A Framework for Performance Study of Semantic Databases

Xianwei Shen[1] and Vincent Huang[2]

[1] School of Information and Communication Technology,
KTH- Royal Institute of Technology, Kista, Sweden
[2] Services and Software,
Ericsson AB, Kista, Sweden

Abstract. Presently, the Semantic Web is gaining popularity. Consequently, there is an increased demand for benchmarking of the performance of RDF stores and SPARQL queries with simulated test data. However, there is not sufficient work on how different RDF stores perform with various SPARQL elements. The work presented in this paper seeks to explore the performance issues of data storage, data retrieving and data inference. We present a framework for performance study of semantic databases. We used this framework to evaluate several existing semantic databases. The evaluation results show that our framework can facilitate the performance study and there is room for improving the performance of current semantic databases.

Keywords: Semantic Web; RDF; SPARQL; database; perform study; evaluation

1 Introduction

The Semantic Web is an extension of the existing Web [2]. It evolves step by step leveraging structured data technologies like XML (eXtensible Markup Language), and remote service technologies as well as decades of research and development of artificial intelligence. The goal of the Semantic Web is to give data a meaning, i.e. making data readable for machines. Ontologies build axiom systems, and every data imports such ontology should conform to such axiom systems which are very hard to build. Therefore, ontologies are normally built by experts which is also a bottleneck for the development of the Semantic Web. The Semantic Web is the most promising embodiment of artificial intelligence. It brings huge benefits for integrating, organizing, indexing and searching information accurately and effectively. In a not so far future, the Semantic Web technologies will substantially improve our Web experience.

The higher layer of the Semantic Web architecture has attracted a lot of attention, while the backend of the RDF (Resource Description Framework) data store suffers from performance issues with the currently existing databases which are not originally designed for storing RDF data. There is currently not so much

effort on the performance aspects of the Semantic Web. The performance studies usually lack of comparison of various RDF data stores with specific storage, inference and query methods. In fact, the performance is a vital drawback of the Semantic web. The goal of this work is to design and implement a data independent standalone developer-centric tool to provide evaluation of performance and optimization/tuning during the development process.

The structure of this paper is the following: In Section 2, we give a brief overview of related work in the performance study area. In Section 3, we present in detail our performance study framework. In Section 4, we evaluate some existing semantic databases using our proposed framework. And finally, we give some concluding remarks in Section 5.

2 Related work

Jena Framework [1] is the most popular Java implementation for Semantic Web. Jena maintains RDF triples in memory with a hash map-based data structure. Jena also keeps triples in a so-called GraphTripleStore which consists of three hash maps named as Subjects, Properties, and Objects respectively. In Jena, each triple will be stored three times in GraphTripleStore, because each field can be treated as key that such triple will be put into Subjects, Properties and Objects with subject, property and object as key respectively. This design trades off more memory space for finding a triple quicker.

There are also a number of benchmarks developed for studying the performance of semantic web. Each benchmark implements its own framework. LUBM stands for Lehigh University Benchmark [5] which is a *de facto* benchmark for Semantic Web technology vendors presently. It is easy for generating and customizing data. It partially supports OWL (Web Ontology Language) Lite and assumes a university case with generated data of user-defined number of university, department, student, etc. LUBM provides 14 non-optional queries according to the published university ontology. UOBM(University Ontology Benchmark) is an extension of LUBM which embraces better inference capability support, i.e. partial of OWL DL. It provides also better scalability support, because they add more interlinks between the isolated university assumption in LUBM [3]. SP^2Bench(SPARQL Performance Benchmark) [4] uses a real dataset of DBLP Computer Science Bibliography. It considers RDF layouts and RDF data management approaches.

From these benchmarks, we found that they are only valuable if the target domain being evaluated is similar with the benchmark's simulated dataset, otherwise it is not accurate. In the view of development process, these benchmarks focus on the production process in the sense of they are intended to help vendors to enhance their products and encourage application developers to choose the appropriate products, rather than assisting developers to optimize or tune the performance of products.

[1] http://jena.sourceforge.net/

3 Evaluation Framework

In this work, we designed and implemented a novel evaluation framework to explore the performance issue precisely and comprehensively. According to the extracted requirements introduced below, a highly customizable architecture of this evaluation framework is designed and implemented.

3.1 Requirements

Each requirement item is denoted in the form of EFR-Number, where EFR stands for Evaluation Framework Requirement. All requirements are illustrated in Table 1 below.

EFR ID	Requirements
EFR-1	**Generic architecture**
	1. It should be easy to integrate different data stores and inference engines. 2. Data independent.
EFR-2	**Diverse SPARQL queries**
	1. Fine-grained query design with specific granularity, e.g. different combinations of SPARQL elements
EFR-3	**Comprehensive evaluation**
	1. A large set of test cases covers various aspects of SPARQL and data store layouts
EFR-4	**Cost-aware monitor**
	1. Time-aware 2. Memory-aware
EFR-5	**Smart & Trustable Logger**
	1. Apache Log4j-like logger, but with addition service as keeping track of log files 2. Evaluated results are able to be plotted automatically
EFR-6	**Backward compatible testing**
	1. Adaptive to frequently changing versions

Table 1. Requirements for designing evaluation framework

3.2 Architecture

According to the previous analysis, we propose an architecture as in Figure 1. Note that the gray colored components are existing framework or software incorporated into this architecture, and the white colored components are our contributions. Presently, the framework is only available for Jena's API, which means that all the components being integrated should provide interfaces to Jena. This is also a restriction in designing this framework.

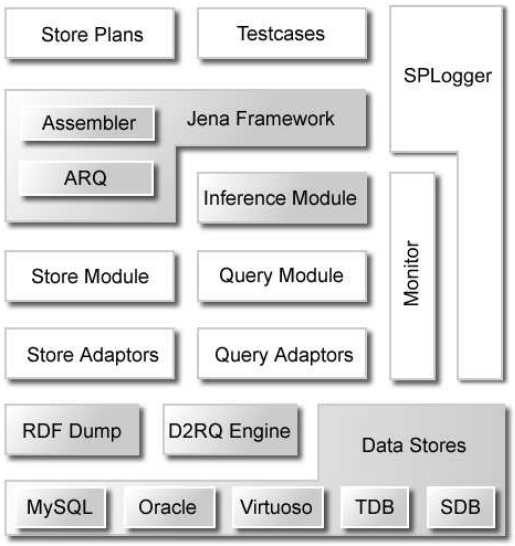

Fig. 1. Architecture of evaluation framework

The lowest layer mainly focuses on the maintenance of data. All the component names are written in italic style. The component *RDF Dump* provides the plain RDF file in RDF/XML, N3, Turtle, etc. While the *D2RQ Engine* above the *Data Stores* can convert RDBMS data into RDF data in runtime, and translate SPARQL to SQL (Structured Query Language) on the fly. So we say these two components answer the queries, i.e. either from the plain text or converted from relational data. The *Data Stores* component consists of a set of data stores, e.g. five data stores are used, MySQl, Oracle 11g, Virtuoso, Jena TDB and SDB. Each data store may have several storage layouts.

The adaptor layer adapts different interfaces to general interfaces for storing and querying. The component *Store Adaptors* adapts different storage requests into a general format, which is specified in the *Store Module* component. The component *Query Adaptors* converts various query implementations into a general interface, and process query in a uniform style. The two kernel modules *Store Module* and *Query Module* are responsible for data storage and query handling. The *Store Module* component defines a general style to store data according to various storage plans. The component *Query Module* integrates different query forms and query execution implementations into a general format.

Jena Framework with *Inference Module* provides all necessary API for RDF models and SPARQL specification. The *Assembler* is used to load RDF data or configurations. All the configurations are also RDF models, the ontology used is predefined in Jena. ARQ is also integrated in Jena Framework, it allows querying with SPARQL. Jena Framework is the underlying package we used through

this work as well. The *Inference Module* component provides the capability of reasoning over RDF triples with rules, e.g. Jena rules engine, Pellet, PelletDb, Oracle rules and Virtuoso rules.

The top layer defines a scripting system which allows user to define storage plans or test cases arbitrarily without interfering with the underlying components. The *Store Plans* component specifies various layouts for different data stores as well as data sources. The component *Testcases* allows scripting work for various queries against diverse store plans.

The *Monitor* and *SPLogger* are two components for keeping track of running status and evaluation results of the evaluation framework. The component *Monitor* keeps track of the time consumption, memory cost as well as the data statistics. *SPLogger* is an Apache Log4j-like logger, which is used for recoding the produced results by the evaluation framework into Microsoft Excel, and plot them using MATLAB.

3.3 Implementation

D2RQ Engine [1] is modified to adapt to our evaluation framework so that it can map RDB to RDF and to establish a bridge between SQL and SPARQL. Note that we use manually mapping scheme, i.e. design a mapping file according to D2RQ specification rather than automatically generated the mapping file, because the latter one may lose semantics we need.

The *Store Module* makes the framework independent of either specific non-RDF store or RDF store, i.e. add or remove a data store is simply an update of scripts from *Store Plans*.

The *Store Adaptors* component consists of four types, they are Oracle Store Adaptor(OSA), TDB Store Adaptor(TSA), SDB Store Adaptor(SSA) and Virtuoso Store Adaptor(VSA). OSA loads RDF model with an assemble file fed into *Store Module* and store it in Oracle 11g with JenaAdaptor 2 (released with Oracle's semantic web technology softwares). TSA executes in the same way but stores RDF model in TDB with triple or quad layouts. SSA stores model with layout1, layout2, layout2/hash, layout2/index schemes. VSA follows triple and quad layouts as TDB does.

The *Query Module* provides a fair and uniform environment for evaluating queries. Because data stores may have their own interfaces for executing queries, we have to hide these differences and let them conform to the same specification for evaluation, both before and after the query processing. It consists of three subcomponents, they are Load Module, Dispatch Module and Query Agent respectively. The Load Module is responsible for loading test cases, establishing connections, loading ABox and TBox or other scripts. The Dispatch Module seeks to feed Query Agent with SPARQL queries for evaluation. The Query Agent takes responsibility of query evaluation. It supports two paradigms, both sequentially and concurrently. In sequential evaluation, the adjacent queries have no interference to each other. They are stateless. We achieve this by the steps of start service, execute query and close service. Concurrent evaluation is planned as future work.

The component *Query Adaptors* includes five kinds of adaptors so far, they are Oracle Query Adaptor(OQA), TDB Query Adaptor(TQA), Virtuoso Query Adaptor(VQA), SDB Query Adaptor(SQA) and D2RQ Query Adaptor(D2QA) respectively. OQA takes test cases from the Testcases module and queries against Oracle 11g (configured with Oracle Jena Adaptor 2). TQA takes test cases and queries against TDB. VQA takes test cases and queries which are in correspondence with Virtuoso's SPARQL syntax. D2RQ rewrites SPARQL to SQL in runtime, it retrieves data from RDB and then converts it to RDF according to D2RQ mapping language [1].

As the previous sections described, the evaluation framework satisfies the Generality criteria: it is able to generally support a variety of backends and various layouts and a wide range of queries with comprehensive query features as well. It satisfies the criteria of Modularity due to the component base design with functional cohesion and data coupling specification. And it satisfies the Automaticity criteria as well, thanks to the scripting system design and multithreaded paradigm.

The framework is still a test version. It does not support other RDF model implementations other than Jena. Moreover, remote query evaluation is not implemented.

4 Performance Evaluation

4.1 Test Environment

Table 2 shows the evaluation environment of this work.

Name	**Configuration**
Evaluation Tool	Self-developed Evaluation Framework
Data Model	Media Model and LUBM benchmark model
Data Store	Jena TDB, Jena SDB, Oracle 11g, MySQL, Virtuoso
Inference Engine	Jena rule engine, Oracle rule engine, Virtuoso rule engine, Pellet, PelletDb
Storage Layout	TDB: triple layout & quad layout SDB: layout1, layout2, layout2/hash, layout2/index Virtuoso: triple layout & quad layout Oracle: triple layout
PC Configuration	CPU: Core2 -2.20 GHz I/O read: ˜50MB/s OS: 32-bit Vista enterprise RAID: None

Table 2. Configuration of evaluation environment

We adopt LUBM for our evaluation work. Trying to cover different dataset sizes, we choose two settings, LUBM(10) and LUBM(8000), with 10 and 8000 as

random seed respectively. LUBM(10) has approximate 1.3 Million triples, with 113MB as RDF/XML format, and LUBM(8000) has close to 1.1 Billion triples with 262GB as RDF/XML. For our own research purpose, we also studied the performance with dataset from an internal project. The data set is used for movie recommendations. It contains 29 classes, 19 object properties, 47 data type properties, and a total of 1,256,350 triples.

4.2 Evaluation Results

Figure 2 shows the comparison of time and memory consumption of loading RDF model to various data stores. The memory cost is calculated only with the consumption of JVM, other resource consumptions are not considered. Therefore, the memory cost showed here is only for a reference, rather than a formal evaluation results due to inaccuracy in some cases. For example, the Virtuoso Open edition is developed with C++, the consumption is not calculated. As the pair shows in Figure 2 that *Virtuoso Default Inferred* only costs 3454 KB. *Default* means storing with default mode, i.e. triple layout, *Inferred* means the data involves inferred graph as well. Figure 3 illustrates the time cost of loading model for SPARQL query before query execution. It is obviously that querying over plain text is impractical. Databases are rather fast and should be used.

Figure 4 shows the inference cost of combinations of different data stores and various inference engines. We find that Pellet/PelletDb outperforms Jena rule engine, Oracle with PelletDb is the fastest choice.

Figure 5 compares the original triples and the count after inference with the setting of RDFS and several SWRL rules.

We present two examples of the evaluation results of SPARQL queries. The first query is to find 100 movies. The second query is to find movies which are released in a specific time range, and without any cover. The query results are shown in Figure 6 and Figure 7.

SPARQL Example 1:

```
PREFIX mo: <http://localhost:8080/SwedPevo/ontology/Media.owl#>
SELECT ?S
WHERE
{ ?S rdf:type mo:MovieAsset .
}
LIMIT 100
```

SPARQL Example 2:

```
PREFIX  mo:   <http://localhost:8080/SwedPevo/ontology/Media.owl#>
SELECT  ?S ?D
WHERE
  { ?S  rdf:type  mo:MovieAsset ;
        mo:assetInputDate     ?D .
    FILTER ( ?D > "2009-03-03T16:00:00"^^xsd:dateTime )
    OPTIONAL
```

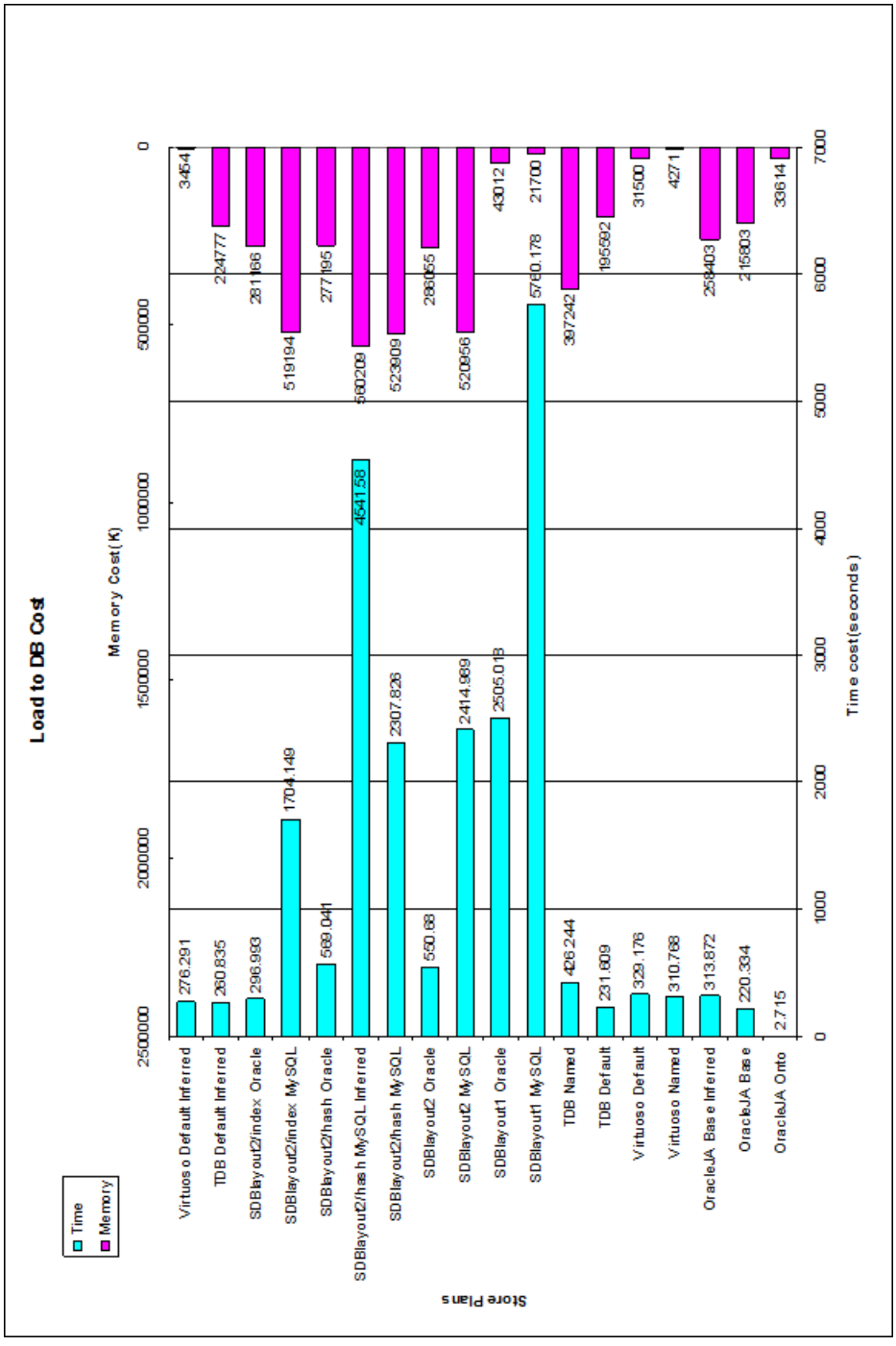

Fig. 2. Time & memory cost of loading RDF model to data stores.

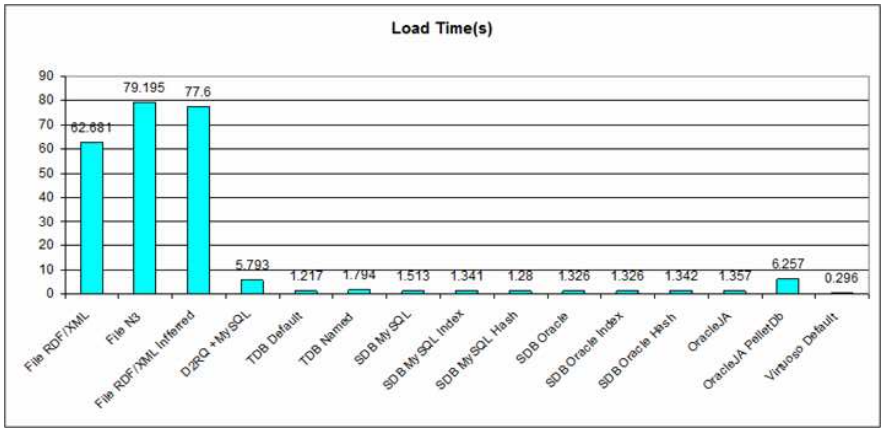

Fig. 3. Time cost of loading RDF model for SPARQL query

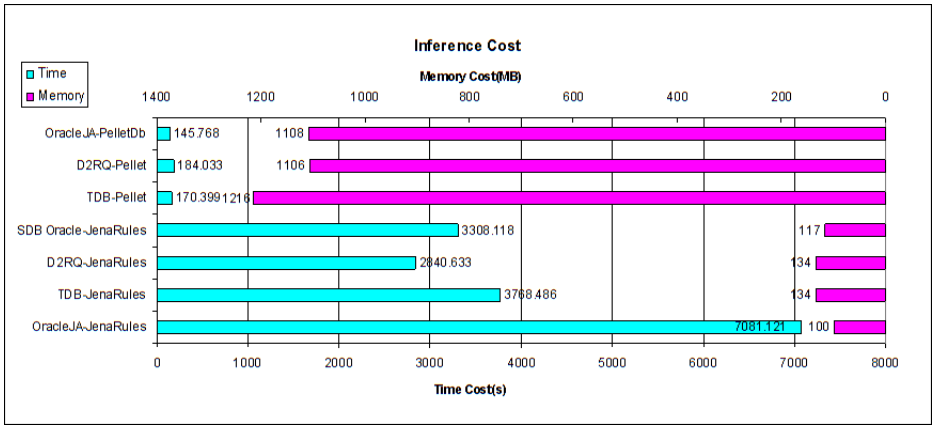

Fig. 4. Inference cost of various combinations of data store and inference engine

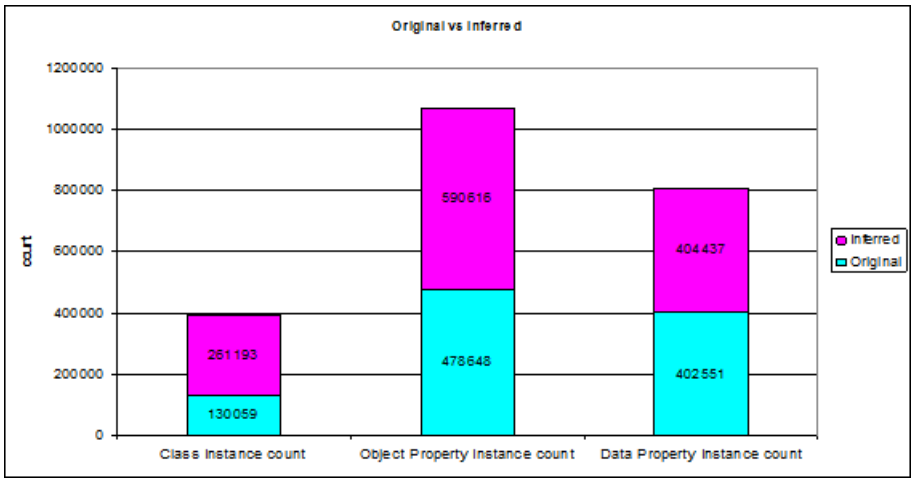

Fig. 5. Original triples vs. Inferred triples

```
      { ?S  mo:hasCover  ?C .}
    FILTER ( ! bound(?C) )
  }
ORDER BY ?s
```

4.3 Observations

From the above evaluation results, we can summarize the following observations:

1. Query with inference on the fly is not recommended, due to hours of reasoning time.
2. Databases without ACID property (Atomicity, Consistency, Isolation and Durability) are faster than the those with ACID property, e.g. TDB is the fastest one in most cases.
3. Databases have poor performance with processing patterns with OPTIONAL element and nested complex SPARQL query, due to imperfect translation from SPARQL to SQL.
4. Storage layouts have significant effects on performance.
5. The query optimizer of database does not work very well on triple store because of lack of available semantics.
6. The order of query patterns has significant effects on performance of query processing.

Note that, all the evaluation process of databases are performed without caching. The database service is restarted when a query is evaluated so that the second observation item is not affected by the first one. But with caching available, the results will not be accurate. From the evaluation results, we can

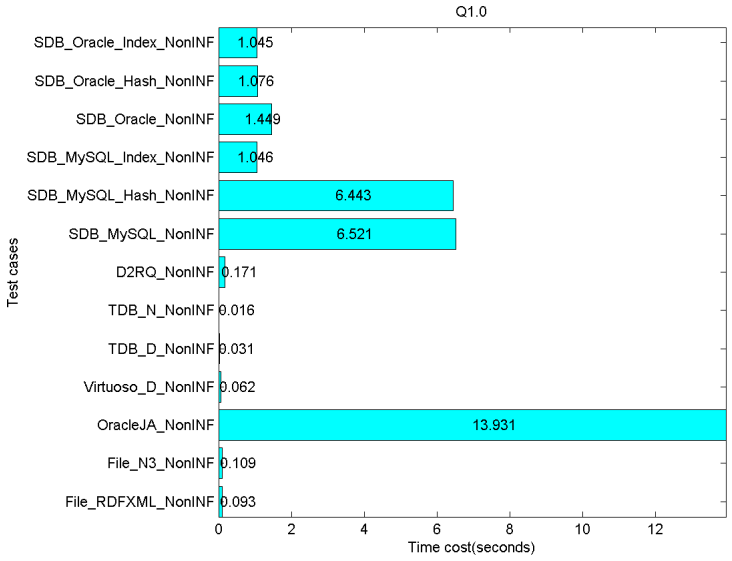

Fig. 6. Query results of SPARQL Example 1

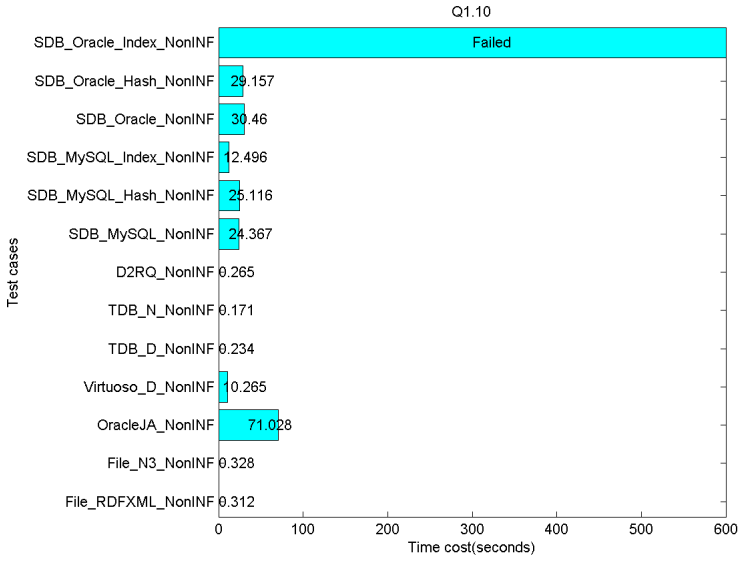

Fig. 7. Query results of SPARQL Example 2

see that TDB is the fastest RDF data store, while Virtuoso is the fastest SQL-based database. The combination of OracleJA and PelletDb outperforms other combinations.

5 Conclusion

In this paper a novel evaluation framework for performance study of semantic database is presented. Firstly, the requirements are analyzed and the framework is designed and implemented according to these requirements. Then we further discussed several components involved in the framework in detail. From the evaluation results, we can concluded that the performance of existing semantic databases for RDF storing and query processing is acceptable to a certain level, but it is still not suitable for real world usage and there is large room for improvement. To a large extent, the performance relates to how SPARQL queries are written. The performance can be improved by SPARQL query optimization.

References

1. The d2rq platform v0.6 user manual and language specification, http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/index.htm, 2009
2. Berners-Lee, T.: Semantic web road map, http://www.w3.org/DesignIssues/ Semantic.html, 1998
3. Ma, L., Yang, Y., Qiu, Z., et al: Towards a complete owl ontology benchmark. In European Semantic Web Conference (2006)
4. Schmidt, M., et al: Sp2bench: A sparql performance benchmark. In 8th International Semantic Web Conference (2009)
5. Y Guo, Z Pan, J.H.: Lubm: A benchmark for owl knowledge base systems. Journal of Web Semantics 3 (2) 158-182 (2005)