Semantic Web Storage with 3store

Stephen Harris and Nicholas Gibbins
Department of Electronics and Computer Science,
University of Southampton, United Kingdom
{swh,nmg}@ecs.soton.ac.uk

October 24, 2003

1 Introduction

The development and deployment of practical Semantic Web applications requires technologies for the storage and retrieval of RDF data that are robust and scalable. We have developed 3store, an RDF storage and query engine, within the Advanced Knowledge Technologies (AKT) [1] project with the explicit requirement of efficient handling of large RDF knowledge bases.

Our experiences of constructing Semantic Web applications have involved large datasets of up to twenty-five million RDF triples. For our CS AKTiveSpace application [9], we used the `hyphen.info` dataset, a constantly-updated knowledge base describing computer science research in the UK. This was constructed by members of the AKT project as a testbed for Semantic Web research, and to use as a foundation on which to build Semantic Web applications and investigate tools. This dataset is expressed in RDF form using the AKT Reference Ontology [2], which consists of around 200 classes and 150 properties. We expect other applications under development to include separate ontologies and sub-ontologies of a similar size.

2 Requirements

We determined the base scale requirements for an RDF store from our experiences and made the decision to construct a system which would be able to handle at least 20 million triples and 5000 classes and properties. Our specific requirements relate to the performance of such a system for query processing, for its ancillary capabilities and for its inferential capabilities..

Many of the applications under development in AKT require interactive-level performance while evaluating queries containing significant numbers of constraints. For example, when formulated in RDQL, the CS AKTiveSpace user interface uses queries with between four and twelve triple patterns in the `WHERE` clause, returning a few hundred result rows. These applications are commonly

based around a Web browser interface; the expectation of most users is that such an interface should be no less responsive than simple Web browsing, so response time for the queries used must be kept to the order of a few milliseconds on available hardware, in order to maintain the responsiveness of the interfaces.

An orthogonal concern is that of the time taken to assert new knowledge. The knowledge sources that the AKT project uses as a testbed for its research applications are gathered on a variety of schedules ranging from daily to monthly. Maintaining the integrity of the data while it is being reasserted is an important concern, and for this reason the time during which the knowledge base is potentially inconsistent or incomplete should be kept to a minimum. The store must also be able to import and replace RDF data sufficiently quickly to be able to refresh its contents on a daily basis, in order to deal with rapidly changing knowledge bases. Given the minimum gathering schedule, it should be possible to assert the entire AKT RDF knowledge base in a few hours and replace significant portions of it in a reasonable time for an overnight batch process.

Finally, if an RDF store is to be more than just a database for storing triples, it should be able to perform some inference over the data asserted within it. Following our requirements for scale and performance, and based on the expected inference requirements for our applications, we required only the entailments described in the RDF semantics [6], rather than those of any more expressive language such as DAML+OIL or OWL [8].

3 Implementation Details

3store is implemented in C as an RDF abstraction layer on top of an RDBMS using a schema which has been designed for the efficient storage and retrieval of triples, and uses the Raptor toolkit [3] for parsing RDF/XML syntax. Our implementation records the origin of the triples it contains at the granularity of (RDF/XML) files. Such simple context provides the basis of a mechanism for managing provenance. In our experience of building SW applications, we have found file-level provenance to be sufficient, especially compared to the cost of triple-level provenance.

The inferential capabilities of 3store are implemented as a hybrid of forward- and backward-chaining production rules, in which we have tried to find a compromise between eager entailment evaluation (which leads to efficient querying) and lazy entailment evaluation (which reduces the size of the stored data).

The previous (prototype) version of our 3store software supported an RDF-based dialect of the Open Knowledge Base Connectivity (OKBC)[5] API which used HTTP as its transport layer. This was intended as a lightweight interface by which RDF-aware clients could invoke the knowledge base through a set of web services which provided specific competences relating to the manipulation of a knowledge base in a frame-based manner. Typical examples of such services are get-class-subclasses (return the subclasses of a class), slot-has-value (return the value of a property on an object) or get-frame-sentences (return the

assertions involving a given object). The OKBC-HTTP interface was used in a number of our existing applications, so the maintenance of this interface was a requirement to ensure backwards compatibility with previous versions, as well as an opportunity to reimplement it in a more efficient manner.

In addition to this OKBC API, we felt it appropriate to implement a more natural and versatile RDF query interface, based on the RDQL query language [7]. Our existing applications made heavy use of the stored procedure capability of the OKBC API, indicating that the simple API calls were not expressive enough to support the development of sophisticated applications. This RDQL interface provides an HTTP interface that returns the results in an XML format, and a database-style C API that queries the knowledge base directly and which could be used to provide bindings in other languages.

4 Future Work

Our development of 3store has identified a expanded set of requirements which present a number of avenues for future development:

Limited OWL Support: Although our applications do not require sophisticated description logic reasoning, there are aspects of the OWL ontology language which we feel it would be advantageous to implement in isolation as a complement to the existing RDFS support. These include owl:sameAs, owl:InverseFunctionalProperty and owl:FunctionalProperty (for the identification and coalescing of coreferent resources), and owl:-TransitiveProperty, owl:SymmetricProperty and owl:inverseOf (in order to provide a more expressive characterisation of properties in our ontologies).

Truth maintenance: The experiences of the Sesame developers with their implementation of a triple-level justification-based truth maintenance system [**?**] suggest that fine-grained TM is overly costly. As an alternative, we propose to implement TM at a context-level, in order to provide more coarse-grained TM at what we hope will be a more acceptable cost.

Web Services integration: Our development of SW applications is increasing focussed on a service-level characterisation of system components, possibly mediated through some broker. To fit in this model, we intend to write an OWL-S description of the services offered by 3store and implement a SOAP interface to complement the existing HTTP RDQL interface. Our longer term goals include the ad-hoc generation of service descriptions that correspond to ad-hoc query templates, so providing an application domain-specific characterisation of 3store knowledge services.

API rationalisation and deprecation: In the 3store prototype, all applications used the OKBC API, but all subsequent applications under the current version use the RDQL query interface. Given the complexity of

maintaining both, we propose to deprecate the OKBC interface within 3store and replace it with an OKBC abstraction layer which sits above the RDQL query interface.

Distribution: There are two areas in which we intend to address distribution. In the short term, distribution of the underlying RDBMS may improve overall system performance. In the longer term, our goal is the development of a federated RDF store, although this raises many issues relating to the efficient implementation of sound and complete inference in a distributed environment.

References

[1] Advanced Knowledge Technologies. `http://www.aktors.org/`, 2000.

[2] The AKT Reference Ontology. `http://www.aktors.org/publications/ontology/`, 2002.

[3] Dave Beckett. Raptor RDF Parser Toolkit. `http://www.redland.opensource.ac.uk/raptor/`, 2003.

[4] Jeen Broekstra and Arjohn Kampman. Sesame: A generic architecture for storing and querying RDF and RDF Schema. Technical report, Aidministrator Nederland b.v., October 2001. `http://sesame.aidministrator.nl/publications/del10.pdf`.

[5] Vinay Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. Open knowledge base connectivity. Technical report, OKBC Working Group, April 1998. `http://www.ai.sri.com/~okbc/spec.html`.

[6] Patrick Hayes. RDF Semantics. Working draft, World Wide Web Consortium, January 2003. `http://www.w3.org/TR/rdf-mt/`.

[7] Hewlett-Packard Labs. RDQL - RDF data query language. `http://www.hpl.hp.com/semweb/rdql.htm`, 2003.

[8] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Working draft, World Wide Web Consortium, March 2003. `http://www.w3.org/TR/owl-features/`.

[9] Nigel R. Shadbolt, monica m.c. schraefel, Nicholas Gibbins, and Stephen Harris. CS AKTive Space: or how we stopped worrying and learned to love the semantic web. `http://eprints.ecs.soton.ac.uk/archive/00007440/`, 2003.