RMLStreamer with Reference Conditions in the KGCW Challenge 2023

Els de Vleeschauwer[1], Gerald Haesendonck[1], Dylan Van Assche[1] and Ben De Meester[1]

[1]IDLab, Dept. Electronics & Information Systems, Ghent University – imec, Belgium

Abstract

Knowledge graph construction of heterogeneous data has seen a lot of uptake in the last decade from compliance to performance optimizations with respect to execution time. Besides execution time as metric for comparing knowledge graph construction, other metrics, e.g. CPU or memory usage, are often not considered. The Knowledge Graph Construction Workshop (KGCW) 2023 Challenge aims to be a competitive challenge for knowledge graph construction systems to encourage optimizations for execution time, CPU, and memory usage. We participated in this challenge with RMLStreamer, an RML mapping engine which processes all data in a streaming fashion. As the second part of the challenge is based on the Madrid-GTFS-Bench, which cannot be handled by RMLStreamer, we added RMLLooseGenerator as a first step for those experiments. RMLLooseGenerator is a proof-of-concept implementation that simulates the effect of using reference conditions in RML mapping rules. In previous work we showed that using reference conditions in the GTFS-Madrid-Bench mapping file results in exactly the same graph output. The challenge results show that RMLStreamer has a very scalable performance on execution time and CPU usage, while maintaining a constant memory usage. Therefore it received the Scalability Award in the KGCW 2023 Challenge. The challenge also highlighted some weaknesses of RMLStreamer: no support for relational databases, inefficient implementation of join operations, and longer execution time when handling nested sources such a JSON and XML files. After the challenge RMLStreamer has been expanded with support for relational databases. For future work we will research how to further optimize the handling of joins and of nested sources.

Keywords

RMLStreamer, challenge, knowledge graph construction

1. Introduction

Knowledge graph construction of heterogeneous data has seen a lot of uptake in the last decade from compliance to performance optimizations with respect to execution time [1, 2, 3]. Besides execution time as metric for benchmarking knowledge graph construction systems, other metrics, e.g. CPU or memory usage, are often not considered.

KGCW'23: 4th International Workshop on Knowledge Graph Construction, May 28, 2023, Crete, GRE

els.devleeschauwer@ugent.be (E. de Vleeschauwer); gerald.haesendonck@ugent.be (G. Haesendonck); dylan.vanassche@ugent.be (D. Van Assche); ben.demeester@ugent.be (B. De Meester)
https://dylanvanassche.be/ (D. Van Assche); https://ben.de-meester.org/#me (B. De Meester)
0000-0002-8630-3947 (E. de Vleeschauwer); 0000-0003-1605-3855 (G. Haesendonck); 0000-0002-7195-9935 (D. Van Assche); 0000-0003-0248-0987 (B. De Meester)

The Knowledge Graph Construction Workshop (KGCW) 2023 Challenge[1] aims to be a competitive challenge for knowledge graph construction systems to encourage optimizations for execution time, but also CPU and memory usage. This challenge consists of two parts: (i) knowledge graph construction parameters to evaluate individual parameters, e.g. joins and duplicates, with artificial data, and (ii) GTFS-Madrid-Bench [1] to focus on real-life use cases based on public transport data from Madrid.

In this paper we present the results of the KGCW 2023 Challenge for RMLStreamer [4], in combination with RMLLooseGenerator[2], the proof-of-concept implementation simulating the effect of using reference conditions [5]. The full setup and results are available on Zenodo[3].

Section 2 describes the components of our knowledge graph construction pipeline. Section 3 discusses the setup we used to execute the challenge's experiments. Section 4 explains our setups with other RML engines, to enable a comparison. We present our results in Section 5 and our conclusion in Section 6.

2. Knowledge Graph Construction Pipeline

Our knowledge graph construction pipeline consists of two parts: (i) RMLLooseGenerator emulates the effect of reference conditions [5], and (ii) RMLStreamer executes the RML mapping rules in a streaming fashion [4].

2.1. RMLLooseGenerator

RMLLooseGenerator is a proof-of-concept implementation for simulating the effect of using reference conditions [5] in RML mapping rules. It replaces joins that do not need references outside the join conditions with crafted URIs, and can be used by any RML engine as a first step of the knowledge graph construction process. This component is only used for the GTFS-Madrid-Bench cases. RMLStreamer cannot generate any output for the GTFS-Madrid-Bench within one hour when using join conditions. However, the output of the GTFS-Madrid-Bench remains identical when re-interpreting all join conditions as reference conditions [5].

2.2. RMLStreamer

RMLStreamer executes RML mapping rules to generate high quality Linked Data from multiple originally (semi-)structured data sources in a streaming way. RMLStreamer processes all data in a streaming fashion. Therefore, RMLStreamer handles big input files and continuous data streams like sensor data, without consuming more memory when the input data size increases. RMLStreamer leverages Apache Flink to scale vertically across multiple CPU cores and horizontally across multiple machines. For the challenge

[1]https://doi.org/10.5281/zenodo.7689310
[2]https://github.com/RMLio/rml-loose-generator
[3]https://doi.org/10.5281/zenodo.8034245

we deployed RMLStreamer version 2.4.2 with an embedded Flink version in a Docker container[4].

3. Experiment setup

The KGCW 2023 Challenge provides CSV files as source datasets, baseline results (i.e. the expected set of triples and query results) and an example pipeline based on the RMLMapper, MySQL, and Virtuoso for reaching those results. We adapted the provided experiments to enable execution with RMLStreamer.

In the provided end-to-end pipelines the CSV files are loaded into a relational database. As the RMLStreamer did not support SQL yet when the execution of the challenge was performed, we used CSV files directly to construct the knowledge graphs.

We executed all experiments on a Intel Xeon CPU E5645 (12 cores with Hyper-Threading, 2.4GHz) with 24GB RAM and 250GB HDD. All experiments were performed 5 times and the experiment with the median execution time is reported.

We compared our experiments' results to ensure that our output is correct with respect to the baseline results of the challenge. For the first part of the challenge, where the output of RMLStreamer is not loaded into a triples store, we deduplicated the output results as RMLStreamer cannot eliminate duplicates by itself. Our validation concluded that all tested experiments gave correct output.

4. Comparison with other RML engines

We also executed the experiments in our setup (section 3) with RMLMapper[5] [6] for both parts of the challenge, and with Morph-KGC[6] [2] for the GTFS-Madrid-Benchmark part. This way, we can compare the results of RMLStreamer on the same setup with other implementations.

5. Results

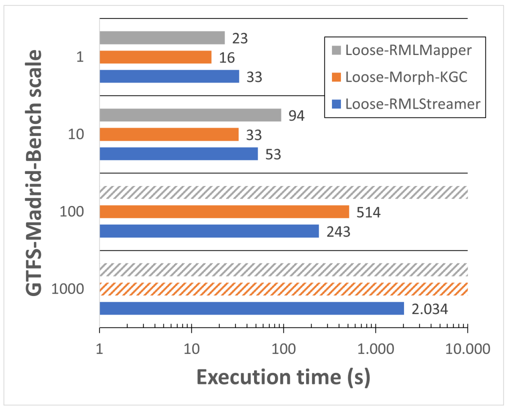
Figure 1a and fig. 1b show that RMLStreamer scales linear with the size of the input data, where RMLMapper and Morph-KGC cannot complete the experiments for respectively scale 100 and scale 1000.

At scale 100 RMLStreamer is two times faster than the state-of-the-art RML engine Morph-KGC (fig. 1a).

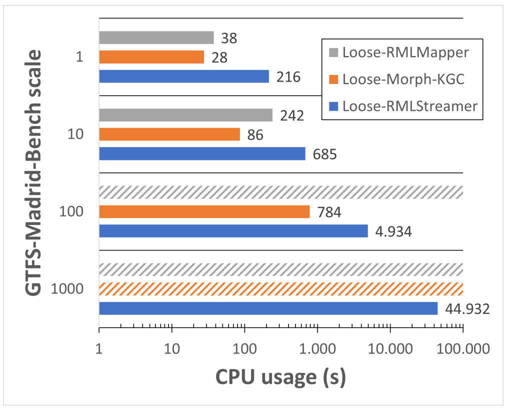
RMLStreamer uses more CPU (fig. 1b) in comparison to RMLMapper and Morph-KGC because it maximizes parallelism over all given slots. However, CPU usage scales linear with the size of the input data.

The peak RAM memory (fig. 1c) measured is similar for all scales when using RML-Streamer, where RMLMapper and Morph-KGC hit the limits of the available memory

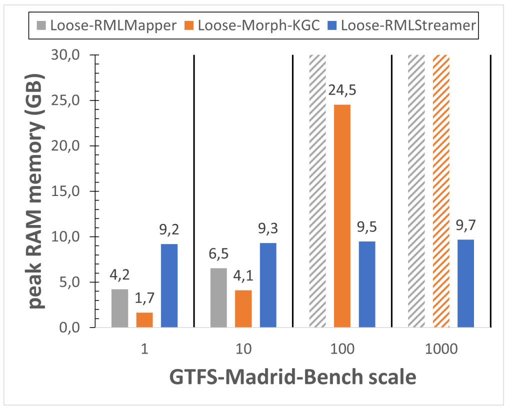
[4]https://doi.org/10.5281/zenodo.7181800
[5]https://github.com/RMLio/rmlmapper-java
[6]https://doi.org/10.5281/zenodo.5543552



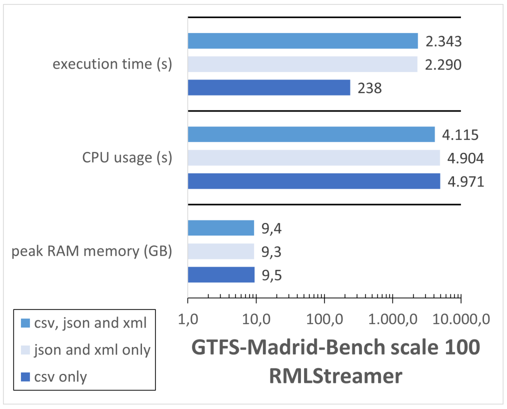
(a) Linear trend: the execution time of RMLStreamer increases with the same factor as the data size for higher scales.



(b) Linear trend: the CPU usage of RMLStreamer increases with the same factor as the data size for higher scales.



(c) RMLStreamer has a constant memory usage independent of data size



(d) The execution time of RMLStreamer increases with a factor 10 when including json and xml sources

Figure 1: Metrics of the knowledge graph construction pipeline for the GTFS-Madrid-Bench experiments. A striped bar signifies that this test could not be completed by the engine at hand.

and fail to complete all GTFS-Madrid-Bench experiments. RMLStreamer has a constant memory usage independent of the data size. We assume this is mostly due to the fact that it processes everything in a streaming way, which is to a lesser extent the case for RMLMapper and Morph-KGC.

Adding nested sources, such as JSON and XML files, increases the execution time of RMLStreamer with a factor ten (fig. 1d). The difference in execution time is the consequence of RMLStreamer chunking CSV files and processing the chunks in parallel. This is not the case for the XML and JSON formats yet.

The measurements for the knowledge graph construction parameter experiments show similar trends: linear scaling of execution time and CPU usage, proportional to the size of the input data, in combination with a constant memory usage. Table 1 shows the

	Execution time (s)	CPU (s)	Peak RAM (GB)	Output (triples)
1. Easiest experiment: records 10K rows 20 columns				
RMLMapper	8	15	1,4	200.000
RMLStreamer	21	107	2,4	200.000
2. Hardest experiment for RMLMapper: properties 1M rows 30 columns				
RMLMapper	262	646	13,2	20.000.000
RMLStreamer	120	2.247	9,2	20.000.000
3. Hardest experiment for RMLStreamer: records 10M rows 20 columns				
RMLMapper	out of memory	out of memory	out of memory	out of memory
RMLStreamer	653	14.360	9,2	200.000.000
4. Easiest experiment with joins: join 1-1 0%				
RMLMapper	out of memory	out of memory	out of memory	out of memory
RMLStreamer8	219	2409	9,3	0
RMLStreamer	43	371	9,4	0
5. Hardest experiment with joins: join 5-5 100%				
RMLMapper	out of memory	out of memory	out of memory	out of memory
RMLStreamer8	434	4145	9,3	2.500.000
RMLStreamer	66	1093	9,5	2.500.000

Table 1
Metrics of the knowledge graph construction step for selected knowledge graph construction parameter experiments

results of the experiments with the lowest and the highest measured values, which are dubbed easiest and hardest experiments in the table respectively.

We also added the easiest and hardest experiment including joins. At the time of the challenge we limited RMLStreamer to eight task slots (referenced as RMLStreamer8 in Table 1) for the execution of experiments including joins, because RMLStreamer reported an error and failed to start processing some mapping files including joins (e.g. the original GTFS-Bench-Mapping with joins). We assumed that this error appeared with any mapping file including joins. Further investigation afterwards revealed that this error got triggered by mappings with a large number of mapping rules (i.e. a mapping with two triples maps and one join operation does not result in error). Hence, the limitation of task slots is not required for the experiments with joins in the first part of the challenge. For completeness we added the results of those join experiments with all task slots. Using all 24 task slots on the hardest experiment with joins decreases the execution time by a factor of 16, and the CPU usage by a factor of nine, compared to the results registered during the challenge using RMLStreamer8.

6. Conclusion

The KGCW 2023 Challenge results show that RMLStreamer has a linear scaling of execution time and CPU usage, proportional to the size of the input data, while maintaining a constant memory usage. Therefore it received the Scalability Award in the KGCW 2023 Challenge.

We noted following improvement areas for RMLStreamer: (i) a lack of support for relational databases, (ii) an inefficient implementation of join operations (e.g. GTFS-Madrid-Bench experiments with joins cannot be handled by RMLStreamer), and (iii) a longer execution time when handling nested sources such a JSON and XML files.

Additionally we noticed that there are cases where the number of task slots used by RMLStreamer needs to be limited. At the time of the challenge this could only be achieved by modifying RMLStreamer's code.

Support for changing the number of task slots dynamically was implemented in RMLStreamer 2.5.0[7], together with support for SQL databases.

For future work we will investigate further optimizations for execution of joins and nested sources.

Acknowledgments

The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10), the imec ICON project AI4Foodlogistics (Agentschap Innoveren en Ondernemen project nr. HBC.2020.3097), and funded by the Special Research Fund of Ghent University under grant BOF20/DOC/132.

References

[1] 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. doi:10.1016/j.websem.2020.100596.

[2] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, Semantic Web (2022) 1–20. doi:10.3233/sw-223135.

[3] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, M.-E. Vidal, SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020. doi:10.1145/3340531.3412881.

[4] Sitt Min Oo, G. Haesendonck, B. De Meester, A. Dimou, RMLStreamer-SISO: An RDF Stream Generator from Streaming Heterogeneous Data, in: U. Sattler, A. Hogan, M. Keet, V. Presutti, J. P. A. Almeida, H. Takeda, P. Monnin, G. Pirrò, C. d'Amato (Eds.), The Semantic Web – ISWC 2022, Springer, Springer International Publishing, Cham, 2022, pp. 697–713. doi:10.1007/978-3-031-19433-7_40.

[5] E. de Vleeschauwer, S. Min Oo, B. De Meester, P. Colpaert, Reference conditions: Relating mapping rules without joining, in: Proceedings of the 4rd International Workshop on Knowledge Graph Construction (KGCW 2023) co-located with 20th Extended Semantic Web Conference (ESWC 2023), 2023.

[7]https://doi.org/10.5281/zenodo.7998156

[6] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Proceedings of the 7th Workshop on Linked Data on the Web, volume 1184, 2014.