Temporal Views Over RDF Data

Geetha Manjunath
Hewlett-Packard Labs
Bangalore, India
+91-080-2205-2259
geetham@hp.com

R.Badrinath
Hewlett-Packard
Bangalore, India
+91-80-2516-5005
badrinath@hp.com

Craig Sayers
Hewlett-Packard Labs
Palo Alto, US
+1-650-857-3838
craig.sayers@hp.com

Venugopal K S
Hewlett-Packard
Bangalore India
+91-80-2516-5005
venuks@hp.com

ABSTRACT
Supporting fast access to large RDF stores has been one of key challenges for enabling use of the Semantic Web in real-life applications, more so in sensor-based systems where large amounts of historic data needs to be stored. We propose a semantics-based temporal view mechanism that enables faster access to time-varying data by caching into memory only the required subset of RDF triples. We describe our experience of implementing such a framework in the context of a wide area network monitoring system. Our preliminary results show that our solution significantly improves client access time and scales well for moderate data sets.

Categories and Subject Descriptors
D.2.11 [**Software Architectures**] D.2.13 [Reusable Software]

General Terms
Languages, Performance, Experimentation, Design

Keywords
Semantic Web, views, time varying data, sensors, RDF, OWL, networking, data, Jena, Joseki, SPARQL, ontology, snapshots

1. INTRODUCTION
Business applications are increasingly using sensor data to provide intelligent services. For example analyzing the performance of a network requires repeatedly measuring round-trip packet delays to provide timely and accurate knowledge of highly dynamic network properties [1]. We believe an RDF-based store for sensor data would enable the system to be flexible enough to adapt to schema variations of data due to changes in sensor infrastructure as well as enable sophisticated applications that infer newer information using an ontology. Today, as the cost of storage and processing have reduced, one can afford to store large amounts of information for long periods of time, and deploy innovative new applications and services to mine this huge data store. Fast access to such large RDF stores is therefore necessary.

A key characteristic of sensor data is that it is time varying in nature, and its temporal aspect is an important facet of the applications processing this data. A typical operation in such applications is to extract temporal snapshots of data. For some applications, only specific snapshots of the data are relevant (say latest events), while some may create live snapshots for efficiency reasons. Thus, snapshot-generation is an important component in sensor-data based applications. This brings forth a need for a simple way of describing and extracting a relevant subset of information (materialized views) over large RDF stores.

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

While view materialization is well understood for traditional relational databases, it remains an active research for XML and RDF stores. Further, using current methodologies[2,3,5], operations on custom snapshots over time-varying data not only requires a sequence of queries (using SQL or SPARQL) but also extensive program logic to update the snapshot to track data changes over time. We introduce a simple method of generating and operating on snapshots through a mechanism that we call as *temporal views*. Here, an application specifies a *live* snapshot of interest in a declarative fashion and then makes only simple queries over this snapshot. It is naturally, the responsibility of the runtime system to update the snapshot when data changes.

In this paper, we propose a temporal view language to enable a semantic description of application-relevant data subsets. We briefly describe the design and implementation of our runtime framework that supports live temporal views over RDF data. We also present the preliminary results showing improved query efficiency through snapshot generation using data from a wide area network monitoring system [1] over the PlanetLab [6].

2. OUR SOLUTION
In order to keep the business applications less sensitive to schema changes in sensor data and to support inference over raw data, we decided to store the sensor data as RDF triples. Over this RDF store is the core part of our solution, a view mechanism that maintains relevant subsets of base data for every application. The client applications first declaratively specify the data subset of interest and then use SPARQL queries to retrieve required information from the materialized view (also an RDF model). The view generator uses optimized SPARQL queries over the base data to maintain liveness of the view model.

Our new view specification language based on OWL brings in the power and expressivity of ontology languages for view specification. Using this language, an application can define a view using the semantics of the domain and temporal properties of the data. Every view specification includes:
a) Part of the schema of interest (the "TBox" in description logic)
b) Instances of data of interest (the "Abox" in description logic)
c) The temporal aspects of the snapshot of interest.

A temporal class defined in the specification gives the temporal aspect of the selected subset of data (say, *latest n events only or events collected in the last m minutes*) which is based on a given time property (a view:Time). The rest of the specification, described in [4], selects what else comes with these instances (e.g. *while accessing network routes, do you want access to host information also*?). Now, for every live snapshot required by a new application, a temporal class is defined in its view specification. We are unable to give a complete description of our view language due to space constraints. However, an example view specification is shown in Figure 1 and is described in the context of our test application in section 3.

Once an application-specific view definition is in place, most of the data processing required by the application can be expressed using SPARQL, the RDF query language – thus reducing the application development time.

To support temporal views, we have introduced a novel runtime paradigm as well: we define a concept of "notional" equivalence classes that partition the data instances; perform a temporal ordering within each class; and pick instances from each class that meet the specified temporal criteria for the materialized view.

3. EXPERIMENTS

We have integrated the complete view infrastructure within the Jena and Joseki open source semantic web framework. Joseki, is a SPARQL query web service and Jena provides Java APIs for manipulating RDF stores. Our view generator framework is implemented as a new Jena model assembler. We have performed certain optimizations in the view generator (pre-compilation of view specification to SPARQL templates with backend query optimizations) to significantly decrease the view generation time. Further, only updates to RDF store triggers a view update.

To demonstrate a practical application of our ideas, we chose the wide-area network monitoring system described in [1]. The primary data used by this application is traceroute data periodically collected from about a hundred nodes of PlanetLab. We developed an ontology for the traceroute data and wrote an adapter to convert traceroute instances into RDF using the ontology. The data store has many routes for each source-destination pair since traceroutes are periodically rerun. Figure 1 shows a view specification that defines a snapshot of all latest routes containing a specified IP address. The Class myRoutes would refer to all instances of routes that pass through the specified IP address and the class AllLatest ensures that only the latest routes between a specific source and destination nodes are considered. It also states that all routes having the same source and destination nodes should belong to an equivalence set so that the latest route from each set will form the required snapshot.

The performance of our views framework was measured using the most common query in the application called the Netvigator Query[1] that estimates time delays between source-destination node pairs using their distances to common intermediate nodes. Clearly, the Netvigator query makes sense only on the latest routes recorded between all relevant source-destination pairs.

Figure 2 shows query response times (y-axis) for the Netvigator query over varying sizes of the base model (x-axis). Performing the query directly on the entire database ('db') is very slow, taking **167** seconds for the largest dataset. Loading the entire dataset into memory and then performing the query ('mm') takes 24 seconds for the largest model – but clearly is not feasible for larger data sizes. When using our temporal views, the first query requires materializing the view from the database, but subsequent queries significantly benefit from the view as they can operate directly on the in-memory view taking only **0.67** seconds ('view'). Notice that the time to query the view is more or less independent of the base data size as the size of the view remains constant. We also note that it is faster to generate a view and then query it ('viewgen+qry') than it is to perform the query directly on the database (**92** sec vs 167 sec). Using RDF and SPARQL also reduces the application complexity. An equivalent effort with relational databases using fixed schema required a complex 50 line SQL statement versus a 6 triple SPARQL query!

```
<view:View rdf:ID="latestRoutesThruMyServer">
    <view:SelectInd rdf:resource="#AllLatest" />
    <view:SelectInd rdf:resource="#myRoutes" />
    <view:SelectAll />
</view:View>

<view:TemporalClass rdf:about="#AllLatest">
    <view:Latest />
    <view:Time rdf:resource="#dateAndTime" />
    <view:Base rdf:resource="#Route" />
    <view:Distinct rdf:resource="#srcHost" />
    <view:Distinct rdf:resource="#destHost" />
</view:TemporalClass>

<owl:Class rdf:ID="myRoutes">
  <owl:intersectionOf rdf:type="Collection">
    <owl:Class rdf:about="#Route">
    <owl:Restriction> <owl:onProperty rdf:resource="#viaNode"/>
      <owl:hasValue rdf:resource="15.76.98.110/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Figure 1: An Example View Specification

4. CONCLUSIONS

We have proposed a temporal view mechanism that reduces application complexity and enables faster access to time-varying information. The views are expressed using an OWL-based specification language which enables developers to select items of interest using semantics of the domain. Introducing the concept of notional equivalence classes simplified the specification of temporal snapshots. For a real-world networking application, we found that as the dataset size increased it was faster to generate and query the view than to directly query the database. We also note the reduced query complexity due to use of SPARQL over SQL.

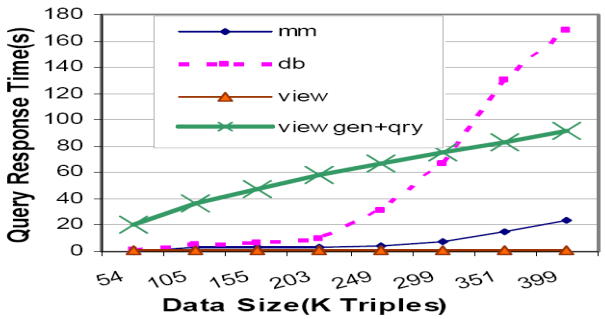

Figure 2: Query Performance over RDF

5. ACKNOWLEDGMENTS

We thank Sujata Banerjee, Puneet Sharma, Dave Reynolds, Kevin Wilkinson from HPLabs for their help in this project. We also thank interns Akshay & Srihari from BITS for their SQL work.

6. REFERENCES

[1] *"Estimating Network Proximity and Latency," Puneet Sharma, Zhichen Xu, Sujata Banerjee et al, ACM Sigcomm, July 2006*

[2] *RDF Aggregate Queries and Views*, Edward Hung,Yu Deng, V.S.Subrahmanian, *ICDE'05* , pp. 717-728, 2005.

[3] *Challenges for Data Mining in Distributed Sensor Networks,* Virginio Cantoni, Luca Lombardi, Paolo Lombardi

[4] *Implementing Views for Controlled Access to the Semantic Web,* Geetha Manjunath, Craig Sayers, et al, SWeCKa 2007

[5] *Views for light-weight web ontologies*, Raphael Volz, Daniel Oberle, Rudi Studer, SAC2003

[6] *PlanetLab*, http://www.planet-lab.org