A Management and Performance Framework for Semantic Web Servers

M. Mesarina, Venugopal K.S., N. Lyons, C. Sayers
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2007-54
March 30, 2007*

management, performance

The unification of Semantic Web query languages under the SPARQL standard and the development of commercial-quality implementations are encouraging industries to use semantic technologies for managing information. Current implementations, however, lack the performance monitoring and management services that the industry expects. In this paper, we present a performance and management framework interface to a generic SPARQL web server. We leverage existing standards for instrumentation to make the system ready-to-manage through existing monitoring applications, and we provide a performance framework which has the distinct feature of providing measurement results through the same SPARQL interface used to query data, eliminating the need for special interfaces.

* Internal Accession Date Only
Published in the 16th International World Wide Web Conference, 8-12 May 2007, Banff, Alberta, Canada
Approved for External Publication

A Management and Performance Framework for Semantic Web Servers

M. Mesarina, Venugopal K.S, N. Lyons, and C. Sayers
Hewlett-Packard
Palo Alto, CA, USA
malena.mesarina@hp.com

ABSTRACT

The unification of Semantic Web query languages under the SPARQL standard and the development of commercial-quality implementations are encouraging industries to use semantic technologies for managing information. Current implementations, however, lack the performance monitoring and management services that the industry expects. In this paper, we present a performance and management framework interface to a generic SPARQL web server. We leverage existing standards for instrumentation to make the system ready-to-manage through existing monitoring applications, and we provide a performance framework which has the distinct feature of providing measurement results through the same SPARQL interface used to query data, eliminating the need for special interfaces.

Categories and Subject Descriptors

RDF, SPARQL, Semantic Server, Semantic Web, JMX

General Terms: Management, Measurement, Performance.

Keywords: Management, Performance.

1. INTRODUCTION

The vision of the Semantic Web as a universal medium for knowledge sharing and autonomous transactions between machines has spurred increasing research activity in the last five years, resulting in specifications such as the W3C standards for describing resources (RDF), constructing ontologies (OWL), and specifying queries (SPARQL) [2]. Now that implementations of these standards are readily available, businesses are starting to explore semantic technologies to improve the efficiency of internal enterprise operations, provide new services, and invent new applications. For example, we are currently exploring the capabilities of semantic technologies in the areas of sensor data collection and information management. In our research, as we started using an implementation of a SPARQL server, we stumbled into performance problems. We could not diagnose if our problem was with the query, the network, the remote server, or the size of the store. Just as we faced these problems, so will other end users. Recognizing that wide corporate adoption will require solutions for management and performance we have focused on building such tools.

Semantic SPARQL servers are different than traditional web servers, they are designed to service queries in a semantic web language (called SPARQL) to specific semantic stores (called models) stored in memory or persistent storage. The underlying mechanism is a semantic layer stack through which queries are decomposed, augmented with inference engines, and processed. In this type of system, performance concerns typically revolve around the 'query' as opposed to higher level web server tasks. The types of state information and performance metrics that a user would want will be related to the semantic query and model rather then the higher level metrics of the server or lower-level details of the underlying database. For example, a typical request for state information would be for RDF model names and for the number of RDF triples in the models, rather than for database tables or the number of rows. And a typical request for performance would be query response time rather than servlet processing time.

Our approach to management control consists of utilizing a widely adopted standard for software instrumentation (JMX) to leverage existing implementations of monitoring applications. And our approach to performance monitoring consists of leveraging the underlying semantic technology to store and provide performance measurements as RDF, providing a homogeneous interface to both data and performance data to the user. This eliminates the need for the user to understand the server environment or knowledge of a specific language to retrieve performance information.

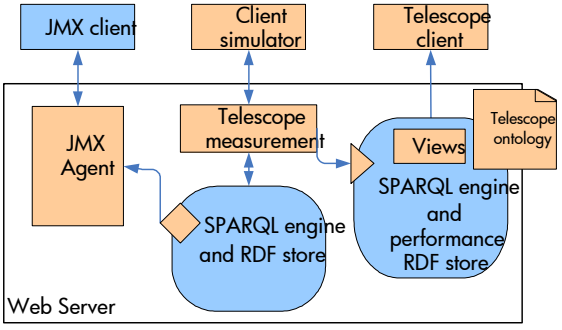

Figure 1 - Architecture for our management and monitoring framework.

2. ARCHITECTURE

Our implementation consists of a management component, a performance framework and an additional query simulator for testing. Figure 1 shows how these components fit together. The JMX Agent and instrumentation of the SPARQL engine form the management control, while the Telescope measurement and Telescope client form the performance framework. For our implementation we used the open-source Joseki SPARQL server and will refer to it in the coming descriptions [2].

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

2.1 MANAGEMENT

Some of the basic operations that a SPARQL server manager would like to perform include starting and stopping the server, monitoring model sizes, and observing logging and debugging events. We used JMX, which is an industry standard specification for embedding management controls in Java based applications. JMX clients connect to the JMX Agent to monitor and control the SPARQL engine. In the case of Joseki, we rely on the built-in support for JMX in the Jetty servlet engine to control the server and supplement that with our own Joseki agent and dynamic MBeans that instrument internal component of the Joseki server, such as the Configuration, Logging, Jena, and Debug objects. The Agent also utilizes JMX monitoring services to periodically query the state of objects, and create trend models for later analysis. Any standard JMX client, such as JConsole, or the MX4J http client can then manage the server.

While the JMX framework provides low-level operational control to a server administrator, there is also a need to provide performance monitoring information to SPARQL clients. This is provided via our Telescope toolset.

2.2 PERFORMANCE MONITORING

To monitor performance of the SPARQL server when servicing queries we instrument it with a system we term 'Telescope'.

The Telescope measurement framework (see Figure 1) is built with a servlet filter, which intercepts a client's request, extracts client and service information, times the query response and stores all this information in a performance RDF store before forwarding the response to the client. This performance information is stored in an ontology for performance measurements which we created. Because servlet filters are detachable components, our Telescope filter is conveniently independent of the underlying server system.

Any client application can then query for performance information by merely issuing an additional query to the performance store.

We have also developed an AJAX style performance visualization tool, the Telescope client, which displays charts with performance results. The performance results are categorized and displayed by query types to make it easier to distinguish performance of different queries.

Figure 2 shows a screenshot of this tool. The top chart shows query response times vs date, and the bottom chart shows dataset size vs date, thus it is easy to correlate the trend of response times as the dataset size increases.

2.3 CLIENT SIMULATOR

The final element in our framework is a client simulator for testing and profiling the end-to-end performance of the SPARQL server under various query conditions. This tool simulates an end user making multiple queries and records response times. It allows a user to enter a list of SPARQL queries, the desired time delays

between queries, and the number of times the query list should be cycled. We found this simulator very useful not only for testing and profiling but also for closing the network performance gap, as it provides statistical query response times at the client. We have developed working implementations of the entire framework.

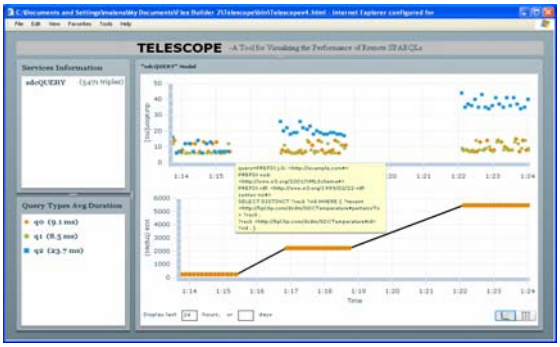

Figure 2- Telescope performance visualization tool

3. PRIOR WORK

Oracle Spatial 10g, currently supports SPARQL like queries embedded in SQL commands. The Oracle Application Server provides standard server performance metrics but it does not provide performance information for native RDF operations, like the query response times for varying service workloads and data set sizes that we provide. In addition, our system allows a query writer to analyze performance by simply issuing another query. To the best of our knowledge, there are no SPARQL server systems that provide the performance and management functions we provide.

4. NEXT STEPS AND CONCLUSIONS

In the current implementation, performance results for all clients are available to any client. We will enhance Telescope to provide security and succinct results with the addition of a views mechanism to select the query types of interest to a particular user. And for the JMX management interface, we plan to use the JMX controls to collect internal state information (e.g. database size) and map this and other state information into another persistent model that could be queried by SPARQL clients..

5. REFERENCES

[1] M. Davis, "The Semantic Wave, Part-1: Executive Guide to Billion Dollar Markets", a Project10x Special Report, January 2006.

[2] http://www.hpl.hp.com/semweb/

[3] JMX Specification (JSR-000003)

6. ACKNOWLDEGEMENTS

Many thanks to all those who have contributed to Jena and Joseki and especially Andy Seaborne. Thanks also to our colleagues: G. Majunath, R. Badrinath, J. Recker, and T. Close for their comments and advice on this work. Any errors/omissions are the responsibility of the authors.