A Functional and Distributed RML Processor

Corentin MICHAUX and Christophe DEBRUYNE

Montefiore Institute, University of Liège, 4000 Liège, Belgium

Research Question

Can a distributed RML processor for bounded batches be fully conformant with the RML specification *and* scale on large datasets?

Related Work

RML is a declarative language for mapping heterogeneous, non-RDF sources to RDF. Most existing engines target single-machine execution. Distributed processing of large, bounded batches remains rather underexplored.

- *Morph-KGC* and *SDM-RDFizer*→ *parallelized* RDF generation on a *single* machine.
- *RMLStreamer* → distributed, but for *streaming* data.
- *Stadler et al.* [1] → rewrites RML into extended SPARQL CONSTRUCT queries, evaluated through a *SPARQL-to-RDD* algebra layer.

FPMapper instead models RML constructs *directly* as composable functions → no intermediate SPARQL rewriting or query-algebra layer!

Design & Implementation: FPMapper

FPMapper is an RML processor written in **Scala** on top of **Apache Spark**.

RML constructs are modeled as *pure, composable functions*. The same pipeline runs unchanged locally or distributed over Spark partitions. Effectful operations (file I/O) are isolated using Cats Effect `IO`.
Apache Jena handles RDF serialization.

Logical iterations are parallelized into a *Resilient Distributed Dataset* (RDD), and term-map evaluation runs independently per iteration in a single `mapPartitionsWithIndex` pass.
Some "clever" management to avoid shuffles when generating blank nodes.

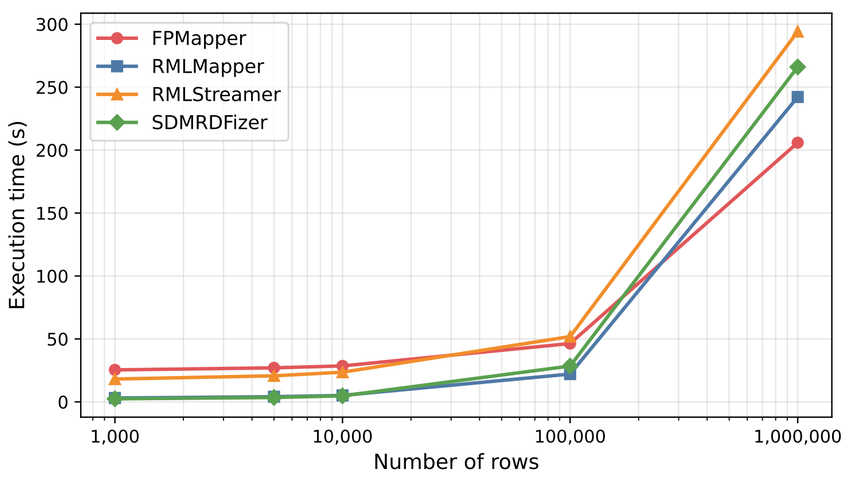
Deduplication is exact but graph-local in the default output mode. A fully deduplicated `-stdout` mode exists at the *cost* of collecting output to the driver.
Join resolution for RML-LV and Referencing Object Maps (currently) happens once, before distributed term-map evaluation.

Conformance

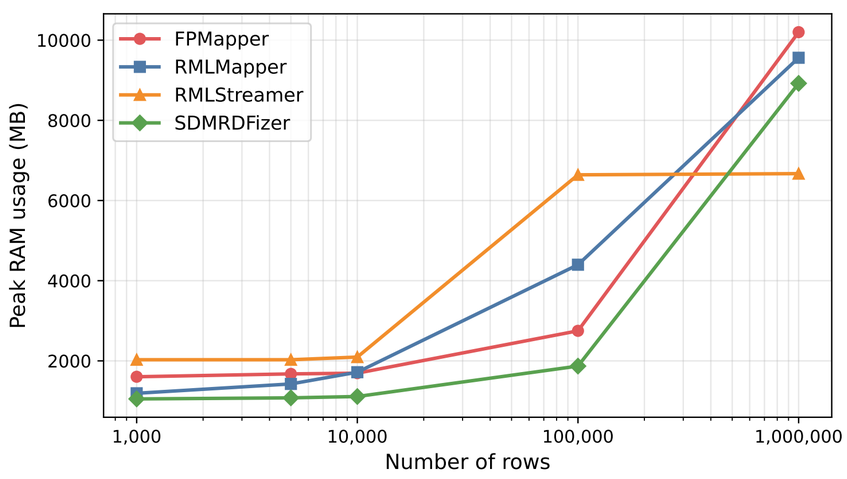
- Full conformance with the RML specification (Core, FNML, LV, and Star) on the RML test suite.
- RML-IO partially supported; we focused on CSV and JSON → This is engineering work, not a conceptual limitation.
- RML-CC not supported (cross-partition grouping, ordering, and blank-node coordination needed) → Requires research.

Scalability: KROWN Benchmark

FPMapper was compared against RMLMapper, SDM-RDFizer, and RMLStreamer using KROWN. The same Azure cluster was used (1 coordinator + 2 workers for FPMapper), though only FPMapper used the workers.



High fixed startup cost (≈20s) makes FPMapper and RMLStreamer slower at small/medium scale. At 1M rows, FPMapper becomes the fastest of the four engines, as data is big enough to be partitioned across the cores/workers.



FPMapper uses the most RAM at 1M rows (10.2 GB, driver-side only) → a **speed-vs-memory trade-off** accompanies its speed advantage.

Conclusions & Future Work

A functional and distributed RML processor is *viable for large bounded data, but not universally preferable* → favor single-machine engines at small/medium scale, while FPMapper's throughput performs better at large scale.
RMLMapper / SDM-RDFizer for typical workloads; FPMapper for very large batch jobs or data exceeding single-node memory; and RML-Streamer for streamed/chunked data.
Future work includes (identifying the limits of) distributed join resolution; more extensive experimental evaluation. The community also needs a KROWN(-like) benchmark for distributed RML processing.

Code

References

[1] Claus Stadler, Lorenz Bühmann, Lars-Peter Meyer, and Michael Martin. Scaling RML and sparql-based knowledge graph construction with apache spark. In *Proceedings of the 4th International Workshop on Knowledge Graph Construction co-located with 20th Extended Semantic Web Conference ESWC 2023, Hersonissos, Greece, May 28, 2023*, CEUR Workshop Proceedings. CEUR-WS.org, 2023.