VLog: A Column-Oriented Datalog System for Large Knowledge Graphs*

Jacopo Urbani[1], Ceriel Jacobs[1], and Markus Krötzsch[2]

[1] Dept. Computer Science, VU University Amsterdam, Amsterdam, The Netherlands
[2] Center for Advancing Electronics Dresden (cfaed), TU Dresden, Germany

1 Introduction

Rule-based ontology languages are well-suited for data-intensive applications, and rules therefore are an important topic in Semantic Web applications and research up to the present day [7,8,9,10]. The common foundation of many rule languages is *Datalog*, which was recently the focus of much renewed interest [4]. Modern Datalog systems such as *LogicBlox*, *SocialLite*, or *EmptyHeaded*, now often compete successfully with state-of-the-art techniques in their target area.

In spite of these advances, scalability remains a big challenge. We therefore have recently presented a new Datalog system, *VLog* (Vertical Datalog) [10], which exploits data management technologies used in column stores to compute efficiently rule-based computation on large Knowledge Graphs (KGs). Compared to traditional row stores, column-based approaches have shown performance advantages on analytical workloads [6], but were so far deployed mostly in relational DBMSs. With VLog, we show that Datalog too can benefit from column-store technology. This is not obvious, since the advantages of column stores are set off by a comparatively high cost of updates [2], whereas the iterative computation of Datalog query results can produce large numbers of derived facts that need to be inserted. Indeed, as far as we know, ours is the first work to successfully use column-based technology for Datalog processing.

VLog can process arbitrary Datalog rules on top of a range of RDF stores and relational databases. The heart of VLog is an efficient materialization procedure that computes derivations bottom-up. To improve performance, VLog packs a significant number of carefully engineered optimizations on several levels.

The goal of this demonstration is to provide hands-on insights into the workings of the system. We will show Datalog processing performed live, on a laptop, through a graphical monitoring interface that displays details of the computation. We will use different datasets, and show the impact of several optimizations. A demonstration provides an ideal format for discussing implementation aspects and practical experiences that would be difficult to present in a paper. Moreover, we will show important features that have never been published before, especially VLog's new RDBMS bindings, which were introduced only recently.

* This work was partially funded by COMMIT, the NWO VENI project 639.021.335, and the DFG in grants KR 4381/1-1 and CRC 912 *HAEC*.

2 VLog: Vertical Datalog

VLog was designed to perform efficient rule-based computations on knowledge graphs (KGs). VLog can evaluate arbitrary (positive) Datalog programs using several computing strategies as detailed in [10]. The heart of bottom-up evaluation is a modified *semi-naive evaluation*, aided by precomputed relations that are computed top-down using the *QSQ-R* algorithm (or optionally by a *Magic Sets* procedure [3]). Essential characteristics of the system include:

1. A variety of database systems can be used as a source for the data that rules are evaluated on. The separation of the underlying database from the actual query computation is natural in Datalog, where one distinguishes *EDB* relations (*extensional DB* relations; the given input data) from *IDB* relations (*intensional DB* relations; the derived data). VLog completely isolates the IDB layer, which is a column-based in-memory store, from the EDB layer, which can be any database. In addition to an RDF backend[3], VLog also natively supports popular relational databases (MySQL, MonetDB) and a generic ODBC interface.

2. Space-efficient in-memory storage of derivations is achieved using column-store technology. VLog stores IDB tables column-by-column instead of row-by-row. To avoid costly updates, VLog never inserts new tuples into existing tables, but creates new tables instead. This is efficient when processing data *set-at-a-time*, with many new facts derived and stored in one step. Column-based layouts can safe memory through simple yet effective data compression, such as run-length-encoding [1], and by sharing whole columns between multiple tables.

3. Bottom-up and top-down computation strategies, combined automatically and guided by dynamic optimizations, can lead to significant speed-ups. While VLog's column-based approach improves memory efficiency, computation can be slowed down when derivations from many tables must be considered. We counter this effect by adding several dynamic optimizations that enable us to disregard, at runtime, the contents of some IDB tables. In this way, we can avoid expensive unions and save significant computation (see [10] for details).

4. Free and open source code (C++) is available at https://github.com/jrbn/vlog. VLog was designed to be easy to use, and offers a command line and a Web interface. Rules can be defined in simple text files.

3 Experiments and Demonstration Setup

To gain insights into the performance of VLog, we report some experiments on reasoning on large knowledge bases. Additional experiments are also found in our other works [10,11]. We selected RDFox [8] as our main competitor since it is the leading Datalog engine for RDF. Experiments have been executed on a MacBook Pro with a 2.2GHz Intel Core i7 CPU, 512GB SDD, and 16GB RAM running on MacOS Yosemite, which is almost identical to the computer to be used during the demonstration. This means we can re-run the experiments live.

[3] *Trident*, the in-house RDF triple store used in our earlier work

Our selected experiments use two scenarios: "DBpedia" with 112M triples and 9,396 Datalog rules, and "LUBM" with 17M triples and 66 rules. VLog can easily handle much larger datasets, but we selected these two since they are popular and can run on a laptop. The inputs have been obtained by taking subsets of *DBpedia*[4] and LUBM [5], together with Datalog encodings of the OWL RL fragments of their native ontologies (see [10,11] for details).

First we compare VLog to RDFox on DBpedia. VLog can materialize all derivations for this input in 67sec using at most 648MB of RAM, while RDFox needs 177sec and 7,917MB. RDFox is optimized for parallel processing, so it may achieve better runtimes on a more powerful computer, but the effectiveness of VLog's memory savings (factor 12 in this case) would remain the same.

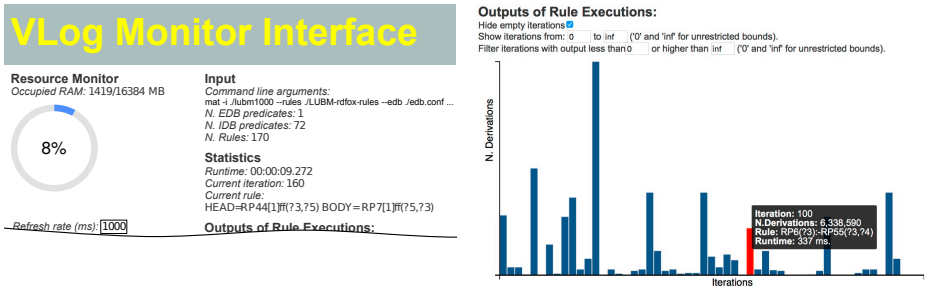
In a second experiment, we measure the performance of VLog using different backends on LUBM (a much smaller input). The runtimes we obtain are 16sec (Trident), 459sec (MySQL), 209sec (MonetDB), and 232sec (MonetDB via ODBC), respectively. It is not surprising that Trident is much more efficient than external relational stores, since it is specifically optimized for graph-structured data while the other systems support more general schemas. Moreover, Trident and VLog are compiled together and can run in a single process, while the others communicate via HTTP calls.

Demo Description We have developed a GUI that shows the state of the system during and after the computation. In our live demonstration, we can visualize the progress of Datalog derivations as they are computed. We will use a variety of datasets, rule sets, backend DBMS, and optimization settings to illustrate the working of VLog. When the computation has finished, the GUI offers access to further statistical information collected during the computation, which can be inspected and compared with results of other runs to understand the dynamics of the processing in detail.

The GUI is a dynamic HTML page that can be displayed in any browser. Figure 1a shows its header, which displays general parameters, the current state of the execution, and memory usage. The GUI also visualizes the progress of the computation, showing both the number of newly derived tuples (see Fig. 1b) and the time taken in each step (a similar bar chart). It is interesting to compare both views in order to see the effects of some optimizations and to detect possible performance bottlenecks. Discussing these differences provides interesting insights and can highlight potential for further optimization.

Demo Walkthrough For the live demonstration, we will launch several executions and trace the operation of VLog in our monitoring tool. We focus on cases that terminate in a few minutes since these are most appropriate to show the system at work. The demo setup is suitable to discuss not only the performance and optimizations of VLog, but also the behaviour of various database backends (MySQL, MonetDB, Trident) and the characteristics of the variety of benchmarks we have used. The execution can be customized with different settings, and we can quickly add new rules or disable others. We plan to discuss

[4] http://www.dbpedia.org



(a) Process Monitor Interface (b) Tuples Derived in Each Step

Fig. 1: Examples of the GUI used during the demo.

the results individually and in comparison with other settings to highlight interesting issues. We added several customizations to our interface to allow a lively interaction. For instance, the bar chart in Fig. 1b can be changed by restricting the view to particular subsets of iterations or rules. A 2-minute screencast of our demo can be found online: https://iccl.inf.tu-dresden.de/web/ISWC16demo/en.

References

1. D. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *Proc. of SIGMOD*, pages 671–682, 2006.
2. D. Abadi, A. Marcus, S. Madden, and K. Hollenbach. SW-Store: a vertically partitioned DBMS for Semantic Web data management. *VLDB J.*, 18(2):385–406, 2009.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.
4. O. de Moor, G. Gottlob, T. Furche, and A. Sellers. *Datalog Reloaded*, volume 6702. Springer, 2012.
5. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. of Web Semantics*, 3:158–182, 2005.
6. S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. MonetDB: two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.
7. M. Krötzsch and V. Thost. Ontologies for knowledge graphs: Breaking the rules. In *Proc. of ISWC*, LNCS. Springer, 2016. To appear.
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. of AAAI*, pages 129–137, 2014.
9. Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee. RDFox: A highly-scalable RDF store. In *Proc. of ISWC*, pages 3–20, 2015.
10. J. Urbani, C. Jacobs, and M. Krötzsch. Column-oriented datalog materialization for large knowledge graphs. In *Proc. of AAAI*, pages 258–264, 2016.
11. J. Urbani, C. Jacobs, and M. Krötzsch. VLog: A column-oriented datalog reasoner (extended abstract). In *Proc. of 39th Annual German Conf. on AI (KI'16)*, LNAI. Springer, 2016. To appear.