KGCW2023 Challenge Report RDFProcessingToolkit / Sansa

Simon Bin, Claus Stadler and Lorenz Bühmann

Institute for Applied Informatics (InfAI), Leipzig

Abstract

This is the report of our participation in the KGCW2023 Challenge @ ESWC 2023 with our RDFProcessing-Toolkit/Sansa system which won the "fastest" tool award. The challenge was about the construction of RDF knowledge graphs from RML specifications with varying complexity in regard to the mix of input formats, characteristics of the data and the needed join operations. We detail how we integrated our tool into the provided benchmark framework. Thereby we also report on the issues and shortcomings we encountered as a base for future improvements. Furthermore, we provide an analysis of the data measured with the benchmark framework.

Keywords

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

1. Introduction

This is the report of participating in the KGCW2023 Challenge[1] using the RDF Processing Toolkit (RPT) / Sansa execution engine [1].[2] The challenge was divided into two main parts, called "Knowledge Graph Construction Parameters" and "GTFS-Madrid-Bench" [2]. The challenge description and possible results ("ground truth") are also published on Zenodo [3]. The remainder of this report is structured as follows: In Section 2 we describe how we set up our tool with the benchmark environment. In Section 3 we present an analysis of the results obtained with the benchmark system. Finally, in Section 4 we conclude this report.

2. Setting up RPT/Sansa with the KGCW Challenge Tool

Accompanying the challenge, a pipeline tool[3] was provided. *The tool* is built on Python, JSON, and Docker. It was *strongly encouraged* to use the tool and of course, the promise of its provisioning is to ease the evaluation of the system under test and have a common ground for comparing results.

KGCW'23: 4th International Workshop on Knowledge Graph Construction, May 28, 2023, Crete, GRE
sbin@informatik.uni-leipzig.de (S. Bin); cstadler@informatik.uni-leipzig.de (C. Stadler); buehmann@informatik.uni-leipzig.de (L. Bühmann)
0000-0001-9948-6458 (C. Stadler)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

[1] https://kg-construct.github.io/workshop/2023/challenge.html
[2] https://github.com/SmartDataAnalytics/RdfProcessingToolkit
[3] https://github.com/kg-construct/challenge-tool

After installing *The tool*, first we had some issues getting it to work. One major point of critique, which was also shared by other challenge participants, is that the provided version of *The tool* proceeded to *delete all Docker containers on the system*, even those that were not related to the challenge. For this reason, we had to patch our own fork of *The tool* such that only containers that were also created by it were removed. Additionally, the included MySQL component caused several troubles: — Amongst others, *The tool* created a MySQL database setup in the host computer's `/tmp` directory which was not cleaned up, leading to hard-to-debug version conflicts later on; — MySQL failed to start unless the file `bench_executor/config/mysql/mysql-secure-file-prive.cnf` was made read-only — the Python package `cryptography` was missing from the provided `requirements.txt`, leading to another exception in the MySQL adaptor.

Then, it was needed to integrate RPT/Sansa. The steps involved were:

1. Creation of a Docker image for RPT. As RPT already ships with several packaging options, including Docker,[4] there was no additional work needed.
2. Writing of a Python file that connects RPT to *The tool*. The file is `bench_executor/rpt.py` and was sufficiently easy to create by copying and adapting the `rmlmapper.py` file. However, there were some difficulties: — Class names *must not* start with lowercase, otherwise it is silently ignored. — *The tool* did not support running multiple steps with the same tool. However, RPT/Sansa required multiple invocations of the RPT tool for converting RML to a SPARQL workload, optimising the SPARQL workload and finally executing it. We had to add a hack with a global instance counter; — our container required configuring a working directory (Docker command line flag `-w`); this was not supported by *The tool* so we had to extend the base `ContainerManager` class.
3. Setting up the pipeline files (`metadata.json`) for the 63 test cases that execute the case using RPT.

The challenge dataset comes with 63 `metadata.json` files which use RMLMapper as the reference to implement the necessary steps. At first, we pondered if a search and replace operation on these files would be sufficient to run RPT, however it is easy to make mistakes in the process, which would have been detrimental to our challenge run. So instead we examined all files and identified the common patterns. On this basis, we reverse-extracted two YAML templates from those 63 files, one for each major part of the challenge. Then we continued to implement a template-to-`metadata.json` program which generates all 63 files from those two templates as required.[5]

Because we also wanted to be able to run and compare multiple tools using *The tool*, we furthermore opted to change the pipeline filename from `metadata.json` to *my-tool*`.json`, and added a command line switch to *The tool* to change the filename that it will read.

2.1. Fixing the mapping files

Unfortunately, the RML mappings provided in the challenge have some issues. As part of our pipeline, we first ran some SPARQL Update queries to fix the mappings: — replace invalid usage

[4] Maven artifact `rdf-processing-toolkit-pkg-docker-cli`
[5] https://github.com/SimonBin/kgc-challenge-tool-template

of string constant with IRI type by proper IRI,[6] — replace invalid JSON iterator `[*]` with `$.[*]`.[7]

Next, we also replaced R2RML in the mappings with RML and changed the table names to the CSV file names, as RPT / Sansa did not support relational databases (RDB) at the time of the challenge. It is noteworthy that this adaption was also made by at least one more participant. In this case, separate tasks for RDB and CSV would have been desirable.

2.2. Verifying the results

The results produced by our tool were then compared to the ground truth provided on [3] by sorting all the produced triples and comparing them one by one. It quickly became clear that some triples did not match up. However, it turned out that the issues lay with the ground truth.

- Many numbers in the ground truth are replaced with 999.999, one example from GTFS-Madrid-Bench, `scale_100`:
  ```
  <http://transport.linkeddata.es/madrid/metro/shape_point/00000000000000000001-1000803>
    <http://www.w3.org/2003/01/geo/wgs84_pos#long>
    "999.999999999999999"^^<http://www.w3.org/2001/XMLSchema#double> .
  ```
  (correct answer: `"3971182"^^<http://www.w3.org/2001/XMLSchema#double>`)
- Data types are suddenly present in the ground truth where none are specified in the mapping files, one example from GTFS-Madrid-Bench, `heterogeneity_files`:
  ```
  <http://transport.linkeddata.es/madrid/metro/trips/000000000000000009kc>
    <http://vocab.gtfs.org/terms#direction>
    "1"^^<http://www.w3.org/2001/XMLSchema#integer> .
  ```
  (correct answer: `"1"`)

2.3. Data quality issues

There is one more caveat with the GTFS benchmark: wrong use of data types. The `arrival_time` and `departure_time` are mapped to `xsd:duration`, however their values are not a valid duration. (Example: `0000000000000000006l`; valid duration example: `PT2M10S`). This causes systems that validate RDF terms, such as ours which is based on Apache Jena, to spend a significant amount of time emitting warning messages. Thus, a recommendation for the future would be to improve the benchmark with only well-formed RDF terms and/or have dedicated tasks for mappings that produce malformed ones.

3. Performance results

In this section we present the results which we obtained for RPT/Sansa using *The tool*.

Unfortunately for this challenge it was not possible to evaluate all participating tools on the same hardware. Each participant executed their own tool on their own system. Our **system**

[6] https://github.com/oeg-upm/gtfs-bench/issues/142#issuecomment-1453784200
[7] https://rml.io/specs/rml/#examples

was an AMD Ryzen 9 5950X with 128 GB RAM, of which Java was allowed to use 50% of the RAM.

THE TOOL generated 126 CSV files when giving the command `./exectool stats`. To that end, we wrote a program to first extract these files in an automated way and then to draw plots of the measured values of RPT. Otherwise it would be laborious and error-prone to create the charts. We also calculated the *number of triples* resulting from the mapping as well as the *input sizes of the source files* processed by the mapping (these values are not present in the CSV files created by *THE TOOL*).

For a comparison of RPT to other systems please refer to our system paper [1].

How to read the plots. Each of the following plots captures all of the following information. In order to assist the reader, we added circled numbers to Figure 1 which correspond to the numbers of this list:

1. The challenge was organised into multiple directories with different tasks. The parent *directory/task* is displayed on the *left* outer legend, the *left y-axis* shows the directory/different *parameter* configurations for this task.
2. The thick bar next to the parameter configuration is the *main bar*. It records the *run-time of RPT*. The total run-time in seconds is also printed at the end of the bar, and the run-time is also displayed on the *bottom x-axis*.
3. Each task "pipeline" can be configured to consist of multiple steps. We configured RPT to use six steps, the *duration of each step* corresponding to the different *coloured segments* on the main bar. (**1**) fix the mappings (see section 2.1), (**2**) configure the CSV "NULL" value (meaning unbound in SPARQL), (**3**) run the XML mappings (not parallelised), (**4**) convert the RML mapping into an equivalent SPARQL SERVICE query (see [1]), (**5**) run source optimisation on the SPARQL query, (**6**) execute the SPARQL query mapping description using RPT/Sansa.
4. The *right y-axis* shows the *CPU time scale* factor. This shows the distributedness of the execution (higher = more distributed). It was calculated by dividing the CSV column `cpu_user_system_diff` by the `duration`.
5. A *red* I *mark* shows the value of the `memory_ram_max` CSV column (*memory consumption*). The memory consumption is also displayed on the *top x-axis*. Note, that the measurement has very limited informative value because *THE TOOL* captures the initial virtual memory claimed by the Java Virtual Machine (JVM, 28 GiB) and not the actually used memory. We have no reason to suspect that less memory would be a problem for RPT.
6. The *small bar* above the main bar and …
7. the *small bar* below the main bar are to be read from top to bottom. They display the *relative size* of the *input data* on *top* and the relative size of the *output triple count* on the *bottom* of the main bar (blue). In the case of *heterogeneous input formats*, the top bar is *colour coded* for the size of the XML, JSON and CSV input. Note that while the raw numbers are absolute, the plot only gives a *relative perception* of the input and output sizes, with a size of 0% on the far left.

The tasks *join 1-1*, *join 1-N*, *join 1-10*, *join N-1*, and *join 10-1* (which are supposed to test scaling the number of joins) as well as *empty-values* (which should test the handling of empty

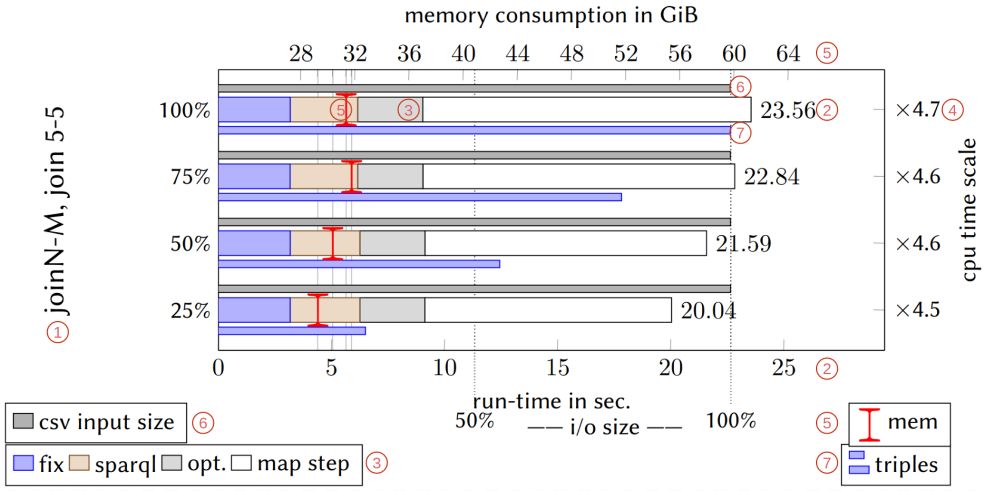

Figure 1

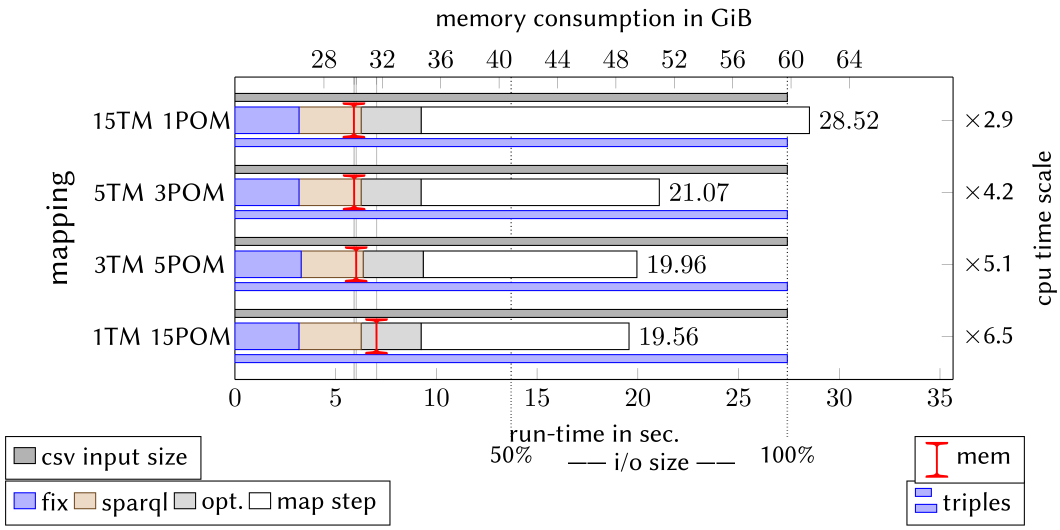

Figure 2

— unbound — values in the dataset) do not exhibit much variation. RPT requires roughly 20 seconds to complete them, with a CPU time scale of ×4.4. (Of these, approx. 12 seconds are start-up overhead of Docker/the system.) None of the parameter variations seem to have much influence here which might suggest the data/size/test case is too small for RPT. For the tasks *join N-M*, with N, M $\neq 1$, we can detect a slight increase in run-time depending on the number of joining triples and thus also the resulting triple count (see Figure 1). For the *duplicated-values*, RPT gets slightly faster the more duplicates, as it can eliminate them. This is not observed for empty-values, because there are always 100 000 unique `id` values whereas in the duplicated-values task x% of the *IDs are duplicates*.

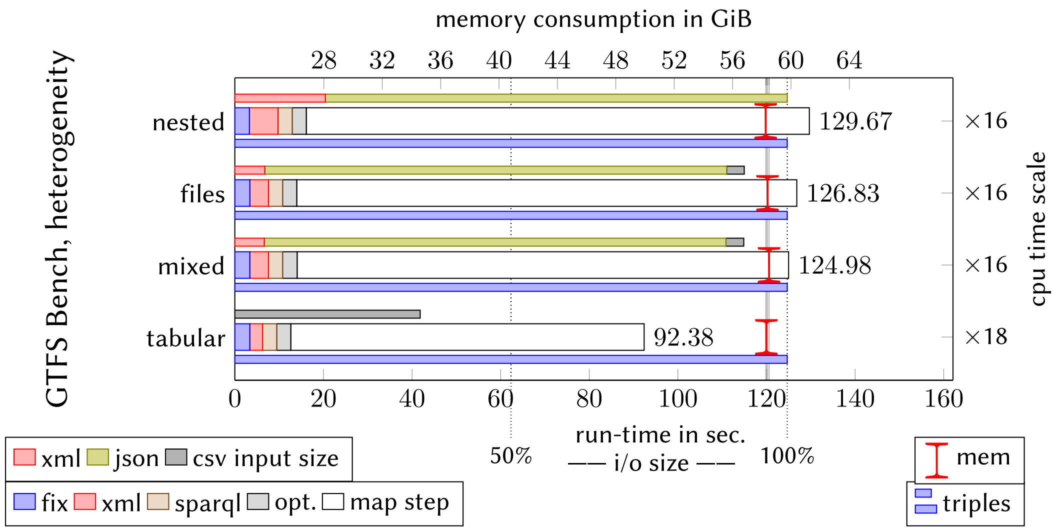

Figure 3

The *mappings* task sought to evaluate the balance between multiple *triple maps* (TM) and *property-object maps* (POM). Before running the task, we assumed that the resulting graphs should be equivalent and thus the run-time of RPT identical, since the source CSV and the triple counts are identical. However, the mappings provided are quite different. In the case of 15TM 1POM there are 1.5 mio. entities with exactly the same property `:p1` generated whereas in the 1TM 15POM case, we have only 100 000 entities with 15 properties each. Hence for 15TM 1POM the distinct operation is more expensive and cannot be parallelised well, as is evident in Figure 2.

For the *scaling* of the number of *records* (n.b. this task is in mismatch to the description of the challenge provided on [3]) and *properties* we can see that RPT scales exceptionally well. In the scaling of records, 1 mio. rows are processed in 45 seconds whereas ten times as many rows are processed in 425 seconds (×10), and for the properties 20 columns are processed in 46 seconds whereas 30 columns (+50%) are processed in 65 seconds (+41%).

The second part of the challenge was the *GTFS-Madrid-Bench*. We have also used this benchmark in [1], but we did not test RPT with *heterogeneous* GTFS Bench (n.b. there seems to be a mismatch between the description of the challenge and the actual content, we found the composition to be: csv+sql (tabular), xml,json,csv+sql (mixed), xml,json,csv (files), and xml,json (nested)). From Figure 3 we can see that CSV (tabular) has a much smaller input size. This benefits RPT two-fold, making it the fastest configuration of this task group. JSON files take the majority of the input data for all the other cases. In the nested configuration there is three times more XML than in the other two cases. Here, we can also observe that our XML processing time (second step) takes 6.5 seconds as compared to 4.2 seconds or 2.9 seconds (when no XML is present; start-up overhead). Additionally, with this data size the parallelism can be utilised, leading to a CPU time scale of ×16.

When *scaling* the GTFS Bench, we can see RPT shine. The GTFS Bench can generate input

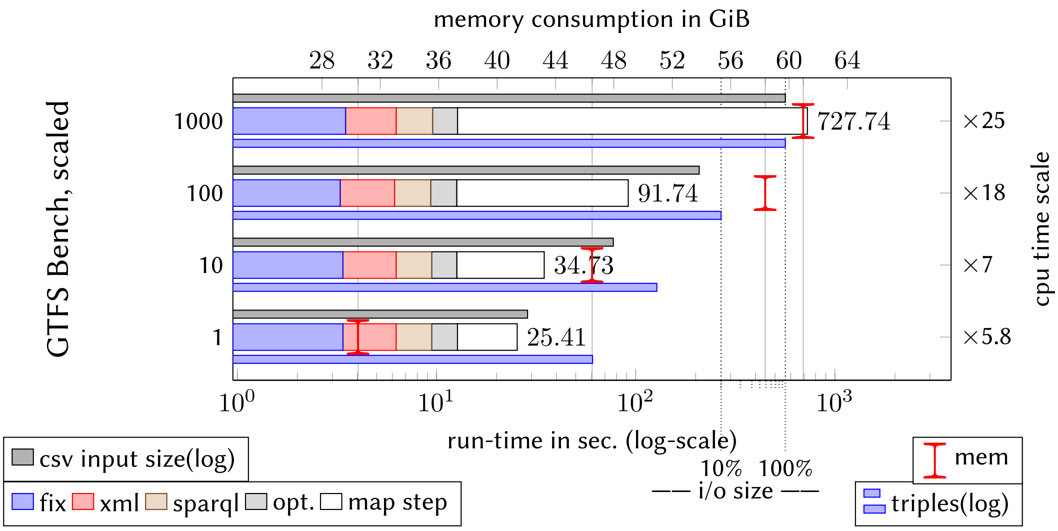

Figure 4

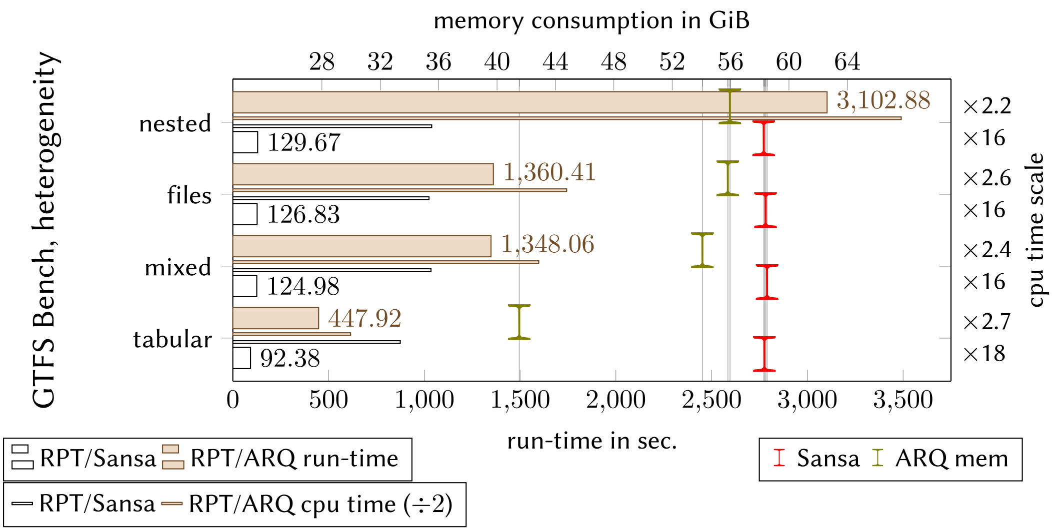

Figure 5

data for multiples of 395 953 output triples. For scale 1000 that amounts to almost 400 mio. triples. The performance of RPT can be seen in Figure 4. We changed to a logarithmic x scale to make it easier to compare differences in the order of magnitude. Similar to the scaling of records, the GTFS Bench at scale 100 takes 91.7 seconds, and at ten times it takes 727 seconds (an increase of only ×8). The parallelism also increased once more from a CPU time scale of ×18 for GTFS scale 100 to ×25 for GTFS scale 1000. That means in number of CPU seconds, for a ten-fold increase in output triples we have a still satisfactory eleven-fold increase.

Figure 5 shows a bonus plot comparing the execution time of RPT/Sansa to that of RPT/ARQ, another execution engine which is running on the same infrastructure as RPT/Sansa but with Jena ARQ instead of Apache Spark as the execution engine. This is a single-process execution engine without distributed/parallel computation.

The complete PDF of all plots, as well as the raw data, can be found in the RPT supplements repository.[8]

4. Conclusions and Future Work

We assumed that maybe all contenders in the challenge would use *The tool* to run their code, so that in the end we might have a collection of Docker containers to easily run different Knowledge Graph Construction tools. However, this effort has yet to happen. It would have been stellar if *The tool* were capable of generating a statistics report similar to the one we described automatically (ideally even with support for comparing multiple systems). So far *The tool* only collected data in a set of CSV files but all further interpretation had to be done individually by each participant. Finally, as apparent in the results, RPT/Sansa executes most tasks in around 20 seconds. As such, by comparing RPT only to itself there are not many interesting differences to be seen from which insights could be derived - besides that RPT executes those tasks in a stable and reliable way. Still, for future challenge runs, it may be worthwhile to refine the tasks further such as by scaling them up to larger sizes in order to see whether this leads to differences.

Acknowledgments

The authors acknowledge the financial support by the German Federal Ministry for Economic Affairs and Energy in the project CoyPu (project number 01MK21007A) and by the German Federal Ministry of Education and Research in the project StahlDigital (project number 13XP5116B).

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. 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.
[3] D. Van Assche, D. Chaves-Fraga, A. Dimou, U. Şimşek, A. Iglesias, KGCW 2023 Challenge @ ESWC 2023, 2023. doi:10.5281/zenodo.7837289.

[8] https://github.com/AKSW/RdfProcessingToolkit-Resources/tree/main/2023-05-28-KGCW-at-ESWC