KGCW2024 Challenge Report: RDFProcessingToolkit

Claus Stadler, Simon Bin

Institute for Applied Informatics (InfAI), Leipzig

Abstract

This is the report of the participation of the RDFProcessingToolkit (RPT) in the KGCW2024 Challenge at ESWC 2024. The RPT system processes RML specifications by translating them into a series of extended SPARQL CONSTRUCT queries. The necessary SPARQL extensions are provided as plugins for the Apache Jena framework. This year's challenge comprises a performance and a conformance track. For the performance track, a homogeneous environment was kindly provided by the workshop organizers in order to facilitate comparability of measurements. In this track, we mainly adapted the setup from our last year's participation. For the conformance track, we updated our system with support for the *rml-core* module of the upcoming RML revision. We also report on the issues and shortcomings we encountered as a base for future improvements.

Keywords

RML, SPARQL, RDF, Knowledge Graph, Big data, Semantic Query Optimization, Apache Spark, Challenge

1. Introduction

This is the report of the participation of the RDFProcessingToolkit[1] (RPT) in the KGCW2024 Challenge.[2] RPT is a command line toolkit that features several sub-commands for SPARQL-based processing of RDF data. RPT builds upon the Apache Jena Semantic Web framework[3] and thus leverages its SPARQL engines and SPARQL extension systems. RPT supports the translation of an RML document to a set of extended SPARQL queries which can be subsequently executed on an appropriate engine. The most essential SPARQL extension is a special SERVICE clause which is used to specify how to load a data source as a sequence of SPARQL bindings. The SPARQL extensions that are necessary to execute RML in SPARQL are all registered with Jena's plugin system. The extensions can also be added as a standalone module to any Jena-based project.[4]

Our own special purpose SPARQL engine is based on the Sansa framework.[5] This engine leverages Apache Spark[6] in order to parallelize the following two types of operations: Ingestion

KGCW'24: 5th International Workshop on Knowledge Graph Construction, May 27, 2024, Crete, GRE

cstadler@informatik.uni-leipzig.de (C. Stadler); sbin@informatik.uni-leipzig.de (S. Bin)
0000-0001-9948-6458 (C. Stadler)

[1] https://github.com/SmartDataAnalytics/RdfProcessingToolkit
[2] https://kg-construct.github.io/workshop/2024/challenge.html
[3] https://jena.apache.org/
[4] https://scaseco.github.io/jenax/jenax-arq-parent/jenax-arq-plugins-parent/README.html
[5] https://github.com/SANSA-Stack/SANSA-Stack
[6] https://spark.apache.org/

of CSV and JSON source data and execution of SPARQL algebra operations, such as JOIN and DISTINCT. The Sansa engine is not a general purpose SPARQL engine. By leveraging Spark's map-reduce processing model, it is best used for extract-transform-load (ETL) workloads, which includes RML mapping execution. RPT/Sansa [1] refers to the use of RPT with the Sansa engine.

The challenge was divided into an *Performance* and an *Conformance* part. The challenge description and output files were published on Zenodo [2]. The remainder of this report is structured as follows: In Section 2 we report on our setup for the participation in the performance challenge. In Section 3 we provide insights into how we added support for the *core* module of the upcoming revised RML specification. We conclude this report in Section 4.

2. Performance Track

In this section, we report on our setup and results for the performance track. Overall, since our last year's participation [3], there were no major updates to our system that targeted performance. However, we had a series of bug fixes and maintenance upgrades, the most significant one being the upgrade of the code base to Jena 5.

The tasks of the performance challenge were also the same as last year's. One notable addition was that the RML mapping files were also provided in the upcoming RML revision. However, we did not evaluate against those because we completed our work on the conformance part only after the performance evaluations.

A significant improvement over last year's organization was that uniform hardware (virtual machines) was kindly provided to all participants by the challenge organizers. This allowed evaluation of all participating systems on a similar hardware, so that the results are expected to be much more comparable than the year before. The specifications reported by the VM were: 4 virtual cores (Intel(R) Xeon(R) Gold 6161 CPU), 136GB VMware virtual disk, 16 GiB RAM.

The benchmark tool's implementation differed from last year's and we forked it in order to deal with the following issues:

- We needed to add support to allow for the configuration of the working directory of the docker container running our RPT tool.
- We needed to make it possible to pass Java options to our docker container, especially for setting the maximum heap memory (-Xmx).
- The benchmark tool failed to extract system information on SUSE Linux systems.

At the time of writing, the changes are tracked in our fork[7] and there is an open pull request to the benchmark tool.[8]

We employed the same two workarounds as last year: Instead of reading tabular data from an SQL database, we ingest the corresponding CSV files directly. Due to the lack of parallel ingestion of XML data in RPT/Sansa, we adjusted the affected task pipelines by adding an extra XML-to-JSON conversion step (which is counted towards the total time) and by manually changing the XPath expressions in the RML files to corresponding JSONPath ones. With this workaround, we could leverage RPT/Sansa's parallel data ingestion and produce estimated measures despite the lack of proper XML support.

[7]https://github.com/AKSW/kg-construct-challenge-tool/tree/code-on-new
[8]https://github.com/kg-construct/challenge-tool/pull/4

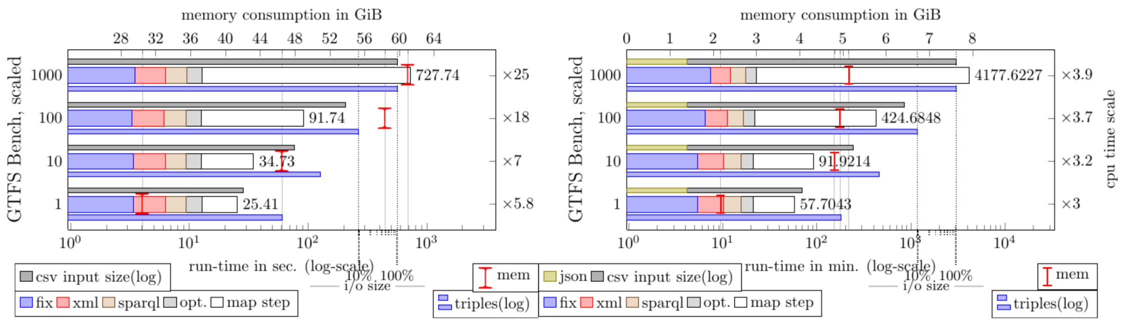

Figure 1: Comparison of results for RPT/Sansa between KGCW 2023 (left) and 2024 (right) for different scales of the GTFS-Madrid-Bench. The thick bars show the total run times of mapping task executions (the segments correspond to the different steps in the process), whereas the slim bars above and below indicate the relative byte size of the input and output data. The I symbol marks the peak RAM usage.

The performance results for RPT/Sansa are consistent with those from last year, albeit the execution times were generally higher due to the weaker hardware. The hardware specs from the year before were: 32 threads (AMD Ryzen 9 5950X 16-Core CPU), 4TB SSD, 128 GiB RAM (60GiB assigned to the JVM). Figure 1 shows a juxtaposition of this year's results for the GTFS-Madrid-Bench[4] with those of last year. The complete recent as well as past measurements for RPT/Sansa can be found in our in our evaluation results repository.[9]

Also, we incorrectly allocated only 5 GB of heap memory to our Java process, although 16 GB of RAM would have been available. However, it also shows that our system effectively leverages Apache Spark in order to run with limited resources. A noteworthy finding is related to the GTFS-Madrid-bench: The amount of RAM and disk space were insufficient to process the GTFS1000 task without compression. To get around this problem, we experimented with two approaches: (1) a hard disk volume compressed with lzo and (2) using Spark's built-in bzip2 compression. The following commands were used:

```
# Commands to create and mount a compressed filesystem image
fallocate -l 60G ./filesystem.btrfs
mkfs.btrfs ./filesystem.btrfs
mount -o loop,compress=lzo ./filesystem.btrfs /compressed-fs

# Spark's built-in compression enabled via system properties
JAVA_OPTS="-Dspark.hadoop.mapred.output.compress=true \
  -Dspark.hadoop.mapred.output.compression.codec=org.apache.hadoop.io.compress.BZip2Codec" \
  rpt sansa query converted-rml.rq --out-file data.nt.bz2
```

The file system compression setup turned out to be significantly faster: 4 178s vs 10 368s. The shorter duration was obtained by running Spark with a 5 GB JVM heap memory limit, whereas the longer one was obtained with a limit of 14 GB which means that more RAM was available when the compression was done in the JVM. The main reason for the better performance of file system level compression is most likely that lzo is designed to sacrifice compression

[9]https://github.com/AKSW/RdfProcessingToolkit-Resources/tree/main/2024-05-27-KGCW-at-ESWC

ratio for speed and is thus generally faster than bzip2.[10] Also, we'd expect the implementation of a compression algorithm in the file system driver to be faster than in Java (even with JIT compilation). However, a more in-depth evaluation, such as by configuring Apache Spark with an lzo codec, is future work.

3. Conformance Track

The challenge of the conformance track is to establish conformance with the upcoming revised RML specification.[11] At the point of writing, the revision did not have a final official name, so we refer to it as RML2 in the following.

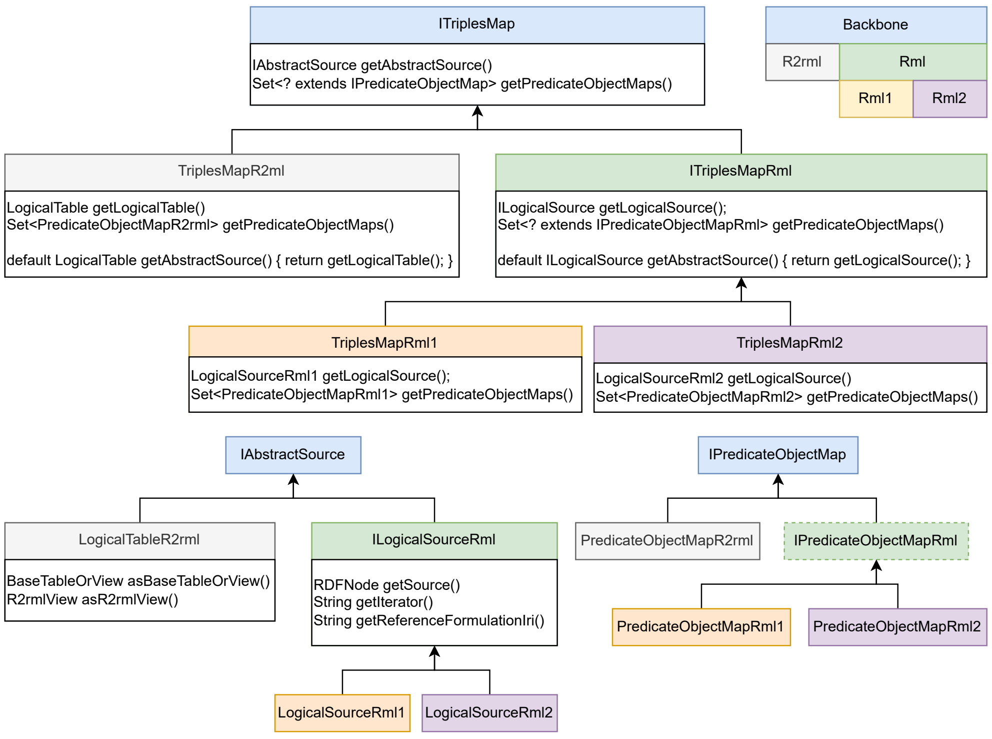
ITriplesMap
IAbstractSource getAbstractSource()
Set<? extends IPredicateObjectMap> getPredicateObjectMaps()
Backbone
R2rml
Rml
Rml1
Rml2
TriplesMapR2ml
LogicalTable getLogicalTable()
Set<PredicateObjectMapR2rml> getPredicateObjectMaps()
default LogicalTable getAbstractSource() { return getLogicalTable(); }
ITriplesMapRml
ILogicalSource getLogicalSource();
Set<? extends IPredicateObjectMapRml> getPredicateObjectMaps()
default ILogicalSource getAbstractSource() { return getLogicalSource(); }
TriplesMapRml1
LogicalSourceRml1 getLogicalSource();
Set<PredicateObjectMapRml1> getPredicateObjectMaps()
TriplesMapRml2
LogicalSourceRml2 getLogicalSource()
Set<PredicateObjectMapRml2> getPredicateObjectMaps()
IAbstractSource
IPredicateObjectMap
LogicalTableR2rml
BaseTableOrView asBaseTableOrView()
R2rmlView asR2rmlView()
ILogicalSourceRml
RDFNode getSource()
String getIterator()
String getReferenceFormulationIri()
PredicateObjectMapR2rml
IPredicateObjectMapRml
PredicateObjectMapRml1
PredicateObjectMapRml2
LogicalSourceRml1
LogicalSourceRml2

Figure 2: UML showing an excerpt of how R2RML, RML1 and RML2 are aligned in our system.

Some of the major changes introduced by RML2 are as follows:

- RML2 is now a model on its own and no longer an extension of R2RML. As a consequence, most ontology elements now reside in the namespace `http://w3id.org/rml/`.
- The specification is now modular, with the modules for (a) the core (*RML-Core*), (b) sources and targets (*RML-IO*), (c) containers and collections (*RML-CC*), (d) RDF-Star generation (*RML-Star*), and (e) functions (*RML-FNML*). A *logical views* module is being worked on.

[10] https://linuxaria.com/article/linux-compressors-comparison-on-centos-6-5-x86-64-lzo-vs-lz4-vs-gzip-vs-bzip2-vs-lzma
[11] https://kg-construct.github.io/rml-resources/portal/

In this work, we only attempted to establish conformance with RML-Core. In order to add support for RML2, we needed to decide whether to rewrite our engine from scratch or whether to generalize the interfaces and classes used to capture the RML model. We decided to take the latter approach. An excerpt of the revised class hierarchy is shown in Figure 2. We integrated the conformance test suite as unit tests using *JUnit*[12] and *Testcontainers*.[13] One issue we encountered was that the benchmark tool would download version 0.9.0 of the conformance test suite rather than v1.0.0, which caused issues in running the PostgreSQL tests. As a remedy, we downloaded the version 1.0.0 manually. This also has been fixed in the meantime. Furthermore, we were not able to generate the `results.zip` for the test cases with the benchmark tool because it would terminate abnormally. The reason has yet to be investigated. Of the 238 test cases of rml-core, we were able to establish conformance in 236 cases. The failing tests cases were 9a-mysql and 9b-mysql. These test cases include a join between an `int` and `varchar` column. RPT first maps the MySQL types of the columns to *xsd:int* and *xsd:string* respectively, before executing the join in SPARQL. Since these types are incompatible in SPARQL, the join fails. While the test case is arguably not ideal, the solution to mitigate the issue in the future is to push the join to the database. Interestingly, the corresponding PostgreSQL test cases have the column types fixed because PostgreSQL refuses to execute such joins.

4. Conclusions and Future Work

A significant improvement in this year's performance setup was that the participants could evaluate their systems on a homogeneous environment. Although the benchmark tool is capable of collecting measurements for various performance metrics, it still lacks the functionality to generate summary reports. The performance results showed that our tool is robust under limited resources and in the course of our evaluation it turned out that lzo compression on the file system level clearly outperformed Spark's built-in bzip2 support. As future work we aim to add proper support for parallel ingestion of XML data. We will analyze to which extent the remaining RML2 modules can be translated to SPARQL elements. For example, while support for *RML-FNML* should be fairly easy to add, support for *RML-IO* will require an extra RDF vocabulary for describing how to transfer the results of SPARQL queries to specified destinations.

Acknowledgments

The authors acknowledge the financial support by the German Federal Ministry for Economic Affairs and Climate Action in the project Coypu (project number 01MK21007A) and by the German Federal Ministry for Digital and Transport in the Project Moby Dex (project number 19F2266A).

[12]https://junit.org
[13]https://testcontainers.com/

References

[1] C. Stadler, L. Bühmann, L.-P. Meyer, M. Martin, Scaling RML and SPARQL-based knowledge graph construction with Apache Spark, in: Proceedings of the 4th International Workshop on Knowledge Graph Construction, ESWC, 2023.

[2] D. Van Assche, D. Chaves-Fraga, A. Dimou, U. Şimşek, A. Iglesias, KGCW 2024 Challenge @ ESWC 2024, 2024. doi:10.5281/zenodo.10973433.

[3] S. Bin, C. Stadler, L. Bühmann, KGCW2023 Challenge Report RDFProcessingToolkit/Sansa., in: KGCW Challenge @ ESWC, 2023.

[4] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus, O. Corcho, GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain, Journal of Web Semantics 65 (2020) 100596.