on Very Large Data Bases (VLDB 2010), Singapore (2010)
37. Kotoulas, S., Oren, E., Van Harmelen, F.: Mind the data skew: distributed inferencing by speeddating in elastic regions. In: Proceedings of the 19th International Conference on World Wide Web (WWW 2010), Raleigh (2010)
38. Heath. T.: Linked data – connect distributed data on the web. http://linkeddata.org/ (2007)
39. World Wide Web Consortium (W3C): Linking open data. W3C SWEO community project. http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/ (2007)
40. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y.: Towards a complete OWL ontology benchmark. In: Proceedings of the Third European Semantic Web Conference (ESWC 2006), Budva. Lecture Notes in Computer Science, vol. 4011, pp. 125–139. Springer, Berlin (2006)
41. Manola, F., Miller, E. (eds.): RDF primer, W3C Recommendation. http://www.w3.org/TR/rdf-primer/ (Feb 2004)
42. McGuinness, D.L., van Harmelen, F. (eds.): OWL web ontology language. Overview. W3C Recommendation. http://www.w3.org/TR/owl-features/ (2004)
43. Momtchev, V., Kiryakov, A.: Second generation ontology representation and data integration (ORDI) framework, specification. Ontotext technical documentation. http://www.ontotext.com/ordi/ORDI_SG/ORDI_SG_ Specification.pdf (2006). Accessed 13 Oct 2006
44. Momtchev, V., Peychev, D., Primov, T., Georgiev, G.: Expanding the pathway and interaction knowledge in linked life data. In: Proceedings of the International Semantic Web Challenge (ISWC 2009), Chantilly. Lecture Notes in Computer Science, vol. 5823. Springer, Berlin. http://challenge.semanticweb.org/ (2009)
45. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 web ontology language profiles, W3C Candidate Recommendation. http://www.w3.org/TR/owl2-profiles/ (Feb 2009)
46. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. J. Web Semant. **3**(1), 41–60 (2005)
47. North, K.: The NoSQL alternative. Dr. Dobb's J. http://www.drdobbs.com/database/224900500. Accessed 21 May 2010

48. O'Donovan, J.: The World Cup and a call to action around linked data. BBC blog post. http://www.bbc.co.uk/blogs/bbcinternet/2010/07/the_world_cup_and_a_call_to_ac.html. Accessed 9 July 2010
49. ONTOCORE: Ontology logic and reasoning at semantic Karlsruhe. Home page, http://logic.aifb.uni-karlsruhe.de/
50. Ontotext Lab: RDF(S), rules and OWL dialects. http://www.ontotext.com/inference/rdfs_rules_owl.html. Accessed Dec 2009 (2009)
51. Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R., Ten Teije, A., Van Harmelen, F.: MARVIN: distributed reasoning over large-scale semantic web data. J. Web Semant. **7**(4), 305–316 (2009)
52. PricewaterhouseCoopers: Spring of the data web. Technology Forecast. A Quart. J. Spring 2009. http://www.pwc.com/techforecast/ (2009)
53. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF, W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/ (Jan 2008)
54. Rayfield, J.: BBC World Cup 2010 dynamic semantic publishing, BBC blog post. http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_cup_2010_dynamic_sem.html (2010)
55. Reed, D.P.: Naming and synchronization in a decentralized computer system. MIT dissertation, http://portal.acm.org/citation.cfm?id=889815 (1978)
56. Ruhloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triple-store technologies for large data stores. In: Proceedings of the Scalable Semantic Systems Workshop (SSSW 2007), Portugal (2007)
57. Salvadores, M., Correndo, G., Omitola, T., Gibbins, N., Harris, S., Shadbolt, N.: 4s-reasoner: RDFS backward chained reasoning support in 4store. In: Proceedings of the 2010 International Workshop on Web-Scale Knowledge Representation, Retrieval, and Reasoning (Web-KR3 2010), Toronto (2010)
58. Stoilos G., Grau B.C., Horrocks I.: How incomplete is your semantic web reasoner? In: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 10), Atlanta (2010)
59. Swiss Institute of Bioinformatics. UniProt RDF, http://dev.isb-sib.ch/projects/uniprot-rdf/ (2009)
60. SYSTAP LLC: Bigdata: approaching web scale for the semantic web. http://www.bigdata.com/whitepapers/bigdata_whitepaper_07-08-2009.pdf (2009)
61. SYSTAP LLC: Bigdata overview. SemData roundtable, Sofia. http://semdata.org/sites/default/files/bigdata-sofia-roundtable-3-10-2010.pdf (2010). Accessed 12 Mar 2010
62. ter Horst, H. J.: Combining RDF and part of OWL with rules: semantics, decidability, complexity. In: Proceedings of the Fourth International Semantic Web Conference (ISWC 2005), Galway. Lecture Notes in Computer Science, vol. 3729, pp. 668–684. Springer, Heidelberg (2005)
63. Thakker, D., Osman, T., Gohil, S., Lakin, P.: A pragmatic approach to semantic repositories benchmarking. In: Proceedings of the Seventh Extended Semantic Web Conference (ESWC 2010), Heraklion (2010)
64. Todorova, P., Kiryakov, A., Ognyanoff, D., Peikov, I., Velkov, R., Tashev, Z.: Spreading activation components. LarKC project deliverable D2.4.1 (2009)
65. Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., Bal, H.: OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In: Proceedings of the Seventh Extended Semantic Web Conference (ESWC 2010), Heraklion. Lecture Notes in Computer Science, vol. 6088, pp. 213–227. Springer, Berlin (2010)
66. Vertica: The vertica database datasheet. Product overview. http://www.vertica.com/_pdf/VerticaDatabaseDataSheet.pdf (2010). Accessed Jan 2010
67. White, A. (Gartner): Semantic web moving ever close to the 'Semantic Enterprise?' http://blogs.gartner.com/andrew_white/2009/04/30/semantic-web-moving-ever-close-to-the-semantic-enterprise/ (2009)
68. World Wide Web Consortium (W3C): Linking open data. W3C SWEO community project home page. http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData (2010). Accessed Jan 2010
69. Wu, A., Wu, Z., Kolovski, V.: An enterprise inference engine inside Oracle Database 11g Release 2. Presentation at Semantic Technology Conference (SemTech 2010), San Francisco (2010)