Compressing Datalog Materialization

Jacopo Urbani

Vrije Universiteit Amsterdam, The Netherlands

November 17, 2021

Datalog: The Scalability Problem

A key reasoning task with Datalog rules is **materialization**

Definition (informal)

Given as input a set of facts $\mathcal{F}$ and a set of Datalog rules $\mathcal{P}$, the *materialization* is a process that computes the smallest set of facts $\mathcal{M}_{\mathcal{F},\mathcal{P}} \supseteq \mathcal{F}$ that satisfies all the rules in $\mathcal{P}$. Sometimes, we call $\mathcal{M}_{\mathcal{F},\mathcal{P}}$ a *model* of $\mathcal{F}$ and $\mathcal{P}$.

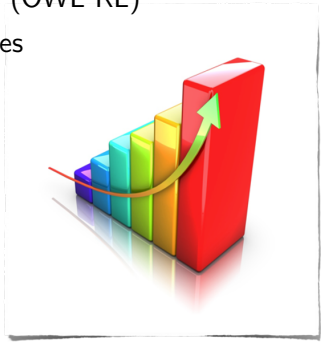
Datalog engines perform materialization prior query answering **OR** perform materialization during query answering (magic sets)

Datalog: The Scalability Problem

In practice, materialization is a task applied in many large-scale scenarios

Some examples

- (Some) reasoning with ontologies can be expressed with rules (OWL RL)
- Rules can be used for data wrangling or recursive graph queries
- LinkedIn uses a Datalog engine in production
- Google has developed its own Datalog engine (Yedalog)
- Samsung proposed its usage on mobile devices
- ...



Problem: We must find ways to materialize very large inputs

Datalog: The Scalability Problem

Terminology

Database: Set of facts $\mathcal{F}$
Program: Set of rules $\mathcal{P}$

Rule: $\underbrace{p_1(\mathbf{x_1}) \wedge \ldots \wedge p_n(\mathbf{x_n})}_{body} \rightarrow \underbrace{q(\mathbf{y})}_{head}$

For convenience, we view $\mathcal{M}_{\mathcal{F},\mathcal{P}}$ as $\Delta_0 \cup \Delta_1 \cup \Delta_2 \cup \ldots \cup \Delta_n$ where $\Delta_0 = \mathcal{F}$ and $\Delta_i, i > 0$ contains all the facts obtained by applying a rule in $\mathcal{P}$ on $\cup_{j<i}\Delta_j$

Problems

1. $\mathcal{F}$ can be too large to be stored on one machine
2. $\mathcal{M}_{\mathcal{F},\mathcal{P}}$ can be too large to be stored on one machine
3. Computing Δ_i can be time consuming
4. n can be very large
5. The problem is PTIME-complete

Techniques for Scalable Materialization

Parallelization and/or distribution is the mainstream approach to improve scalability

Rule parallelism
(e.g., RDFOx)


apply r_1 on $\mathcal{F}$


apply r_2 on $\mathcal{F}$


apply r_3 on $\mathcal{F}$


apply r_4 on $\mathcal{F}$

where $r_1, r_2, r_3, r_4 \in \mathcal{P}$

Data parallelism
(e.g., WebPIE)


apply r on $\mathcal{F}_1$


apply r on $\mathcal{F}_2$


apply r on $\mathcal{F}_3$


apply r on $\mathcal{F}_4$

where $r \in P$ and $\mathcal{F}_1 \cup \mathcal{F}_2\mathcal{F}_3 \cup \mathcal{F}_4 = \mathcal{F}$

Compressing Materialization

Compression is an alternative, possibly complementary, way to improve the scalability of rule-based reasoning.

Advantages

- We can reduce the input size, hence addressing Problems (1) and (2)
- Compression addresses also Problem (3)
- In contrast to parallelism, we do not suffer from Problem (5)
- It can be combined with parallelism to further improve the performance

Example (Dictionary Encoding (DE))

Build a bijective mapping ϕ that maps every symbol in $\mathcal{F}$ to a unique integer. Then, replace every symbol s with $\phi(s)$ in $\mathcal{F}$.

Dictionary Encoding

DE is a popular form of compression that is often done in a suboptimal way

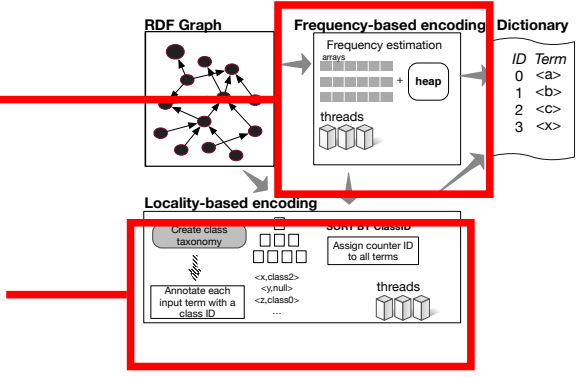
KOGNAC: The choice of numbers matters! [Urbani et al. (2016a)]

- Popular symbols should get smaller numbers
- "Similar" symbols should get consequent numbers

Use frequent set mining algos to find popular symbols. They receive small numbers

Improvement up to 10x!

Classes are grouped in a taxonomy. Symbols in the same class get consecutive IDs



VLog

VLog is a reasoner that exploits columnar storage and compression to improve the scalability of materialization [Urbani et al. (2016b)]

Main features

- Supports materialization with Datalog and existential rules (skolem and restricted chase)
- Supports negation via stratification
- Implements several acyclicity conditions
- Supports equality reasoning via standard axiomatization, singularization, and replacement
- The core engine is written in C++ and has very few dependencies. It works on Windows, Linux, MacOS, Android
- VLog is publicly available https://github.com/karmaresearch/vlog
- There is also a Java library called Rulewerk to facilitate its usage in Java https://github.com/knowsys/rulewerk

VLog

We introduce a short example to describe the main idea behind VLog

Example

Consider the application of rule

$$P(x, y) \wedge R(x) \rightarrow S(x) \tag{1}$$

and $\mathcal{F} = \{P(a_i, b_{i/2}) \mid 1 \leq i \leq 2n\} \cup \{R(b_j) \mid 1 \leq j \leq n\}$. Note that such execution creates n S-facts of the form $S(b_j)$
If we store the facts row-by-row, then the P-, R-, and S-facts are stored as:

$$P(a_1, b_0), P(a_1, b_1), P(a_2, b_1), P(a_3, b_2), \ldots, R(b_1), R(b_2), \ldots, S(b_1), R(b_2), \ldots$$

which leads to a total storage of $2 * n + n + n = 4n$ symbols

VLog

Example (cont.d)

Suppose we store the P- and R-facts as follows

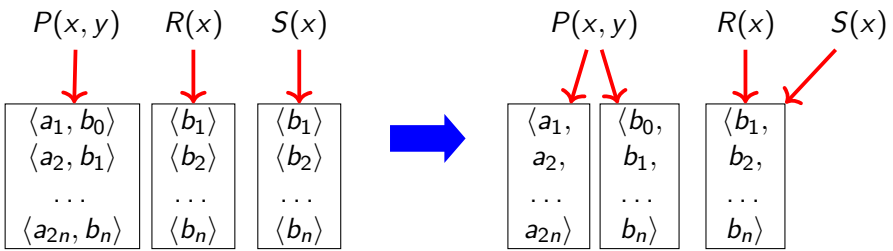
$$P(\langle a_1, \ldots, a_{2n} \rangle, \langle b_0, \ldots, b_n \rangle), R(\langle b_1, \ldots, b_n \rangle)$$

. Then, we can execute $P(x, y) \wedge R(x) \rightarrow S(x)$ more efficiently.

- We can ignore $\langle a_1, \ldots, a_{2n} \rangle$ when joining P- and R-facts. If we do so, the join considers $2n$ symbols instead of $3n$.
- No need to store $\langle b_1, \ldots, b_n \rangle$ again for S-facts. Total storage is $3n$ instead of $4n$.

VLog

The main idea behind VLog is to store the data using columns instead of rows.



Problem

How to deal with updates? **Work in append-only mode**

Three main advantanges

Advantage 1: Structure sharing

Instead of copying columns, we can store pointers (ok due append-only mode)

Advantage 2: Better compression

$\langle \underbrace{b, b, ..., b}_{n} \rangle \rightarrow \langle b \times n \rangle$ (from O(n) to O(1) storage)

$\langle b_1, b_2, \ldots, b_n \rangle \rightarrow \langle b_i \mid 1 \leq i \leq n \rangle$ same as above

Advantage 3: Avoid duplicate derivations

Consider rules $r_1 : P(x, y) \rightarrow Q(y, x)$ and $r_2 : Q(x, y) \rightarrow P(y, x)$.
If the database contains $P(c_1, c_2)$ and r_1 inferred $Q(p_{c_2}, p_{c_1})$, where p_{c_i} is a pointer to c_i, then skip r_2 on $Q(p_{c_2}, p_{c_1}) \Rrightarrow$ **avoided inference of** $|c_1|$ **duplicates**

About the derivation of duplicates (GLog)

Avoiding the derivation of duplicates is the problem that motivated the development of **GLog** – a spinoff of VLog (`https://www.github.com/karmaresearch/glog`) [Tsamoura et al. (2021)].

GLog proposes a new data structure, called *Trigger Graphs*, to perform materialization without generating (most) duplicates.

Example

Consider the rules

$$mother(X, Y) \rightarrow daughter(Y, X) \qquad (r_1)$$
$$daughter(X, Y) \rightarrow mother(Y, X) \qquad (r_2)$$

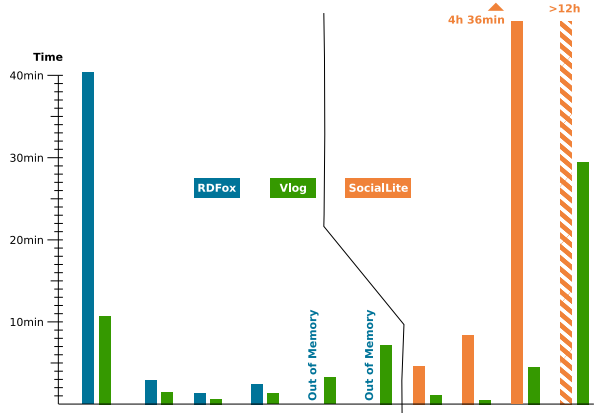
and $\mathcal{F} = \{Mother(Anna, Carla), Mother(Rose, Elena)\}$.

VLog: Evaluation

Competitors:

- RDFOx (ontogic. reasoning)
- SociaLite (graph analysis)

VLog outperforms the other systems, often significantly.

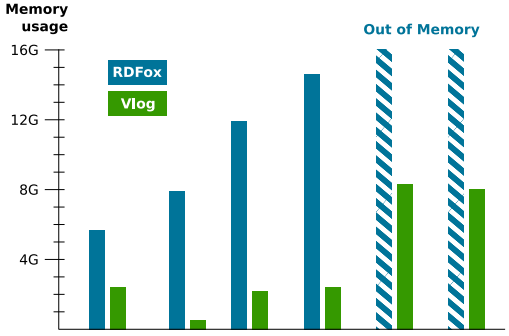


	Claros	DBpedia	LUBM-1K		LUBM-5K		Twitter	Google Webgraph		
Rules	2689	9396	170	202	170	202	2	5	5	5
Input tuples	19M	112M	133M		691M		1.75M	1M	2M	5M
Derived tuples	89M	33M	172M	197M	815M	994M	23M	31M	158M	769M

VLog: Evaluation

Best case: VLog uses 14X less RAM

Worst case: VLog uses 2X less RAM



	Claros	DBpedia	LUBM-1K		LUBM-5K	
Rules	2689	9396	170	202	170	202
Input tuples	19M	112M	133M		691M	
Derived tuples	89M	33M	172M	197M	815M	994M

VLog: Existential Rules

We extended VLog to support rules with existentially quantified variables

Two challenges

- Termination
- Runtime

Termination

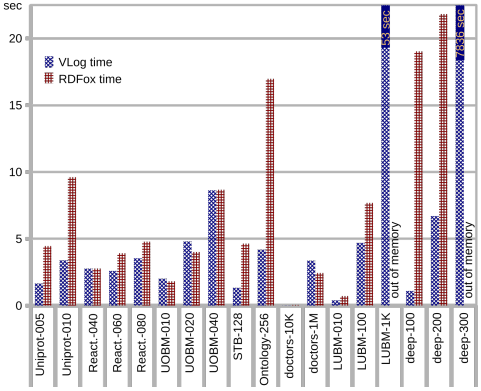
We implemented several well-known acyclicity conditions, namely weak acyclicity, MFA, MSA, JA, etc. [Urbani et al. (2018)] We also designed a novel condition that considers EGDs [Carral and Urbani (2020)].

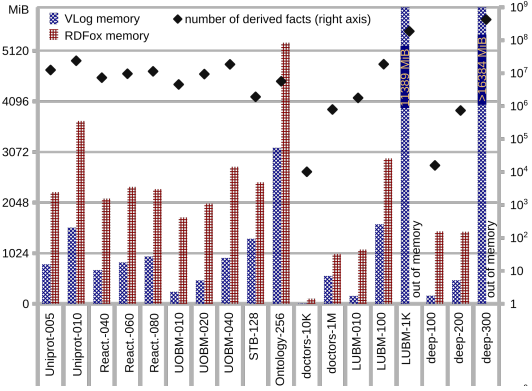
Runtime

We extended the columnar storage to include also null values; we also implement the checks necessary to execute the restricted and skolem chase.

VLog: Evaluation

VLog outperforms the competitors also with the restricted (standard) chase

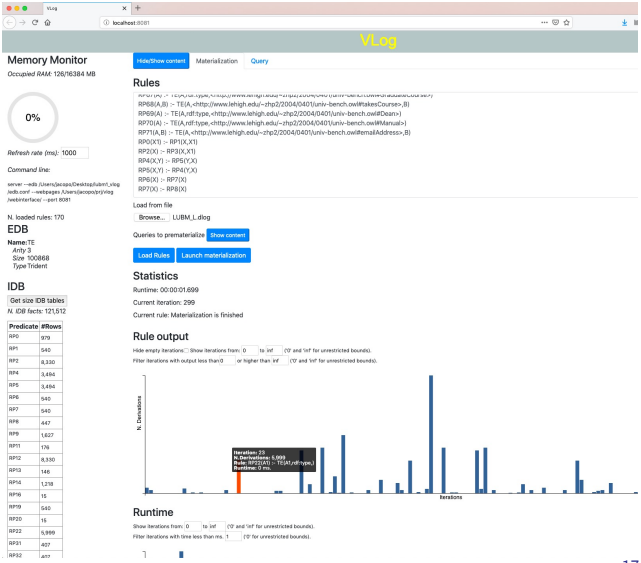




VLog: Web Interface and Docker

Web Interface [Carral et al. (2019)]

- Good for debugging
- Server mode
- Support querying
- Educational tool




```
docker pull karmaresearch/vlog
docker run -ti karmaresearch/vlog
```

Compression: Moving Forward

We can view columns as first-class citizen symbols [Hu, Urbani, Motik, and Horrocks (2019)].

Definition

A *meta-constant* $\mathbf{c}$ is a symbol with a mapping $\mu(\mathbf{c})$ that points either to a vector of non-decreasing constants $\langle a_1, a_2, \ldots, a_n \rangle$ or a to a vector of meta-constants.
Meta-facts are facts with meta-constants.

Differences with VLog:

- all sequences of constants are sorted
- meta-constants are hierarchical objects
- mappings are not immutable

Our goal

Reason efficiently on meta-facts trying to introduce as few meta-constants as possible.

Semi joins

Rules that require semi joins are easy to handle.

Example

Consider the rule $P(x) \wedge Q(x) \rightarrow S(x)$ and the meta-facts $P(\mathbf{a})$ and $Q(\mathbf{b})$ where $\mu(\mathbf{a}) = \langle a_1, a_2, \ldots, a_{2n} \rangle$ and $\mu(\mathbf{b}) = \langle a_2, a_4, \ldots, a_{2n} \rangle$.

1. Unfold $\mathbf{a}$, $\mathbf{b}$.
2. Introduce two fresh meta-constants $\mathbf{c}_1$ and $\mathbf{c}_2$ setting $\mu(\mathbf{c}_1) = \langle a_1, a_3, \ldots, a_{2n-1} \rangle$ and $\mu(\mathbf{c}_2) = \langle a_2, a_4, \ldots, a_{2n} \rangle$.
3. Replace $\mu(\mathbf{a})$ with $\langle \mathbf{c}_1, \mathbf{c}_2 \rangle$
4. Return $S(\mathbf{c}_2)$

Our approach increased database size by $O(1)$ instead of by $O(n)$ done by a conventional approach.

Cross joins

Let us look at a more complicated type of rule.

Example

Consider the rule $R(x, y) \wedge S(y, z) \rightarrow T(x, z)$ and facts $R(a_i, b)$ and $S(b, c_j)$ where $1 \leq i, j \leq n$.

1. Translate the S-facts into a meta-facts $S(\mathbf{b}, \mathbf{c})$ where $\mu(\mathbf{b}) = \langle b \times n \rangle$ and $\mu(\mathbf{c}) = \langle c_1, \ldots, c_n \rangle$.
2. For each $R(a_i, b)$, output a meta-fact $T(\mathbf{a_i}, \mathbf{c})$ where $\mu(\mathbf{a_i}) = \langle a_1 \times n \rangle$.

Our approach infers $O(n)$ new (meta-)facts instead of $O(n^2)$ new facts inferred by a conventional approach.

Evaluation

$||I||$ is the *representation size* of I, i.e., the number of symbols needed to store the facts in I

(numbers in millions)

	$\|\|\mathcal{F}\|\|$	$\|\|\mathcal{M}_{\mathcal{F},\mathcal{P}}\|\|$	$\|\|\langle\mathcal{F},\mu\rangle\|\|$	$\|\|\langle\mathcal{M}_{\mathcal{F},\mathcal{P}},\mu\rangle\|\|$
LUBM-1k	241.3	314.4	195.2	195.7
Reactome	22.7	32.3	20.2	25.1
Claros$_L$	32.2	105.5	28.1	31.2
Claros$_{LE}$	32.2	1065.8	28.1	413.9
	Conv. approaches		Ours	

Observations

- With our approach, the representation size is much smaller
- With our approach, the representation size grows much less

Evaluation

Runtime comparison (in seconds)

	RDFOx	VLog	Ours
LUBM-1k	488.3	300.1	**266.8**
Reactome	53.0	**27.5**	47.3
Claros$_L$	135.9	538.4	**59.1**
Claros$_{LE}$	3492.1	**3302.3**	10.2 k

Observations

- The best case is when the database is very regular
- The worst case is when the rules produces meta-constants with short vectors

	Avg. length μ	Max. length μ	Max. depth μ
LUBM-1k	7993	11.2M	3
Claros$_{LE}$	127	699k	2268

Conclusion

Main message

Compression is an effective way to improve the performance of materialization

Future work

- Explore more adaptive forms of compressions
- Interleave meta-constants with dictionary encoding
- Interleave compression with parallel reasoning
- ...

Thanks!

References I

David Carral and Jacopo Urbani. Checking chase termination over ontologies of existential rules with equality. In *Proceedings of AAAI*, pages 2758–2765, 2020.

David Carral, Irina Dragoste, Larry González, Ceriel Jacobs, Markus Krötzsch, and Jacopo Urbani. VLog: A Rule Engine for Knowledge Graphs. In *Proceedings of ISWC*, pages 19–35, 2019.

Pan Hu, Jacopo Urbani, Boris Motik, and Ian Horrocks. Datalog Reasoning over Compressed RDF Knowledge Bases. In *Proceedings of CIKM*, pages 2065–2068, 2019.

Efthymia Tsamoura, David Carral, Enrico Malizia, and Jacopo Urbani. Materializing knowledge bases via trigger graphs. *Proceedings of VLDB (PVLDB)*, 14(6): 943–956, 2021.

References II

Jacopo Urbani, Sourav Dutta, Sairam Gurajada, and Gerhard Weikum. KOGNAC: efficient encoding of large knowledge graphs. In *Proceedings of IJCAI*, pages 3896–3902, 2016a.

Jacopo Urbani, Ceriel Jacobs, and Markus Krötzsch. Column-Oriented Datalog Materialization for Large Knowledge Graphs. In *Proceedings of AAAI*, pages 258–264, 2016b.

Jacopo Urbani, Markus Krötzsch, Ceriel Jacobs, Irina Dragoste, and David Carral. Efficient Model Construction for Horn Logic with VLog. In *Proceedings of IJCAR*, pages 680–688, 2018.