Evaluating semantic data infrastructure components for small devices

Andriy Nikolov, Ning Li, Mathieu d'Aquin, Enrico Motta

Knowledge Media Institute, The Open University, Milton Keynes, UK
{a.nikolov, n.li, m.daquin, e.motta}@open.ac.uk

Abstract. Existing benchmarks for semantic data management tools primarily focus on evaluating the capability of tools to process large-scale data. In use case scenarios involving small-scale and mobile devices, the data volumes are typically smaller, however, the ability to operate with limited computational resources becomes important. In this paper, we describe the experiments we performed to evaluate components of semantic data infrastructure for devices with limited computational power.

1 Introduction

Performance is an important criterion for selection of semantic infrastructure components such as data stores, rule engines, and query engines. So far, the development effort has primarily focused on large-scale data processing scenarios. In such scenarios, the tools are assumed to have access to considerable computational resources (powerful servers and even clusters of machines) and are expected to process large volumes of data. Because of this, evaluation benchmarking tests (e.g., the Berlin SPARQL benchmark [1] for triple stores or OpenRuleBench [2] for rule engines) are usually conducted on powerful machines and involve large datasets. With the widespread use of small-scale and mobile devices, new scenarios appear in which computational resource limitations become important. For example, in the scope of the SmartProducts project[1], the focus is on "smart products" (e.g., cars and kitchen appliances), which are able to communicate with the user and between themselves and to apply embedded "proactive knowledge" to assist the users with their tasks. In this scenario, it is important to choose semantic data processing tools, which not only require less time to process data, but also require less computational resources to produce results (in particular, memory).

Semantic data processing infrastructure for smart products involves two main components: a triple store which stores RDF data and a rule engine which is able to make inferences based on these data and solve user tasks. In this paper we discuss the experiments conducted to select appropriate components to support our use case scenario. We especially focus on the particularities related to evaluating such components when targeting small, resource-limited devices.

[1] http://www.smartproducts-project.eu/

Proceedings of the International Workshop on Evaluation of Semantic Technologies (IWEST 2010). Shanghai, China. November 8, 2010.

This includes in particular assessing not only the response time of the tools, but also their average resource consumption for different amounts of data. The rest of the paper is organised in the following way. In section 2, we discuss the requirements imposed by our use case scenario and the reasons for our preliminary choice of tools as solutions. Based on these requirements, we build test datasets to perform comparative tests. Section 3 summarises our earlier experiments comparing performance of RDF triple stores on small- and medium-scale datasets. Section 4 describes the experiments we performed with two selected rule engines (Jess[2] and BaseVISor[3]) and discusses their results. Finally, section 5 discusses the directions for future work.

2 Requirements

The SmartProducts project envisages a scenario where user appliances (e.g., a cooking guide, a car computer) make use of embedded proactive knowledge in order to communicate and cooperate with humans, other products, and environment. Proactive knowledge includes knowledge about the product itself (features, functions, dependencies, e.g., oven's maximal temperature), its environment (e.g., other devices in the kitchen, food items stored in the fridge), its users (e.g., health profile), and the usage context (e.g., a cake is being prepared for a birthday party). This knowledge is formally described using ontologies, contained in a semantic data store, and utilised using formal reasoning mechanisms. The data management infrastructure must support these functionalities.

When selecting existing tools to reuse within this infrastructure, two kinds of requirements has to be taken into account:

- *Functional requirements* related to the desired capabilities of the tools.
- *Pragmatic requirements* related to the computational resources needed by the tools.

In order to support its required tasks, a smart product needs to possess several reasoning capabilities. First, in order to make use of semantic data represented in RDF, its inferencing engine must support inferencing based on ontological axioms expressed in RDFS or OWL. Second, and even more important, in order to exhibit situation awareness and react to the changes in the environment, the inference engine has to support reasoning with custom-defined rules. As a result of rule firing, not only the knowledge base can be updated with new facts, but an arbitrary action can be triggered: e.g., starting a user interaction. Thus, for our scenario we considered general-purpose rule engines (in particular, Jess and BaseVISor) rather than OWL-DL reasoners based on description logic.

The pragmatic requirements are imposed by the choice of the hardware platform. Smart products are assumed to be implemented based on devices with reasonable computational capabilities, high-speed networking, and Linux operating system support (e.g., Gumstix[4] or smartphones). The actual hardware

[2] http://www.jessrules.com/
[3] http://vistology.com/basevisor/basevisor.html
[4] http://www.gumstix.com/

parameters may vary within each device class: for instance, the newest Overo gumstix models use a 600MHz processor and 256 MB RAM. The new models of smartphones (like Xperia X10 from Sony Ericsson) can reach 1GHz CPU speed and 1 GB memory.

3 Comparison of triple stores

As possible candidates, we considered three widely used triple stores: Jena[5], Sesame[6], and Mulgara[7] (some others, targeted at large-scale data, were excluded, e.g., Virtuoso[8], 4store[9]). To compare the triple stores, we constructed a benchmark dataset consisting of small- to medium-sized ontologies retrieved from the Watson search server[10]. The sets of ontologies to be used for testing semantic tools have been built following two main requirements. First, they had to cover ontology sizes from very small (just a few triples) to medium-scale ones (hundreds of thousands of triples). These medium-scale ontologies represent the limit of what the best performing tools can handle on small devices. Second, they had to take into account that, especially at small-scale, size is not the only parameter that might affect the performance of semantic tools. It is therefore important that, within each set, the ontologies vary in size, but stay relatively homogeneous with respect to these other characteristics.

We first devised a script to build sets of ontologies from Watson grouping together ontologies having similar characteristics, therefore building homogenous sets with respect to these characteristics. The parameters employed for building these groups are the ratios properties, individuals, and classes, and the complexity of the ontological description, as expressed by the underlying description logic (e.g., $\mathcal{ALH}$). As a result of this automatic process, we obtained 99 different sets of ontologies. We then manually selected amongst these sets the ones to be used for our benchmarks, considering only the sets containing ontologies with appropriate ranges of sizes.

Table 1. Triple store test results (test set 53): Jena (TDB), Sesame (native), Mulgara

	Jena	Sesame	Mulgara
Avg. loading time/triple (ms)	1	1	2
Avg. memory/triple (KB)	1	0.16	0.43
Avg. disk space/triple (KB)	0.17	0.1	32
Avg. query time/Ktriple (ms)	232	43	1291

[5] http://jena.sourceforge.net/
[6] http://www.openrdf.org/
[7] http://www.mulgara.org/
[8] http://virtuoso.openlinksw.com/
[9] http://4store.org/
[10] http://watson.kmi.open.ac.uk/

To test the triple stores, we applied the following procedure:

1. Loading an ontology into the data store.
2. Running test queries.
3. Measuring the disk space taken by the data store.

We developed 8 test SPARQL queries applicable to a wide range of ontologies (e.g., selecting all *rdfs:label* values or selecting all properties of all instances of all classes). The metrics we used included average loading time, memory, disk space, and query response time per single triple. The results for one of our test sets containing the widest range of dataset sizes are provided in Table 1. The dataset contains 21 different ontologies in the range between 3208 and 658808 triples. In more details, our benchmark and tests for triple stores are described in [3]. Based on the tests, we could make several observations concerning the performance of the tools.

In general, Sesame was found to outperform both Jena and Mulgara for small-size datasets, although its advantage tends to decrease with the growth of the dataset size. It is interesting to note that in the large-scale benchmarking tests [1] Sesame generally performed worse than other tools. One of the causes for this was the fact that Jena and Mulgara allocate larger amounts of resources straight from the start (especially Mulgara), while an "empty Sesame" is light-weight. In other terms, the "fixed cost" associated with processing ontologies with Sesame is significantly lower than the one of Jena and Mulgara, whereas the "variable cost" appears to be higher.

This clearly demonstrates that benchmarks targeting different scales, and considering different dimensions as available resources (i.e., not only time), are crucial. Indeed, in resource-limited environment, the best tool at a given scale, may not be the right one at another scale. Here, it appears clearly that Sesame is a strong candidate to be used on a small device, due to its low fixed cost, while, for the same reason, it might become inadequate as devices get bigger.

4 Comparison of rule engines

Based on the functional requirements, we focused on the usage of rule engines implementing the Rete algorithm [4]. We considered two initial options: a general-purpose rule engine (Jess) which we could then adapt for the usage of RDF data and a rule engine designed for dealing with RDF data (BaseVISor). In the case of a general-purpose engine, the advantages were the possibilities to use both forward and backward-chaining rule processing and to define arbitrary facts (n-ary relations) as opposed to only triples. The latter feature was convenient for defining intermediate facts, which were used during reasoning and removed from the working memory afterwards. To represent RDF data, we used the standard approach initially proposed in [5]: representing each RDF triple with a Jess fact *(triple (subject ?s) (predicate ?p) (object ?o))* and implementing OWL inferencing with Jess rules. For Jess, we used two rule processing strategies: *forward-chaining* and *hybrid* (forward- and backward-chaining). With the

forward-chaining option, all data potentially relevant for the task are loaded into the working memory at the start. This option maximises the processing speed at the expense of memory usage. With the hybrid approach, only a minimal amount of data needed to trigger the reasoning are loaded into the working memory. Backward-chaining rules are used to load information from the triple store into the rule engine working memory when needed. For data loading, we used an approach similar to [6]: using calls to Java functions which executed SPARQL queries on a triple store and translated results as Jess facts. This approach allowed saving the working memory but consumed more time.

For testing, we used the recipe selection task from one of our use cases. In this task, the RDF dataset included a set of recipes (with their ingredients), a set of user preferences, and information about available ingredients. Based on this information, the task of the rule engine was to use rules to provide a ranking of recipes which could be proposed to the user. In an example scenario, we used a dataset which, after calculating the OWL-DL inferences, contained about 280000 triples. The rule base included rules which evaluated whether a recipe in the dataset satisfied a specific type of constraint imposed on one of the recipe parameters: e.g., ingredients, cooking time or nutritional value. For example, "IF the user has a preference P for ingredient X AND recipe A contains X, add a fact declaring that A satisfies X". As a result, the rule base produced a ranking of recipes based on the number of satisfied constraints. The test set of 21 rules was originally composed for Jess in the forward-chaining mode. Then, these original rules were adapted for two other cases. For the Jess hybrid mode, rules for uploading data from the triple store were added. For BaseVISor, Jess forward-chaining rules were translated into the BaseVISor native format. Auxiliary n-ary relations created during inferencing were represented by several triple facts.

Table 2. Reasoner test results: Jess and BaseVISor

	Jess (forward)	Jess (hybrid)	BaseVISor
Engine run-time (s)	81	302	4.7
Memory usage (M)	290	60	160

We tested the performance in three cases on a PC laptop with a 791MHz Intel Core 2 Duo CPU and 2GB memory. The results are shown in Table 2. It can be seen that BaseVISor outperformed Jess in the forward-chaining mode in terms of both runtime and memory consumption, apparently, because of its specific support of triple facts. The hybrid mode required substantially less memory, because unnecessary data was not loaded. However, loading data at runtime required 3 times longer than in the forward-chaining case. Although in the forward-chaining case more data had to be loaded from the start, it took less time because asserting new facts did not require checking rule activations. In general, BaseVISor was found to provide a good trade-off between time and memory cost.

5 Conclusion and future work

In this paper, we briefly overviewed two benchmarking tests realised as part of a specific scenario related to the realisation of smart products: products embedding knowledge and smart behaviours. For this reason, the major particularity of these two studies is that they were targeting small devices, i.e., hardware and software environments where strong limitations might apply on the resources available. Through these two tests, we demonstrated the importance of benchmarks specifically designed for resource-limited environments, mainly for two reasons: (i) because, in small devices, other dimensions than response time and scalability should be assessed, including for example storage space and memory and (ii) because semantic tools can achieve different levels of performance at different scale. In our tests, we showed in particular how some of the tools that would be judged best at a large scale would actually perform rather badly when working with smaller datasets, and strong hardware limitations.

More importantly, these two elements made emerge the need for a new, more flexible type of benchmarks. Indeed, almost in all cases, the selection of a given tool is not only a matter of performance, but represents a trade-off between performance and the availability of resources. For this reason, the comparison of semantic technologies should not be an absolute measure, a ranking of the tools, independent from the environment where they apply, but should rather work as guides to elaborate this trade-off for particular scenarios and environments. In other terms, benchmarks such as the ones presented here should evolve to become guidelines supporting developers in selecting the right set of tools, depending on their requirements and constraints.

6 Acknowledgements

Part of this research has been funded under the EC 7th Framework Programme, in the context of the SmartProducts project (231204).

References

1. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. IJSWS **5**(2) (2009) 1–24
2. Liang, S., Fodor, P., Wan, H., Kifer, M.: OpenRuleBench: An analysis of the performance of rule engines. In: WWW 2009, Madrid, Spain (2009) 601–610
3. d'Aquin, M., Nikolov, A., Motta, E.: How much semantic data on small devices? In: EKAW 2010, Lisbon, Portugal (2010)
4. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19** (1982) 17–37
5. Mai, J., Paslaru-Bontas, E., Lin, Z.: OWL2Jess: A transformational implementation of the OWL semantics. In: Parallel and Distributed Processing and Applications - ISPA 2005 Workshops. (2005) 599–608
6. Bak, J., Jedrzejek, C., Falkowski, M.: Usage of the Jess engine, rules and ontology to query a relational database. In: International Symposium on Rule Interchange and Applications. (2009) 216–230