Parallel sort-merge-join reasoning

Julien Subercaze[1] and Christophe Gravier[1]

Laboratoire Hubert Curien, UMR CNRS 5516
Université Jean Monnet
25 rue docteur Rémy Annino
F-42000, Saint-Etienne, France
firstname.lastname@telecom-st-etienne.fr

Abstract. We present an in-memory, cross-platform, parallel reasoner for RDFS and RDFSPlus . Inferray uses carefully optimized hash-based join and sorting algorithms to perform parallel materialization. Designed to take advantage of the architecture of modern CPUs, Inferray exhibits a very good uses of cache and memory bandwidth. It offers state-of-the-art performance on RDFS materialization, outperforms its counterparts on RDFSPlus and can be connected with Jena.
Reasons to see the poster: i) Presentation of the system, how to use it; ii) Discussion about implementation, source code walkthrough.

Keywords: in-memory reasoner, RDFSPlus, Jena, performance, open-source

1 Introduction

Answering SPARQL queries over RDFS and RDFSPlus [2] ontologies can either be solved by applying backward chaining [11, 4] or by materializing the results of the inference of process so that the queries can be processed by any RDF store that does not support inference. For both RDFS and RDFSPlus, the inference can be implemented as a iterative rule application process. In this paper, we present, for the first time to the Semantic Web community, Inferray [10], a high-performance forward-chaining reasoner. Inferray is designed to take advantage of the internals of modern computers: it fully leverages the cache hierarchy through a careful memory layout that fully utilizes the memory bandwidth. Inferray is developed at the University of Saint-Etienne, its source code is publicly available under the Apache 2 License. Being fully developed in Java, Inferray does not require architecture or operating system specific binaries [8] and is therefore completely cross-platform[1].

Rest of the paper is organized as follows: Section 2 we present a brief overview of state-of-the-art reasoners, Section 3 describes Inferray internals and the design choices that underpins its performance, finally Section 3.1 presents experimental results.

[1] https://github.com/jsubercaze/inferray

2 Related Work

Research in reasoner design and implementation roots to the advent of the Semantic Web technologies [3] and the list of related systems and publications is too long to fit here. We hereby focus on recent comparable systems and refer the reader to the recent survey of Kaoudi [7].

Forward-chaining reasoning can be performed either by iterative rules application, as done in Inferray, or by using the RETE algorithm [5]. The RETE algorithm, used by Jena and GraphDB(formely known as OWLIM for the reasoner module), due to its graph-based data structures, incurs lots of random memory access, thus hindering global performance [10]. The iterative rules application does not specify particular underlying datastructures, leaving room for various designs. RDFox [8] uses an almost lock-free data structure to efficiently parallelize hash-based joins and reports a good parallelization results. OWLIM reasoners family uses a custom rule entailment process with a fixed-point inference mechanism.

3 System description

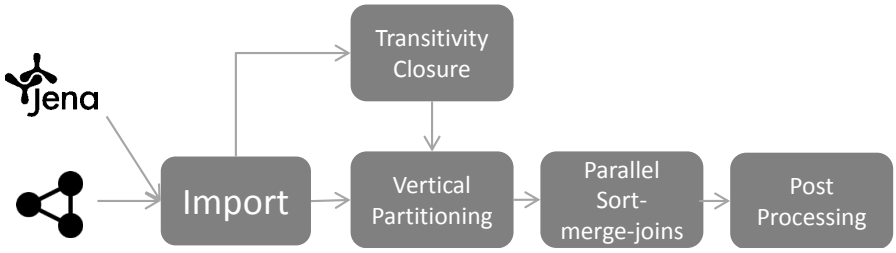

Fig. 1. Inferray Architecture

Inferray imports data either from files on the hard drive or to interact with the widely used Jena. After importation, the inference process is separated into different steps, depicted in Figure 1, the highlights are presented as follows:

Transitivy Closure To perform efficient transitivity closure, Inferray performs this task prior to the iterative rules application. This innovative approach, relies on a temporary data layout (before vertical partitioning) that allows the use of the state-of-the-art algorithm from Nuutila[9] to perform the transitivity closure. When the ontology contains a sufficient number of transitivity relations, the use of the temporary data layout and the data translation cost to the vertical partitioning layout are compensated by the efficiency of Nuutila's algorithm.

Dictionary Encoding & Vertical Partitioning Inferray uses a tricky dictionary encoding to compact the range of the IDs, while allowing an efficient data layout. Instead of starting numeration at 0 and increasing the value with incoming RDF resources, Inferray uses a dense numbering scheme that allows both vertical partitioning and the use of efficient sorting algorithms. The *s p o* triples are then splitted to be vertical partioned, using the standard partition on the predicate *p* [1], that offers a best selectivity for rules application. Triples are stored in arrays, whose indexes correspond to *p*, as continuous pairs of *o s*. Each array is sorted by *s* and possibly by *o* to efficiently perform sort-merge-joins.

Sorting algorithms Efficiently sorting is the cornerstone of high-performance sort-merge-join algorithms. Based on the dense numbering scheme, Inferray uses an adaptative sorting approach including a new counting sort algorithm for sorting pairs of integers. When outside the application domain of the counting sort, a custom MSD-Radix algorithm for sorting pairs kicks in. Our sorting experiments report througputs from 20 to 70 millions pairs per second, results that are at least on par with state-of-the-art algorithms.

Parallel sort-merge-joins Using array based layout, sort-merge-joins are performed efficiently due to a maximization of memory cache usage. In the first step, joins are perform on parallel, on a per-rule basis. Results of the joins is the materialisation of inferred triples, that may contains duplicates already present in the main triple store. The inferred triples are sorted and then merged in linear time into the main store, again in parallel manner. This efficient handling of duplicates largely contribute to the efficiency of Inferray.

Post processing The post-processing step handles corner cases such as rules having only one condition, this is for instance common in RDFS reasoning.

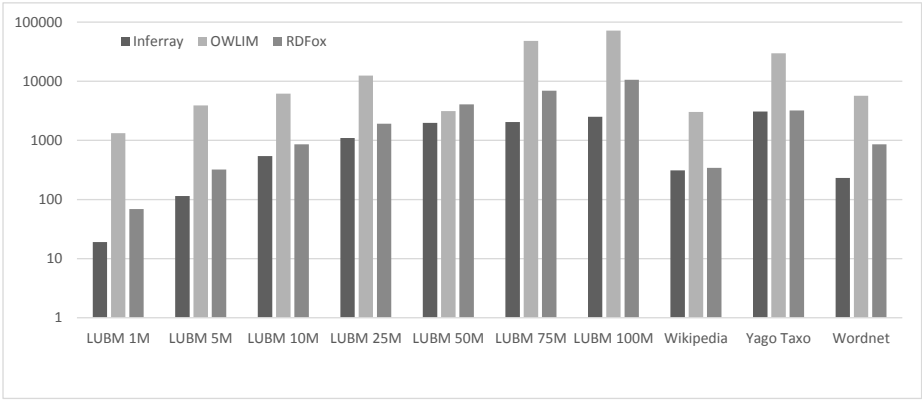

Fig. 2. RDFSPlus Inference time in milliseconds, log scale.

3.1 Performance

Experiments were conducted on a Intel Xeon E3 1246v3 processor with 8MB of L3 cache. Our system is equipped with 32GB of main memory; a 256Go PCI Express SSD. The system runs a 64-bit Linux 3.13.0 kernel with Oracle's JDK 7u67. We compared Inferray against RDFox and OWLIM-SE. To perform our experiments, we developed a dedicated benchmark suite [6] called `USE-RB`, that allows to report various performance metrics (cache pressure, memory usage) in addition to standard execution time. We report in Figure 2 the results obtained on RDFSPlus inference on various datasets: different size of the LUBM dataset as well as real-world ontologies. The results highlight the excellent performance of Inferray on RDFSPlus on both types of dataset.

4 Conclusion

In this paper, we presented Inferray, a high-performance reasoner based on parallel sort-merge-join. We believe that the presented system is of the utmost practical interest for the community. Its performance enable large scale processing of ontologies and its compatibility with the widely used Jena ensures its adoption by the end users.

References

1. D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *PVLDB*, pages 411–422, 2007.
2. D. Allemang and J. Hendler. *Semantic Web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.
3. J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *WWW*, pages 74–83. ACM, 2004.
4. O. Erling and I. Mikhailov. RDF Support in the Virtuoso DBMS. In *Networked Knowledge-Networked Media*, pages 7–24. Springer, 2009.
5. C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, pages 17–37, 1982.
6. C. Gravier and J. Subercaze. USE-RB: benchmarking how reasoners work in harmony with modern hardware. In *submitted to ISWC 2016 poster & demo track*.
7. Z. Kaoudi and I. Manolescu. RDF in the clouds: a survey. *VLDB J.*, 24(1):67–91, 2015.
8. B. Motik, Y. Nenov, R. Piro, I. Horrocks, and D. Olteanu. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *Proc. AAAI*, pages 129–137, 2014.
9. E. Nuutila. *Efficient transitive closure computation in large digraphs*. PhD thesis, Helsinki University, 1995.
10. J. Subercaze, C. Gravier, J. Chevalier, and F. Laforest. Inferray: fast in-memory RDF inference. *PVLDB*, 9(6):468–479, 2016.
11. J. Urbani, F. Van Harmelen, S. Schlobach, and H. Bal. QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases. *The Semantic Web–ISWC 2011*, pages 730–745, 2011.