Evaluation of the performance of open-source RDBMS and triplestores for storing medical data over a web service

Vassilis Kilintzis, Nikolaos Beredimas, Ioanna Chouvarda *IEEE Member*

Abstract— **An integral part of a system that manages medical data is the persistent storage engine. For almost twenty five years Relational Database Management Systems(RDBMS) were considered the obvious decision, yet today new technologies have emerged that require our attention as possible alternatives. Triplestores store information in terms of RDF triples without necessarily binding to a specific predefined structural model. In this paper we present an attempt to compare the performance of Apache JENA-Fuseki and the Virtuoso Universal Server 6 triplestores with that of MySQL 5.6 RDBMS for storing and retrieving medical information that it is communicated as RDF/XML ontology instances over a RESTful web service. The results show that the performance, calculated as average time of storing and retrieving instances, is significantly better using Virtuoso Server while MySQL performed better than Fuseki.**

I. Introduction

The selection of the proper persistent storage solution is an integral part of every modern information system. For an eHealth, or coordinated care system a persistent storage solution is required in order to store and retrieve medical data from various sources.

In this respect, this selection is one of the first technical challenges investigated in the context of the recently launched WELCOME project [1]. WELCOME aims to build an innovative system that goes beyond the state of the art though: a) Integrated care encompassing socio-medical aspects and technology of monitoring and treatment of COPD patients with comorbidities of CHF, Diabetes, Anxiety and Depression, b) Technological elements, like sensing components and microelectronics that will compose system's motoring devices, and c) Cloud services and advanced content delivery. The diversity, volume of data, data access and data analysis needs, signify the importance of the cloud storage engine component.

According to the initial architectural design decisions of WELCOME, the storage server must communicate through a common interface with various modules. A patient hub module resides in the patient's house and collects data from a multi-sensor vest producing considerable amount of data (initial estimations about 2Mbit/sec) and also from other devices and questionnaires. The patient hub module must communicate with the cloud based storage which has also communication with other cloud based modules such as a decision support system or modules that handle extraction of second level features from bio-signals and bio-parameters.

The current common practice in eHealth systems is to use RDBMS to store medical data since for many years this type of systems were used and proved to be robust and efficient. Triplestores although still not widely used, present significant advantages in terms of modeling complex, semantically enriched information and can also be easily integrated to web services architectures since the communication is carried over HTTP. On the other hand exchanging information as triples (data entities composed of subject-predicate-object definitions) adds overhead compared to the exchange of database records for use in an RDBMS. Therefore, the capability to handle large amount of data and system performance are critical issues that must be addressed by a triplestore in order to be selected as a viable alternative to RDBMS.

In the past, triplestores have been characterized by poor read/write performance [2]. Triple stores show their competitive edge when used to store/retrieve RDF/XML graphs [3]. However, most benchmarks available focus on the performance of triple stores when loading in memory serialised RDF from static files [4][5]. Although this provides an insight to the engineering skill of the developers of each system, it can't be used to extrapolate performance in other scenarios. This work focuses in testing the performance of a basic RESTful web service that stores and retrieves numeric and text data about a patient exchanged in RDF/XML format using three different setups. The three setups tested were PHP/MySQL, JENA API/Fuseki triplestore and JENA API/Virtuoso triplestore.

II. Methods

A. The Data model

First a simplified patient record model was developed. The aim was to record and retrieve basic patient information (name, social security number, date of birth), cholesterol measurements (value and timestamp of each measurement) and patient answers to questionnaires (answer value and timestamp along with the related question). The entity model (concepts and properties) was created as a Protégé ontology [6]. The ontology consists of five main classes and the description of the relations between them as object properties. The five classes are: "Patient" whose instances correspond to basic patient information, "BiologicalProperty" whose instances define the possible biological parameters to be measured, "Question" whose instances describe the questions for which the system can store answers and the classes "Answer" and "BiologicalPropertyValue" for storing

The research leading to these results has been partially funded from the FP -ICT Programme under Grant Agreement no 611223 - WELCOME.(http://www.welcome-project.eu/).

V. Kilintzis, N. Beredimas and I. Chouvarda are with the Laboratory of Medical Informatics, Medical School, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece (phone: +30-2310-999247; fax: +30-2310-999263; e-mail: {billyk@med.auth.gr, beredim@auth.gr, ioanna@med.auth.gr}.

the actual answers and measurements for each patient as their instances. The idea is to retrieve the instances as RDF triples from the triplestore or the RDBMS via the web service and populate the ontology. The result of this procedure is information that is semantically enriched, self described and can be presented inside Protégé. An example is presented in Figure 1.

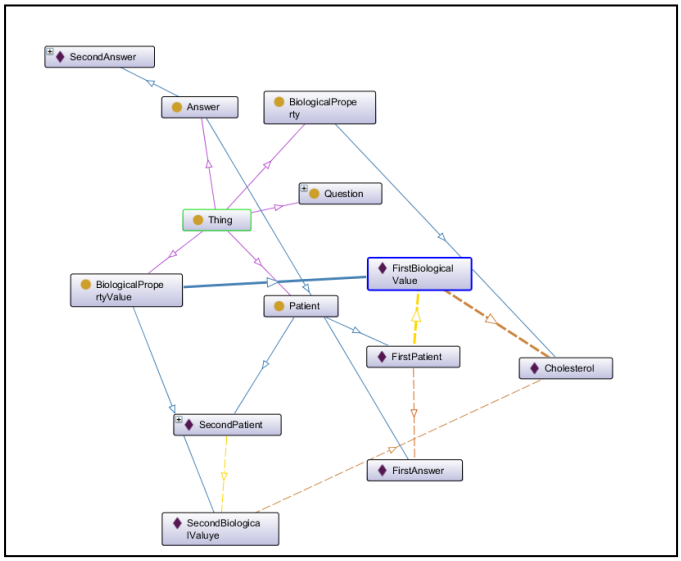

Figure 1. Ontology presenting classes and sample individuals.

B. *The three storage setups*

A REST service API was designed to allow read/write access to the patient record system using RDF/XML. Three systems were implemented as depicted in Figure 2.

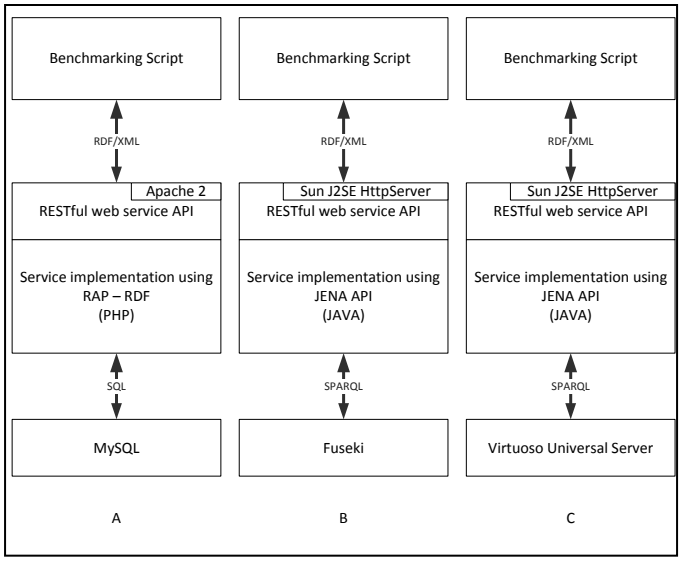

Figure 2. Test systems' design diagrams.

The first setup (Figure 2.A) consisted of the classic solution of an SQL backend (MySQL 5.6 Community edition), along with the API implemented in PHP and running on an Apache web server. The database schema was designed to be able to store multiple data types and various entities in order to be scalable and easily adaptable in various domains and it was not tightly bounded to the data model of the benchmark, in order to simulate the implementation of a real system. The PHP web service used the open-source RAP - RDF API [7] library in order to handle the RDF/XML messages. This system offered a baseline-like measure of performance.

In the second (Figure 2.B) setup the API was implemented on Java using the Jena API [8]. Jena was selected partly because it is RDF-centric and not OWL-centric like the OWL API [9]. This would guarantee that any performance observations would be a result of the triple store characteristics and not skewed by performance shortcomings when converting OWL axioms to triples. In addition, being an Apache project the Jena API offers a big support community making it easier to use for the purpose of rapid testing. The back end server used on the second system was Fuseki [10]. Fuseki is a SPARQL server being developed alongside Jena and following the SPARQL 1.1 recommendation [11].

For the third setup (Figure 2.C), the Virtuoso [12] Universal Server 6 Open Source Edition was used. Being praised for its performance in RDF loading speed in other benchmarks [13], it offered the best possible blind choice to test the current state of triple stores. Due to developer limitations we were forced to use Virtuoso version 6, although we reasonably expect the current version 7 to offer equivalent if not better performance. Source code changes in the API from the second to the third setup were minimal, consisting only of editing the SPARQL endpoint paths and adding an HTTP authentication routine for Virtuoso.

C. *The benchmark*

To benchmark the three setups we implemented a single benchmarking script that executed HTTP requests to the RESTful API in order to test store and retrieve performance. The benchmarking script was the same for each test apart from the path to the web service that was changed corresponding to the setup being benchmarked. Those requests created in sequence a total of 500 patient records. For each record, 4 cholesterol measurements and 4 answers were added. Each record creation, measurement addition and answer addition were performed in distinct individual requests, for a total of 4500 requests. Next, the benchmarking script was used to read the data from each system through specific HTTP requests. First the cholesterol measurements were requested (500 requests), then the answers (500 requests), then the complete patient records (500 requests). All tests were handled on the same hardware (laptop with Intel T3400 CPU, running Windows 7 32-bit) serving both as server and client (no network latency).

III. RESULTS

No significant variation was observed in the execution time of successive same type requests in each of the three systems. In order to be complete, in the results we present the average response time along with the standard deviation (SD) of each setup for all request types.

The average write performance measured in milliseconds for each type of request in each one of the three systems is presented in Figure 3. The Fuseki backend implementation is the slowest of the three with the mean (SD) time being 763

(204)ms, 768 (143)ms and 767 (175)ms for the three request types (Patient info, Cholesterol measurement, Question Answer). The other two performed better. The Virtuoso based implementation was significantly faster in all request types with average response time 15 (7)ms, 14 (3)ms and 12 (2)ms for the three request types compared to MySQL based setup that had 211 (107)ms for patient info, 137 (80)ms for cholesterol measurement and 56 (5)ms for answer to questions.

Figure 3. Average write response time per solution and request type.

Regarding the read performance, the response times are comparable except for the request of complete record in the Fuseki based implementation where a severe performance drop was noticed. The results of the average read request response time are shown in Figure 4. The Virtuoso based system averaged response times of 27 (4)ms for cholesterol measurement read request, 26 (2)ms for question answer request and 46 (4)ms for complete patient record request. The RDBMS based system had average response time of 56 (6)ms, 58 (5)ms and 36 (5)ms respectively. Finally Fuseki had average response times of 28 (8)ms, 24 (3)ms and 539 (147)ms for the three read request types.

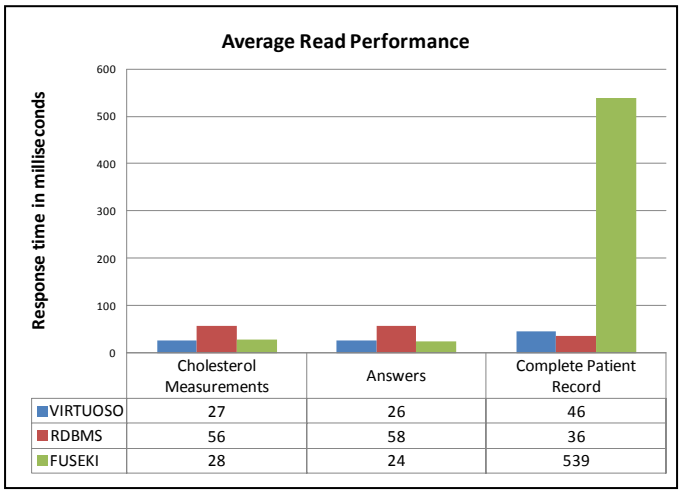

Figure 4. Average read response time per solution and request type.

IV. Discussion

Selecting a storage engine is a critical task in the development process of a medical data management system. The purpose of the proposed benchmark was to assess using a realistic scenario the performance of the complete process of storing and retrieving data through a web service API which is different from most already existing benchmarks that assess the performance of just the storage layer through bulk import export of data from and to the file system.

The results of our work show that triplestores can be used as an alternative to the widely accepted RDBMS based storage solutions. The current generation of triplestores appears to be ready, at least performance-wise, for developing production-grade systems.

The Virtuoso based implementation performed better than the classic MySQL solution in all but one of the test cases.

The Fuseki based implementation suffered from write performance issues, when comparing to either the Virtoso or the MySQL based solutions. Even taking into account the hardware used to evaluate its performance, it is too slow to be considered as an alternative for any production-grade system.

A major advantage that was obvious while developing the testing systems is that the use of triplestores with SPARQL endpoint are easy to interchange since the SPARQL standard is clearer and is followed strictly by the triplestore endpoints. On the other hand different makers of SQL capable systems do not perfectly adhere to the standard, for instance by adding extensions, and the standard itself is sometimes ambiguous. Additionally the use of connectors is not required as SPARQL endpoints operate over HTTP while the use of an RDBMS requires a frontend language specific-RDBMS specific connector library in order to communicate.

Selecting a triplestore to store medical data instances along with an ontology to describe the model, as we did in our case, enables the use of a flexible schema that can be expanded without the need of redesign in the storage level. In the implementation of the PHP web service the model was inevitably described once in PHP code in order to map RDF into proper MySQL queries and also in the MySQL database schema in order to identify stored data. This procedure has obvious implications on the maintenance and flexibility of a complete system.

Still, more issues have to be studied before one chooses to switch to triplestores for persistent storage of medical data. One issue is that of integrity constraints. RDF works under the Open World Assumption [14] and alone offers no validation of constraints. Apart from the obvious solution of implementing this functionality on the application layer, there is no consensus of how to implement it on the storage layer, for example either by extending the RDF standard or semantically identifying constraint violations using SWRL or SPIN.

Another key issue the authors have identified, that needs further study is that of security. One can reasonably expect that as triple stores gain ground in production systems new types of attacks against these systems will be devised. Still, even current attack types analogous to the attacks observed

on relational database systems, like SPARQL injection [15], are not currently well understood by developers.

Future plans for this work include benchmarking in more complex scenarios. These involve enhancing both the ontology and the storage system. The main changes should allow the handling of binary data, such as biosignals originating from wearable biosensors. Then the system's behavior will be tested for requests involving large amounts of data. The assessment of the results of those tests will point out which is the optimal solution for the storage of WELCOME project's data.

REFERENCES

[1] www.welcome-project.eu

[2] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. "Scalable semantic web data management using vertical partitioning", in *Proceedings of the 33rd international conference on Very large data bases '07*, pp. 411-422.

[3] RDF 1.1 Primer, http://www.w3.org/TR/rdf11-primer/

[4] C. Bizer, A. Schultz, "The berlin sparql benchmark." , *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5.2, 2009, pp. 1-24.

[5] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems." *Web Semantics: Science, Services and Agents on the World Wide Web, vol.* 3.2, 2005, pp. 158-182.

[6] Protégé, http://protege.stanford.edu

[7] RAP - RDF API for PHP, http://wifo5-03.informatik.uni-mannheim.de/bizer/rdfapi/

[8] Apache Jena, https://jena.apache.org/

[9] M. Horridge, and S. Bechhofer, "The owl api: A java api for owl ontologies.", *Semantic Web, vol.* 2.1, 2011, pp. 11-21.

[10] Fuseki, http://jena.apache.org/documentation/serving_data/index.html

[11] SPARQL 1.1 Overview, http://www.w3.org/TR/sparql11-overview/

[12] O. Erling, and I. Mikhailov. "RDF Support in the Virtuoso DBMS." *Networked Knowledge-Networked Media*. Springer Berlin Heidelberg, 2009, pp. 7-24.

[13] BSBM V3.1 Results (April 2013), http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V7/index.html

[14] S. Abiteboul, R. Hull, and V Vianu. *Foundations of databases*. Vol. 8. Reading: Addison-Wesley, 1995.

[15] P. Orduña, et al. "Identifying Security Issues in the Semantic Web: Injection attacks in the Semantic Query Languages." *Actas de las {VI} Jornadas Científico-Técnicas en Servicios Web y {SOA}}*, vol. 51, pp. 4529-4542.