OWLIM: A family of scalable semantic repositories

Barry Bishop[a], Atanas Kiryakov[a], Damyan Ognyanoff[a], Ivan Peikov[a], Zdravko Tashev[a], Ruslan Velkov[a]
[a]*Ontotext AD, 135 Tsarigradsko Chaussee, Sofia 1784, Bulgaria*

Abstract. An explosion in the use of RDF, first as an annotation language and later as a data representation language, has driven the requirements for Web-scale server systems that can store and process huge quantities of data, and furthermore provide powerful data access and mining functionalities. This paper describes OWLIM, a family of semantic repositories that provide storage, inference and novel data-access features delivered in a scalable, resilient, industrial-strength platform.

Keywords: Cluster, database, inference, OWL, priming, ranking, RDF, RDFS, reasoner, rules, semantic repository, semantic-Web, SPARQL, text-search, triple-store

1. Introduction

This report gives an overview of the OWLIM [3] family of semantic repositories that are widely used in both commercial and research environments.

There is no formal definition of the term 'semantic repository' so for the purposes of this article we use this term for Database Management Systems (DBMS) that can be used to store, query and manage data structured according to the Resource Description Framework [1] (RDF) standard(s). Compared to Relational Database Management Systems (RDBMS), such systems use flexible ontological schemata where data is processed by an inference-engine according to a well-defined semantics.

Section 2 gives a brief overview of the emerging RDF landscape and outlines some desirable properties of systems that store and process RDF data. Section 3 introduces the OWLIM family of semantic repositories and describes many of its features that make it a world-leading RDF storage, reasoning and query-answering platform. Section 4 covers some advanced features of BigOWLIM that go beyond the RDF/RDFS [5]/OWL [4] language stack and show how other AI and text-processing related functions are combined to enable novel, but powerful data-mining applications. A summary of the current state of the OWLIM family is given in section 4 together with some comments regarding recent independent evaluations that have been carried out. Section 5 concludes with a summary of the information presented and some indications for the future evolution of the OWLIM family.

2. Background

The Resource Description Framework (RDF) was originally designed as a metadata data model for the purpose of annotating Web resources (pages) with machine-processable information leading to the concept of the semantic Web [6]. Due in part to its simple and flexible data model, it has become widely used for general purpose knowledge management and modeling, the most notable being "Linked Data" [2] a concept outlined by Tim Berners-Lee [13]. The principle idea behind Linked Data is that RDF graphs are published on the Web and can be navigated in just the same way that a Web browser is used to browse the current HTML Web. In order for this to function, publishers should adhere to a number of principles involving the use of URIs to identify concepts, the ability to use URIs to get information about concepts and provide links to other RDF graphs.

Linking Open Data (LOD) is a W3C Semantic Web Education and Outreach community project aiming to extend the Web by publishing open datasets as RDF and by creating RDF links between concepts from different data sources. One of the

central datasets of LOD is DBPedia [21] – an RDF extract of the Wikipedia open encyclopedia – which serves as a 'hub' in the LOD graph, because of the many mappings between it and the other LOD datasets. Currently LOD includes more than 40 datasets, containing some 13 billion statements, joined together with many millions of link statements.

While the principles of Linked Data allow for an open and decentralized Web of data, there are intrinsic problems to do with the consumption of such data. While it is technically possible to execute queries spanning multiple datasets published via a number of separate servers, in reality the distributed nature of the data prevents the evaluation of queries with multiple joins happening in reasonable time. To compound the problem, the inference required to properly answer queries according to the intended semantics adds additional computational overhead.

From this environment we see the emerging requirements for software components that can manage the volume of data available and provide mechanisms for the consumption of this data. We call these software components 'Semantic Repositories' and they must be able to store huge volumes of RDF data, perform the necessary inference according to the semantics of the data and provide a powerful query-answering mechanism that operates in real-time.

The following section introduces the OWLIM family of semantic repositories and describes in detail the qualities that make them desirable tools for exploiting the Web of data.

3. OWLIM semantic repository (RDF database)

OWLIM is a family of semantic repository solutions, that each boast a pure Java, native RDF database implementation. Currently there are two variants of OWLIM optimized for different operating environments: SwiftOWLIM is an in-memory, full-featured RDF database, inference-engine and query-answering engine. It uses highly optimized indexes and data structures to be able to process tens of millions of RDF statements on standard desktop hardware. Partly due to its in-memory nature, it is the world's fastest semantic repository being able to load data at over 50,000 statements per second on a 1000 USD machine using with inference. SwiftOWLIM is essentially open-source, but is based upon the free-for-use, but not open source Triple Reasoning and Rule Entailment Engine [16] (TRREE). BigOWLIM is the commercial version published under an RDBMS-style license and is positioned as an enterprise-grade database management system that can handle tens of billions of statements. BigOWLIM uses a number of storage and query optimizations that allow it to sustain outstanding performance even when managing tens of billions of statements. On top of this, BigOWLIM incorporates a number of advanced features and alternative data access methods that seamlessly integrate with standard query answering to provide a powerful, hybrid data mining platform. In the following sections, 'OWLIM' will be used when describing qualities common to both engines.

Both variants of OWLIM are full-featured semantic repositories each packaged as a Storage and Inference Layer (SAIL) for the Sesame openRDF [9] framework.

OWLIM is compatible with all the common RDF syntaxes (XML, N3, N-Triples, Turtle, TRIG, TRIX) and supports the SPARQL [7] and SeRQL query languages. Furthermore, the Sesame HTTP components and Web applications allow OWLIM to be used as a server database system with comprehensive administration utilities.

The rest of this section describes various features of OWLIM related to the management of RDF data.

3.1. Inference Engine

The inferencing strategy in OWLIM is one of total materialization (apart from the owl:sameAs optimization discussed in section 3.3) based on R-Entailment (as defined by ter Horst [10]) where Datalog [11] like rules with inequality constraints operate directly on a single ternary relation that represents all triples. In addition, free variables in rule heads are treated as blank nodes (a feature that must be used with caution in order to avoid an infinite recursive expansion).

Total materialization involves computing all the entailed statements at load time. While this introduces additional reasoning cost when loading statements in to a repository, the desirable consequence is that query evaluation can proceed extremely quickly.

Several standard rule sets are included in all editions of OWLIM and these include (in more or less increasing levels of complexity): 'empty' (no inference), OWL-Horst [8], RDFS, owl-max (RDFS plus most of OWL-Lite) and OWL2-RL [15]. The

desired semantics are chosen by selecting the rule set when a repository is first created.

OWL2-RL has the highest complexity due to the sheer number of rules and the use of RDF lists. Rules **prp-spo2** and **prp-key** have proved to be the most troublesome, because they do not have a straightforward mapping to R-Entailment. Instead a set of recursive rules are used to 'iterate' property chains and lists of keys. In fact, **prp-key** has such poor performance with the current implementation of the rule language, that two versions of the OWL2-RL rule set are provided, one with this rule (conformant) and one without (reduced). Users not making use of the OWL HasKey constructor will benefit greatly from using the reduced rule-set.

In addition to the standard semantics, user-defined rule-sets can be used. In this case the user provides the full pathname to a custom rule file that contains definitions of axiomatic triples, rules and consistency checks. For ease of use, the rule files for the standard included semantics are provided and users can modify or extend these for their specific purposes.

3.2. Consistency checks

Consistency checks are used to ensure that the data model is in a consistent state and are applied whenever an update transaction is committed. The syntax is similar to that of rules, except that the consequences are optional.

Consistency checks that have no consequences will indicate a consistency violation whenever their premises are satisfied. This syntax is suitable for such activities as ensuring that **owl:Nothing** has no members, e.g.

```
Consistency: cls-nothing2
     x rdf:type owl:Nothing
    ----------------------
```

or that no pair of individuals have both **owl:sameAs** and **owl:differentFrom** relationships, i.e.

```
Consistency: eq-diff1
     x owl:sameAs y
     x owl:differentFrom y
    ----------------------
```

Consistency checks that have consequences are similar to normal rules, except that the entailments are not added to the data model, rather they are used to ensure that the inferred statements exist in the repository. If they are not present then a consistency violation is indicated.

3.3. owl:sameAs optimization

owl:sameAs is an OWL predicate used to declare that two different URIs denote one and the same real-world concept. Most often, it is used to align identifiers from different datasets that refer to the same real-world entity. For instance, in DBPedia, the URI of Vienna is http://dbpedia.org/page/Vienna, while in Geonames [22] 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 linking data from different data sources.

Following the formal definition of OWL (OWL 2 RL, to be more specific), whenever two URIs are declared equivalent, all statements that involve one of the URIs should be "replicated" with the other URI 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
```

Since as **gno:parentFeature** is a transitive relationship, in the course of inference it will be derived that the city of Vienna is also part of Austria:

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

Due to 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 we consider that Austria, too, has an equivalent URI in DBpedia,

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

we 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, we had 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). As we see, inference without **owl:sameAs** inflated the dataset by 25% (one new statement on top of 4 explicit ones), while **owl:sameAs** related inference increased the dataset by 175% (7 new statements). Considering that Vienna also has a URI in UMBEL [23], which is also declared equivalent to the one in DBpedia, the addition of one more explicit statement for this alignment, will trigger the inference of 4 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, given a set of N equivalent URIs, N^2 **owl:sameAs** statements will be generated for each pair of URIs (we should admit though that in reality, there are not that many examples of large **owl:sameAs** equivalence classes). Therefore, although **owl:sameAs** is useful for interlinking RDF datasets, its semantics causes considerable inflation in the number of inferred statements that should be considered during inference and query evaluation (either through forward- or through backward-chaining).

To overcome this problem, BigOWLIM handles **owl:sameAs** in a specific manner. In its indices, each set of equivalent URIs (an equivalence class with respect to **owl:sameAs**) is represented by a single super-node. This way, BigOWLIM does not inflate the indices and, at the same time, retains the ability to enumerate all the required solutions to query requests. Special care is taken to ensure that this 'trick' does not hinder the ability to distinguish explicit from implicit statements.

Equivalence expansion can be switched on and off when executing queries, so that when desired, only one URI is used for a particular resource when returning query results. This can make a dramatic difference to the number of 'duplicated' results returned.

3.4. Retraction of assertions

As mentioned above, OWLIM materializes all inferred statements at load time and whenever new statements are added to the repository. This has the highly desirable advantage that query answering is very fast, due to the fact that no further inference needs to be done. Updates that simply add new statements are treated in the same way as at load time, i.e. new statements are fed to the inference engine that applies the inference rules (making joins across new statements with existing statements) until no new inferences are obtained. Since the semantics (both standard and custom) must be monotonic, insert operations incrementally add to the set of explicit and inferred statements. However, retracting explicit statements that are used to infer other statements becomes more complicated. In SwiftOWLIM, this is achieved by simply invalidating all inferred statements and re-computing the full-closure whenever an update is committed. This has the advantage of simplicity of implementation, but the disadvantage of poor update performance and lack of scalability.

BigOWLIM has a specific optimization for handling delete operations that updates the full-closure incrementally. This technique labels statements to be deleted and then uses forward-chaining to identify those statements that can be inferred from them, followed by backward chaining to identify those inferred statements that are still supported by other means.

The result is that delete performance is only slightly worse than the insertion of new statements. This allows the repository to handle rapidly changing data even when answering queries over tens of billions of statements.

3.5. Transaction management

OWLIM supports the 'read committed' transaction isolation level. It guarantees that changes will not impact query evaluation, before the entire transaction they are part of is successfully committed. It does not guarantee that execution of a single transaction is performed against a single state of the data in the repository. Regarding concurrency, multiple update/modification/write transactions can be initiated and stay open simultaneously, i.e. one transaction does not need to be committed in order to allow another transaction to complete Furthermore, update transactions are processed in sequence and do not block read requests in any way, i.e. hundreds of SPARQL queries can be evaluated in parallel (the processing is properly multi-threaded) while update transactions are being handled on separate threads.

One should note that OWLIM performs materialization, making sure that all the statements which can be inferred from the current state of the repository are indexed and persisted (except for those compressed due to the owl:sameAs optimisation, see section 3.3). By the time the commit method completes, all reasoning activities related to changes introduced by the corresponding transaction will have already been performed.

3.6. Benchmarks

There are few widely accepted benchmarks for semantic repositories and all of them fail to address all aspects of the functioning of a particular engine. The Berlin SPARQL Benchmark [17] (BSBM) evaluates the performance of query engines in an e-commerce use case: searching products and navigating through related information. Randomized query mixes (each consisting of 25 queries) are evaluated continuously through a client application that communicates with the repository through a SPARQL end-point. However, the benchmark does not require any inference to take place in the repository and is targeted purely at measuring query-answering performance. Recent evaluation results [19] for some of the most popular engines show that BigOWLIM has the best loading performance for the 100 million dataset being three times faster than the second best. BigOWLIM also has the best query performance for the reduced query mix.

The Lehigh University Benchmark [18] (LUBM) is probably the most popular benchmarking framework for semantic repositories. It uses a relatively simple OWL ontology describing a university organization structure with synthetically generated datasets. The data generated for each university includes a number of departments and related individuals together with relevant descriptions and relations between them. The framework separately measures loading and query performance and inference is required in order to answer queries correctly. However, some important aspects of semantic repositories are not measured in this benchmark, such as update or delete performance.

LUBM(8000) includes data for 8000 universities and contains about 1.1 Billion explicit statements. It is a commonly used as a benchmark, because it is processable by a reasonable cross-section of the best performing semantic repository products. BigOWLIM will load this dataset in 14 hours on a computer costing less than 2000 US dollars (2.93GHz quad-core, 12GB memory and three 320GB disks in a RAID 0 configuration) and will answer all queries correctly within 46 minutes.

However, BigOWLIM has been measured with much larger datasets, including LUBM(90000) that contains over 12 Billion explicit statements (nearly 21 Billion after inference). The loading time of this dataset with OWL-Horst semantics is approximately 290 hours on a machine with 2 quad-core, 2.5GHz processors and 64GB memory.

Another independent benchmark in the context of a commercial image retrieval system [20] compared a number of the leading semantic repositories. An excerpt from the conclusion states that "In our tests, BigOWLIM provides the best average query response time and answers the maximum number of queries for both the datasets ... it is clear to see that execution speed-wise BigOWLIM outperforms AllegroGraph and Sesame for almost all of the dataset queries."

3.7. Triplesets

Triplesets are an extension to the RDF data model that offers greater flexibility than standard named graphs. There are many situations when it is desirable to label a subset of statements from a repository independent of their context, where there is a many to many relationship between the names of subsets and the triples they contain. In these situations, where statements conceptually belong to more than one set, triplesets can be used to associate a name (URI) with any combination of statements, irrespective of their

context and/or membership of some other tripleset, see the example in Fig. 1.

In principle, the model extends the idea of a 'quad' (subject predicate object context) to include an unbounded list of tripleset identifiers. Note how some of the statements (shown as dashed horizontal lines) from the named graphs are associated with more than one tripleset.

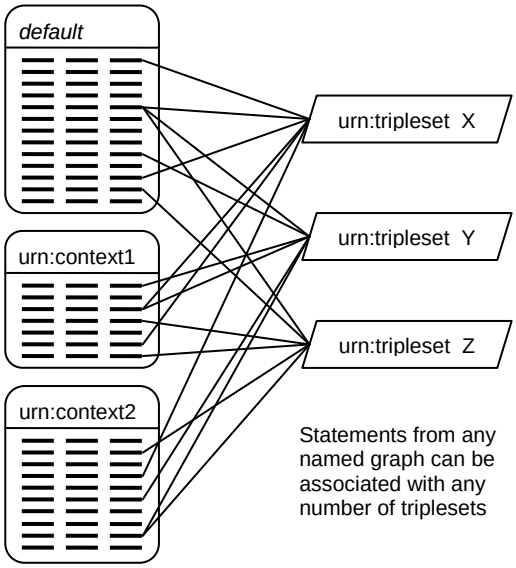

Fig. 1 Example associations of triplesets with statements

The tripleset data model has been present in OWLIM for some years, but is not accessible through the standard interfaces of Sesame that adhere to the standard RDF data model. However, the Ontology Representation and Data Integration [14] (ORDI) framework that uses OWLIM for its implementation does expose this feature through a Java API that allows statements to be associated and disassociated with tripleset identifiers. Furthermore queries can be executed that execute over only those statements associated with specified.

4. Beyond RDF and SPARQL

4.1. RDF Rank

RDF Rank is a technique to measure the relevance of entities by examining their interconnectedness. A numerical weighting is computed for every node (URIs and literals) in the entire RDF graph and stored in a special index. The weights are floating point numbers with values between 0 and 1, and are made available via a special system predicate so that the popularity of entities can be used to order query results. At a high level, the approach is similar to the way in which internet search engines order results, such as how Google orders results using PageRank.

4.2. Full text search

Full-text search (FTS) concerns retrieving text documents out of a large collection using keywords or, more generally, by tokens (represented as sequences of characters). Formally, the query represents an unordered set of tokens and the result is set of documents, relevant to the query. In a simple FTS implementation, relevance is Boolean: a document is either relevant to the query, when it contains all the query tokens, or not. More advanced FTS implementations deal with a degree of relevance of the document to the query, usually judged on some sort of measure of the frequency of appearance of each of the tokens in the document normalized versus the frequency of their appearance in the entire document collection. Such implementations return an ordered list of documents, where the most relevant documents come first.

When compared to a structured query, e.g. SPARQL, FTS is a different information access method based on a different query syntax and semantics, where the results are also displayed in a different form. FTS and databases usually require different types of indices too. The ability to combine these two types of information access methods is very useful for a wide range of applications. Many relational DBMS support some sort of FTS (which is integrated into the SQL syntax) and maintain additional indices that allow efficient evaluation of FTS constraints. Typically, relational DBMS allow the user to define a query, which requires specific tokens to appear in a specific column of a specific table. In SPARQL there is no standard way for the specification of FTS constraints. In general, there is neither a well defined nor widely accepted concept for FTS in RDF data. Nevertheless, some semantic repository vendors offer some sort of FTS in their engines. This section documents the FTS supported by BigOWLIM.

Two approaches are implemented in BigOWLIM, a proprietary implementation called 'Node Search', and a Lucene-based implementation called 'RDF Search'. Both approaches enable OWLIM to perform complex queries against character data, each with their functional differences outlined in Table 1. There can be considerable differences between the indexing

and search speed of the two FTS implementations. Performance-conscious users are recommended to experiment with the performance of both methods using datasets and queries representative for the intended application.

Node Search (when indexing only literals) is similar to typical FTS implementations in relational DBMS. However, Node Search can also index the URIs of all entities, i.e. the subjects and objects of all statements. This makes it particularly useful for executing queries when the exact spelling of an entity's URI is not known.

Table 1 Comparison of Full-Text Search implementations

	Node Search	RDF Search
Query format	List of tokens	List of tokens (with Lucene query extensions)
Result format	Unordered set of nodes	Ordered list of URIs
Textual representation	For literals: the string value. For URIs and B-nodes: tokenized URL	Concatenation of the text representations of the nodes from the molecule (1-step neighbourhood in the graph) of the URI
Relevance	Boolean, based on presence of the query tokens in the text	Vector-space model, reflecting the degree of relevance of the text and the RDF rank of the URI
Implementation	Proprietary full-text indexing and search implementation	The Lucene engine is integrated and used for indexing and search

RDF Search is a novel information retrieval concept that allows for the efficient extraction of RDF resources from huge datasets, where ordering of the results by relevance is crucial.

Both techniques embed full-text search patterns in to standard query formats, i.e. SPARQL or SeRQL. Extra statement patterns are added that use special system predicates thus enabling powerful hybrid queries.

BigOWLIM integrates Lucene [24] – a high-performance, full-featured text search engine – to index the entire repository, i.e. all nodes including both URI local names and literals. For each node in the repository its surrounding molecule is computed, i.e. the collection of statements where this node appears as the subject or object. Then each molecule is converted into a single string document by concatenating the textual representation of all the nodes in the molecule and this document is indexed by Lucene. If a node's RDF Rank is available it is stored in Lucene's index as a boosting factor that will later on influence the selection order.

The facility for integrating a Lucene query with a normal SPARQL query is achieved with a special system predicate. The query in Fig. 2 gives an example of this. The intention here is to retrieve entity identifiers and labels, where those labels contain a token similar to 'air' and a token similar to 'plane'.

```
PREFIX rdfs: <http://.../rdf-schema#>
PREFIX onto: <http://www.ontotext.com/>

SELECT * WHERE {
 ?entity rdfs:label ?label .
 ?label onto:luceneQuery "air~ AND plain~".}
```

Fig. 2 An example RDF Search query using Lucene

This combination of ranking RDF molecules together with full-text search provides a powerful mechanism for querying/analysing datasets even when the schema is not known. This allows for keyword-based search over both literals and URIs with the results ordered by importance/interconnectedness.

FactForge [25] is a demonstrator for this technology that includes eight of the central LOD datasets. This publicly available and free to use Web application uses Node Search (for auto-completion of entered tokens), RDF Search for retrieving statements and RDF Rank for ordering results by relevance. This combination of technologies provides for powerful, user-guided data-mining over a large proportion of the core LOD datasets.

4.3. Replication cluster

BigOWLIM can be used in a cluster configuration where replication is used to improve resilience and provide scalable query answering.

The query performance of the cluster represents the sum of the throughputs that can be handled by each of the instances. In a simple configuration of 3 or 4 worker nodes, hundreds of thousands of query requests can be answered per hour while at the same

time processing thousands of updates per hour – with non-trivial inference.

In a cluster configuration, there are two types of nodes: Masters and workers. Masters act as the gateway to the cluster and all read/write requests go through these nodes. A cluster can have more than one master node, but only one is allowed to operate in read/write mode. The other master nodes operate in read-only mode, otherwise known as 'hot-standby'. They can be used for marshalling read requests and can take over handling updates if the current read/write master fails. Worker nodes are standard BigOWLIM instances exposed by the Sesame HTTP server – a servlet running in Tomcat or similar. Read and write requests are passed to the workers from the master nodes. This simple arrangement allows for a great deal of flexibility in the design of a cluster topology. The example given in Fig. 3 has two master nodes and three worker nodes. At any moment in time, clients of the cluster can send read requests (queries) to either master node, but updates can only be handled by the master in read/write mode. If this master node should fail, the hot standby master can be brought in to read/write mode and from then on will handle both read requests and updates, as well as taking over responsibility for ensuring the synchronization of all the worker nodes.

Each master node implements a JMX MBean [26] that is accessible using standard Java instrumentation tools, such as JConsole [26], and can be used to monitor and control the cluster while it is running. Typical activities supported include the monitoring of the health of each node, statistics gathering, adding and removing worker nodes.

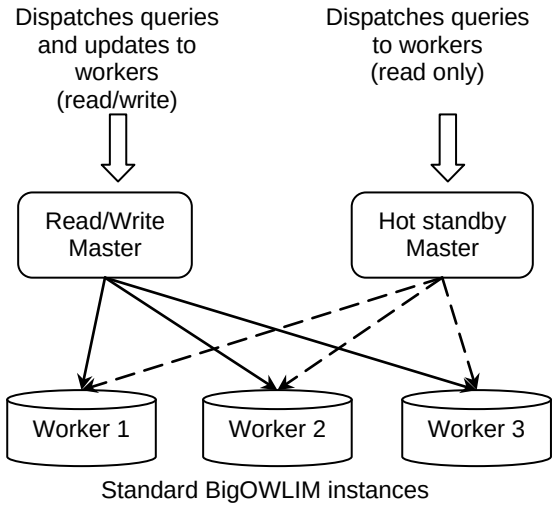

Fig. 3 A typical replication cluster configuration

During normal operation, a master node will keep track of the size of each worker's read request queue, such that each read request is sent to the worker with the shortest read queue. Update requests are handled differently. First of all, the update is tested against a single worker node. If the update is successful and subsequent consistency checks pass then the update request is considered 'safe' and is passed to the rest of the worker nodes. Master nodes take additional care to ensure that the states of all worker nodes are properly synchronized and if an anomaly is detected, the problem worker node is released from the cluster. The monitor and control JMX interface can be used to return worker nodes to the cluster and initiate their synchronization.

In the event of a failure of a worker node, the performance degradation is graceful with respect to the number of healthy workers. The cluster can remain operational with just a single worker node.

4.4. RDF Priming

RDF Priming is a technique that is used to select a subset of available statements for use as the input to query answering. It is based upon the concept of spreading-activation [10] as developed in the field of cognitive science.

RDF Priming is a scalable and customizable implementation of the popular connectionist method on top of RDF graphs. It allows the 'priming' of large datasets with respect to concepts relevant to the context and to the query. It is implemented in the BigOWLIM engine and controlled using SPARQL ASK queries.

The priming module is highly configurable, where the starting nodes, initial activation values, activation pathways, decay factors, threshold values and number of cycles can be individually set. Additionally, the number of worker threads used for computing and propagating activation values in a priming cycle can be specified.

The principles can be explained by way of the following example. Consider the following query that might be executed over the DBPedia:

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT * WHERE {
  ?x dbp:class dbr:V8.}
```

This query will return around 20 results for various engine and car types. However, if the agent using BigOWLIM is operating with a particular interest in certain concepts and those related to them, say the Ford Motor company and a particular make of car, then these two entities could be used to start a priming cycle that selects statements 'close' to these concepts. A sequence of SPARQL ASK queries can be used to set up the priming parameters, including some weightings for suitable predicates. The following query can be used to specify the two starting nodes mentioned earlier:

```
PREFIX onto: <http://www.ontotext.com#>
PREFIX dbr:  <http://dbpedia.org/resource/>
ASK { dbr:1955_Ford onto:activateNode
      dbr:Ford_Motor_Company }
```

After initiating the spreading of activations with another ASK query, the selected statements will be used as input to subsequent queries. Re-running the example query will return a smaller result set containing members of the V8 DBPedia class more closely related to the Ford Motor company and the chosen model of car.

It should be noted that RDF Priming is different from RDF Rank, in that RDF Priming involves selecting a subset of statements by propagating activation values in multiple hops starting from the specified entities. RDF Rank on the other hand, simply counts the number of connections for each node.

4.5. Notifications

Notifications are a publish/subscribe mechanism for registering and receiving events from a BigOWLIM repository whenever new triples matching a certain graph pattern are inserted. The user of the notifications API registers for notifications by providing a graph pattern involving triple patterns combined by means of joins and unions at any level. The order of the triple patterns is not significant.

In general, notifications will be sent for all incoming triples that contribute to a solution of the graph pattern. Furthermore, any statements inferred from newly inserted statements will also be subject to handling by the notification mechanism, i.e. new implicit statements will also be notified to clients when the requested triple pattern matches.

The purpose of the notification service is to enable the efficient and timely discovery of newly added RDF data. Therefore it should be treated as a mechanism for giving the client a hint that certain new data is available and should not be used as an asynchronous SPARQL evaluation engine.

5. Conclusion

The emerging Web of data has provided new challenges for software components that must expose this data and enable its widespread consumption. The OWLIM family of semantic repositories is ideally suited to this task due to its ability to store, reason and answer queries using the massive datasets involved. In addition to world-leading RDF processing performance, OWLIM offers a range of advanced features that seamlessly integrate with existing query standards and provide a variety of alternative data access methods.

OWLIM continues to evolve with various new features planned for the near future. The next release of OWLIM will include enhanced support for geo-spatial data and some of the widely accepted geo-spatial vocabularies. Specialized indices will be used to access spatial data and a range of SPARQL extension functions will allow for expressive queries using 2D and 3D geometry.

The next release will also include interfaces that support the JENA RDF framework, enabling OWLIM to be used with both Sesame and JENA, the two most widely used RDF frameworks.

Later releases will include more advanced full-text search and indexing options based on Lucene, with the ability to create and use multiple Lucene indices each parameterized according to the task at hand. Configuration parameters will allow better control over what statements to include in the RDF molecule. The size of the molecule (number of statement 'hops' from each node) will be controllable as well the choice of which statements to include based on the selected predicates or the selected language tags of literals.

Later releases will expose the existing support for the extended RDF model based on triplesets.

The current set of advanced features and world-leading performance have helped to position OWLIM as the semantic repository of choice for all environments that manage RDF data, particularly for Web-scale applications. The future evolution of OWLIM towards better compatibility and even more powerful data management features will ensure the continued uptake of this technology.

The development of OWLIM has been partly supported by SEKT [27], TAO [28], TripCom [29], LarKC [30], SOA4ALL [31], and other FP6 and FP7 European research projects.

References

1. Klyne, G; Carrol , J. J; (eds). (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 Feb. 2004. http://www.w3.org/TR/rdf-concepts/
2. Christian Bizer, "The Emerging Web of Linked Data," IEEE Intelligent Systems, pp. 87-92, September/October, 2009
3. Atanas Kiryakov, Damyan Ognyanov, Dimitar Manov, OWLIM – a Pragmatic Semantic Repository for OWL, In Proc. of Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2005), WISE 2005, 20 Nov, New York City, USA. Springer-Verlag LNCS series, LNCS 3807, pp.182-192.
4. Dean, M; Schreiber, G. – editors; Bechhofer, S; van Harmelen, F; Hendler, J; Horrocks, I.; McGuinness, D. L; Patel-Schneider, P. F.; Stein, L. A. (2004). OWL Web Ontology Language Reference. W3C Recommendation, 10 Feb. 2004. http://www.w3.org/TR/owl-ref/
5. BRICKLEY, D.; GUHA, R.V, RDF Vocabulary Description Language 1.0: RDF Schema, W3C (10 Feb 2004) http://www.w3.org/TR/rdf-schema
6. BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. (2001) "The Semantic Web". Scientific American, May 2001
7. Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2006.
8. ter Horst, H. J. Combining RDF and Part of OWL with Rules: Semantics, Decidability, Complexity. In Proc. of ISWC 2005, Galway, Ireland, November 6-10, 2005. LNCS 3729, pp. 668-684
9. J.Broekstra and A.Kampman, "RDF(S) manipulation, storage and querying using Sesame" In Demo Proc. of the 3rd Intl. Semantic Web. Conf., Hiroshima, 2004
10. Brian McBride, Jena: A Semantic Web Toolkit, IEEE Internet Computing, v.6 n.6, p.55-59, November 2002
11. Hervé Gallaire, Jack Minker (Eds.): Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, 1977. Advances in Data Base Theory, Plenum Press, New York, 1978, ISBN 0-306-40060-X.
12. Collins A.M., Loftus E.F., A spreading-activation theory of semantic processing (1975) Psychological Review, 82 (6), pp. 407-428.
13. Berners-Lee, T. (2006). *Design Issues: Linked Data*. http://www.w3.org/DesignIssues/LinkedData.html
14. Kiryakov, A; Ognyanov, D; Kirov, V. (2004) An Ontology Representation and Data Integration (ORDI) Framework. DIP project deliverable D2.2. http://dip.semanticweb.org
15. Motik, B; Cuenca Grau, B; Horrocks, I; Wu, Z; Fokoue, A; Lutz, C. (eds.) (2009). OWL 2 Web On-tology Language Profiles. W3C Candidate Recom-mendation 11 June 2009. http://www.w3.org/TR/owl2-profiles/
16. TRREE – Triple Reasoning and Rule Entailment Engine. Home page. http://ontotext.com/trree/
17. Schmidt1, M; Hornung1, T; Meier1, M; Pinkel, C; Lausen, G, Semantic Web Information Management, Springer Berlin Heidelberg 2010, pp. 371-393
18. Guo Y., Pan Z., Heflin J., LUBM: A benchmark for OWL knowledge base systems, (2005) Web Semantics, 3 (2-3), pp. 158-182.
19. Bizer, Ch., Schultz, A.: BSBM Results for Virtuoso, Jena TDB, BigOWLIM (November 2009). http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V5/index.html
20. Thakker , D., Osman, T., Gohil, S., Lakin, P, A Pragmatic Approach to Semantic Repositories Benchmarking. In Proc. of the 7th Extended Semantic Web Conference, ESWC 2010.
21. Auer1, S; Bizer, C; Kobilarov, G; Lehmann1, J; Cyganiak, R; Ives, Z, DBpedia: A Nucleus for a Web of Open Data, Springer Berlin / Heidelberg (2007), Lecture Notes in Computer Science, pp. 722-735
22. The GeoNames geographical database, homepage: http://www.geonames.org/
23. Upper Mapping and Binding Exchange Layer, homepage: http://www.umbel.org/
24. Apache Lucene, a high-performance, full-featured text search engine library, homepage: http://lucene.apache.org/
25. FactForge, a reason-able view to the web of data, homepage: http://factforge.net/
26. Java Management Extensions (JMX), homepage: http://download-llnw.oracle.com/javase/1.5.0/docs/guide/jmx/
27. Semantic Enabled Knowledge Technologies (SEKT), European Research Project, homepage: http://www.sekt-project.com/
28. Transitioning Applications to Ontologies (TAO), European Research Project, homepage: http://www.tao-project.eu/
29. Triple Space Communication (TripCom), European Research Project, homepage: http://www.tripcom.org/
30. LarKC: The Large Knowledge Collider (LarKC), European Research Project, homepage: http://www.larkc.eu/
31. Service Oriented Architectures for All (SOA4All), European Research Project, homepage: http://www.soa4all.eu/