Semantic Web 2 (2011) 33–42
DOI 10.3233/SW-2011-0026
IOS Press

OWLIM: A family of scalable semantic repositories

Editor: Axel Polleres, DERI, National University of Ireland, Galway
Solicited review(s): Andy Seaborne, Epimorphics Ltd, UK; Aidan Hogan, DERI, National University of Ireland, Galway; Giovambattista Ianni, Universita' della Calabria, Italy

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

Abstract. An explosion in the use of RDF for representing information about resources 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 [22] family of semantic repositories that are 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 [23] (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 motivates the development of semantic repositories by giving 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 an RDF storage, reasoning and query-answering platform suitable for large-scale mission critical applications, for example the BBC's World Cup 2010 website [3]. Section 4 covers some advanced features of BigOWLIM that go beyond the RDF/RDFS [9]/OWL [13] language stack and show how other AI and text-processing related functions are combined to enable powerful data-mining applications. Section 5 deals with the performance and resilience of BigOWLIM systems, providing some comments regarding recent independent evaluations, a discussion of BigOWLIM Replication Cluster technology and some early results concerning benchmarking the scalability of query processing when using a cluster deployed in the cloud. Section 6 gives a very brief overview of the development of OWLIM and its increasing adoption in commercial environments. Section 7 concludes with a summary of the information presented and some indications for the future evolution of the OWLIM family.

* Corresponding author. E-mail: barry.bishop@ontotext.com.

2. Background and motivation

The Resource Description Framework (RDF) was designed as a language for representing information about resources in the World Wide Web leading to the concept of the Semantic Web [5]. Due in part to its simple and flexible data model, it is more widely used for general purpose knowledge management and modeling, the most notable being "Linked Data" [6] a concept outlined by Tim Berners-Lee [4]. The principal 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 [2] – 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 have a pure Java, native RDF database implementation. Currently there are two variants of OWLIM optimized for different operating environments: SwiftOWLIM is an in-memory RDF database, inference-engine and query-answering engine. It uses 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 1,000 USD machine using non-trivial inference. SwiftOWLIM is free-for-use and based on the Triple Reasoning and Rule Entailment Engine [38] (TRREE). BigOWLIM is the commercial version published under a per CPU 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 insert and delete performance even when managing tens of billions of statements of linked open data. Query performance with such dataset sizes is likewise good, with sub-second response times for all the example queries found on the FactForge [14] SPARQL query page and good results from benchmarks [29]. While there is no theoretical limit on the maximum number of statements that can be processed by BigOWLIM, the practical limit for a machine with 64GB RAM is around 20 billion statements – any more than this and the loading performance drops off dramatically.

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 semantic repositories packaged as a Storage and Inference Layer (SAIL) for the Sesame openRDF [10] framework. This popular framework brings compatibility with the common RDF serialisations (XML, N3, N-Triples, Turtle, TRIG, TRIX) and support for the

SPARQL [28] and SeRQL [11] query languages. SwiftOWLIM has no special query processing engine of its own and relies on the Sesame framework. However, BigOWLIM implements its own query model and optimizations. 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 [34]) where Datalog [15] 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 to 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 into 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;
- **rdfs** – RDFS semantics using rule entailment [18], but without data-type reasoning, i.e. without the literal generalization and related rules;
- **owl-horst** – equivalent to pD* [34], again without data-type reasoning;
- **owl-max** – RDFS plus that part of OWL-Lite that can be captured in rules (deriving functional and inverse functional properties, all-different, subclass by union/enumeration, min/max cardinality constraints, etc);
- **owl2-rl** – the OWL2 RL profile (fragment of OWL2 Full that is amenable for implementation on rule-engines [27]), but without data-type reasoning.

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 rule-sets are included in the distribution 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 thing. Hence it is often used to align identifiers from different datasets that refer to the same thing.

For example, consider two URIs that identify Sofia (that is a part of Bulgaria) and three URIs that identify Bulgaria, these would be aligned using the following statements:

dbpedia:Sofia	owl:sameAs	geonames:727011
geonames:727011	geo-ont:parentFeature	geonames:732800
dbpedia:Bulgaria	owl:sameAs	geonames:732800
dbpedia:Bulgaria	owl:sameAs	opencyc-en:Bulgaria

Because **owl:sameAs** is a transitive and symmetric relationship, we can treat all identifiers that have been connected using this predicate as belonging to

the same identity class. This means that if any statement contains a member of any identity class, then that statement should be replicated during inference for every other member of the identity class. For the Sofia-Bulgaria example given above, this leads to the entailment of ten more statements, i.e.

geonames:727011	owl:sameAs	dbpedia:Sofia
geonames:732800	owl:sameAs	dbpedia:Bulgaria
geonames:732800	owl:sameAs	opencyc-en:Bulgaria
opencyc-en:Bulgaria	owl:sameAs	dbpedia:Bulgaria
opencyc-en:Bulgaria	owl:sameAs	geonames:732800
dbpedia:Sofia	geo-ont:parentFeature	geonames:732800
dbpedia:Sofia	geo-ont:parentFeature	opencyc-en:Bulgaria
dbpedia:Sofia	geo-ont:parentFeature	dbpedia:Bulgaria
geonames:727011	geo-ont:parentFeature	opencyc-en:Bulgaria
geonames:727011	geo-ont:parentFeature	dbpedia:Bulgaria

As can be seen, inference with **`owl:sameAs`** has inflated the initial four statements with a further ten, a 250% increase even for this trivial example. Furthermore, **`owl:sameAs`** is also reflexive and the OWL semantics dictate that all resources have a **`owl:sameAs`** relationship with themselves, therefore there should be a further five statements inferred:

dbpedia:Sofia	owl:sameAs	dbpedia:Sofia
geonames:727011	owl:sameAs	geonames:727011
geonames:732800	owl:sameAs	geonames:732800
dbpedia:Bulgaria	owl:sameAs	dbpedia:Bulgaria
opencyc-en:Bulgaria	owl:sameAs	opencyc-en:Bulgaria

Although this is a simple example, it provides a good indication about the performance implications of using **`owl:sameAs`** alignment in LOD. Because **`owl:sameAs`** is a transitive, reflexive, and symmetric relationship, given an identity class of N equivalent URIs, N^2 **`owl:sameAs`** statements will be generated for every combination of pairs of URIs. Thus, 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 includes special handling for **`owl:sameAs`**, where an entire equivalence class is indexed using a single node. In 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 optimisation 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 results returned, where statements differ only by the substitution of equivalent URIs.

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 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 is more complicated. In SwiftOWLIM, this is achieved by simply invalidating all inferred statements and re-computing the full-closure whenever a delete operation 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, but does not use additional truth maintenance data structures, such as those developed as part of the Sesame infrastructure. 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.

4. Beyond RDF and SPARQL

4.1. RDF Rank

RDF Rank is a technique to measure the relevance of entities by examining their interconnected-

ness. 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.

The algorithm that creates the weights can be configured using SPARQL ASK queries with special system predicates. Currently, only the maximum number of iterations and the lower cut-off values can be altered. The generated weights are shared by all users of the repository.

RDF Rank is particularly useful when querying very large datasets, where it can be used to identify the popular results out of many. For example, the following query returns the 100 most popular entertainers from a dataset:

```
PREFIX rank:
    <http://www.ontotext.com/owlim/RDFRank#>
PREFIX opencyc:
    <http://sw.opencyc.org/concept/>
SELECT *
WHERE {
    ?Person rdf:type opencyc:Entertainer .
    ?Person rank:hasRDFRank ?rr . }
ORDER BY DESC(?rr)
LIMIT 100
```

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 describes the FTS supported by BigOWLIM.

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 each node and its neighbors
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

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 uses Boolean-relevance and when indexing only literals is similar to typical FTS implementations in relational DBMS. However, Node

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

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

Fig. 1. An example RDF Search query using Lucene.

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.

RDF Search 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 into standard query formats, i.e. SPARQL or SeRQL, where statement patterns using special system predicates enable powerful hybrid queries.

To implement RDF Search, BigOWLIM integrates Lucene [25] – a high-performance, full-featured text search engine – to index the entire repository, i.e. all nodes using both URI local names and literals. For each node in the repository a text document is created by concatenating its text representation with those of other nodes reachable through one predicate arc, i.e. the subjects and objects of all nodes that appear in statements with the indexed node. The resulting 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. 1 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'.

This combination of ranking RDF molecules together with full-text search provides a powerful mechanism for querying/analyzing 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 [14] 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. 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 [12] 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 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 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.

A current limitation of the RDF Priming implementation is that the activation values are maintained globally, so that it is not possible for two separate users to set up their own activation values.

4.4. 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 or deleted. 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 inserted and deleted triples that contribute to a solution of the graph pattern. Furthermore, any inferred statements affected by inserts and deletes 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 or deleted RDF data. Therefore it should be treated as a mechanism for giving the client a hint that certain changes have occurred and should not be used as an asynchronous SPARQL evaluation engine.

The notification mechanism is designed to be used to trigger reactive behavior in client applications that need to respond to either inserted or deleted statements in the update stream.

5. Performance, resilience and scalability

There are few widely accepted performance benchmarks for semantic repositories and all of them fail to address all aspects of the functioning of a particular engine. This section discuss a few well-known benchmarks and some independent evaluations followed by a description of the BigOWLIM Replication Cluster and how this component improves both resilience and concurrent query processing performance.

5.1. Benchmarks

The Berlin SPARQL Benchmark [7] (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 [8] 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 [17] (LUBM) is a commonly used 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 and 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 [35] 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."

5.2. 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. 2 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 [19] that is accessible using standard Java instrumentation tools, such as JConsole [20], 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.

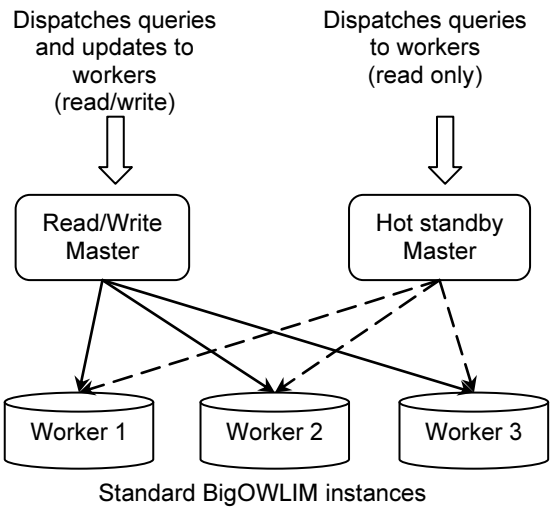

Fig. 2. A typical replication cluster configuration.

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.

5.3. BigOWLIM in the cloud

BigOWLIM Replication Cluster provides a means to dynamically improve concurrent query processing capability by increasing the number of worker nodes. Since worker nodes can be added and removed from a running cluster using management software, a cloud environment is a natural choice for

deploying a BigOWLIM cluster, especially since the cluster is resilient to the failure of individual nodes.

In order to assess the scalability behavior of BigOWLIM Replication Cluster in the cloud, Ontotext conducted a series of experiments using the Amazon EC2 [1] infrastructure. Since the intention was to measure concurrent query performance, the BSBM benchmark was selected with the 100 million statement data set and 1 thousand clients, see Section 5.1. The cluster configuration comprised 1 master node (Amazon HM-2XL instance, 34GB RAM, 4 CPU cores) and between 10 and 100 worker nodes (Amazon HM-XL instance, 17GB RAM, 2 CPU cores) all running 64 bit Linux

Unpublished results show that total query performance scales almost linearly with the number of worker nodes, where the query processing throughput of worker nodes reduces gradually in relation to the total number of nodes. The 20 node configuration was shown to be able to process more than 40,000 query mixes per hour, or 1 million SPARQL queries per hour. The 100 node configuration was able to process 200,000 query mixes, or 5 million SPARQL queries per hour.

6. Development and adoption

OWLIM was originally developed as part of the 'Semantic Knowledge Technologies' (SEKT) and Triple Space Communication' (TRIPCOM) European research projects. It still maintains a presence in European research as the core storage and inference layer in the 'Large Knowledge Collider' (LarKC) and 'Service Oriented Architectures for All' (SOA4All) integrating projects.

As with other semantic technologies, commercial uptake has been relatively slow. However, BigOWLIM is now being used in the life sciences, telecoms and publishing sectors as a flexible data-integration platform for massive amounts of heterogeneous data. BigOWLIM is also being used as part of a project sponsored by the UK Government National Archives [28] to bring new methods of search, navigation and information modeling and in doing so make the web archive a more valuable and popular resource.

One high profile use case was the inclusion of a BigOWLIM cluster as part of the publishing stack for the BBC's World Cup 2010 website [3]. This adoption of semantic technology represents a significant change in the way that the BBC publishes content in that the framework for this website does not author content directly, rather it publishes metadata about the content according to a rich ontological domain model using OWL semantics. Queries to this metadata are used to dynamically generate content for players, groups and teams. The ontology also extends to describing journalist-authored assets allowing them to be associated with the central concepts within the domain model. The peak periods for the site have seen updates of 100's per minute and around a million SPARQL queries per day.

7. Conclusions and future work

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 outstanding RDF processing performance [8], 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 geospatial data and some of the widely accepted geospatial 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 Java-based 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.

The advanced features and excellent performance of OWLIM have helped to position it as the seman-

tic 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 [32], TAO [36], TripCom [37], LarKC [24], SOA4ALL [33], and other Framework 6 and 7 European research projects.

References

[1] Amazon EC2. *Amazon elastic compute cloud*, homepage: http://aws.amazon.com/ec2/.

[2] Auer, S., Bizer, C., Kobilarov, G., Lehmann, 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.

[3] BBC World Cup 2010, homepage: http://www.bbc.co.uk/worldcup.

[4] Berners-Lee, T. (2006). *Design Issues: Linked Data*. http://www.w3.org/DesignIssues/LinkedData.html.

[5] Berners-lee, T., Hendler, J., Lassila, O., The Semantic Web. *Scientific American*, May 2001.

[6] Bizer, C., The Emerging Web of Linked Data, *IEEE Intelligent Systems*, pp. 87–92, September/October 2009.

[7] Bizer, C., Schultz, A., The Berlin SPARQL Benchmark, *International Journal on Semantic Web & Information Systems*, Vol. 5, Issue 2.

[8] Bizer, C., Schultz, A., *BSBM Results for Virtuoso, Jena TDB, BigOWLIM (November 2009)*. http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V5/index.html.

[9] Brickley, D., Guha, R.V., *RDF Vocabulary Description Language 1.0: RDF Schema, W3C (10 Feb 2004)* http://www.w3.org/TR/rdf-schema.

[10] Broekstra, J., Kampman, A., RDF(S) manipulation, storage and querying using Sesame, In *Demo Proc. of the 3rd Intl. Semantic Web. Conf.*, Hiroshima, 2004.

[11] Broekstra, J., Kampman, A., SeRQL: A Second Generation RDF Query Language, In *Proc. of SWAD-Europe Workshop on Semantic Web Storage and Retrieval* 2003.

[12] Collins, A.M., Loftus, E.F., A spreading-activation theory of semantic processing, (1975) *Psychological Review*, **82** (6), pp. 407–428.

[13] Dean, M., Schreiber, G. (Eds.), 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/.

[14] FactForge, a reason-able view to the web of data, homepage: http://factforge.net/.

[15] Gallaire, H., Minker, J. (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.

[16] GeoNames geographical database, homepage: http://www.geonames.org/.

[17] Guo Y., Pan Z., Heflin J., LUBM: A benchmark for OWL knowledge base systems, (2005) Web Semantics, **3** (2–3), pp. 158–182.

[18] Hayes, P. (Ed.) *RDF Semantics, W3C Recommendation 10 February 2004*. http://www.w3.org/TR/rdf-mt/.

[19] Java Management Extensions (JMX), homepage: http://download-llnw.oracle.com/javase/1.5.0/docs/guide/jmx/.

[20] Java Monitoring and Management Console (JConsole), http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html.

[21] Kiryakov, A., Ognyanov, D., Kirov, V. (2004) An Ontology Representation and Data Integration (ORDI) Framework. *DIP project deliverable D2.2*. http://dip.semanticweb.org.

[22] Kiryakov, A., Ognyanov, D., Manov, D., 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.

[23] 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/.

[24] LarKC: The Large Knowledge Collider (LarKC), European Research Project, homepage: http://www.larkc.eu/.

[25] Lucene, a high-performance, full-featured text search engine library, homepage: http://lucene.apache.org/.

[26] McBride, B., Jena: A Semantic Web Toolkit, (November 2002) *IEEE Internet Computing*, **6** (6), pp. 55–59.

[27] Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (Eds.) (2009). OWL 2 Web On-tology Language Profiles. *W3C Candidate Recommendation 11 June 2009*. http://www.w3.org/TR/owl2-profiles/.

[28] National Archives (UK Government Website): http://www.nationalarchives.gov.uk/.

[29] OWLIM Benchmark Web page: http://www.ontotext.com/owlim/benchmarking_index.html.

[30] Prud'hommeaux, E., Seaborne, A., SPARQL Query Language for RDF. Technical report, W3C, 2006.

[31] Schmidt, M., Hornung, T., Meier, M., Pinkel, C., Lausen, G., *Semantic Web Information Management*, Springer, Berlin/Heidelberg 2010, pp. 371–393.

[32] Semantically Enabled Knowledge Technologies (SEKT), European Research Project, homepage: http://www.sekt-project.com/.

[33] Service Oriented Architectures for All (SOA4All), European Research Project, homepage: http://www.soa4all.eu/.

[34] 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.

[35] 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*.

[36] Transitioning Applications to Ontologies (TAO), European Research Project, homepage: http://www.tao-project.eu/.

[37] Triple Space Communication (TripCom), European Research Project, homepage: http://www.tripcom.org/.

[38] TRREE – Triple Reasoning and Rule Entailment Engine, homepage: http://ontotext.com/trree/.

[39] Upper Mapping and Binding Exchange Layer (UMBEL), homepage: http://www.umbel.org/.