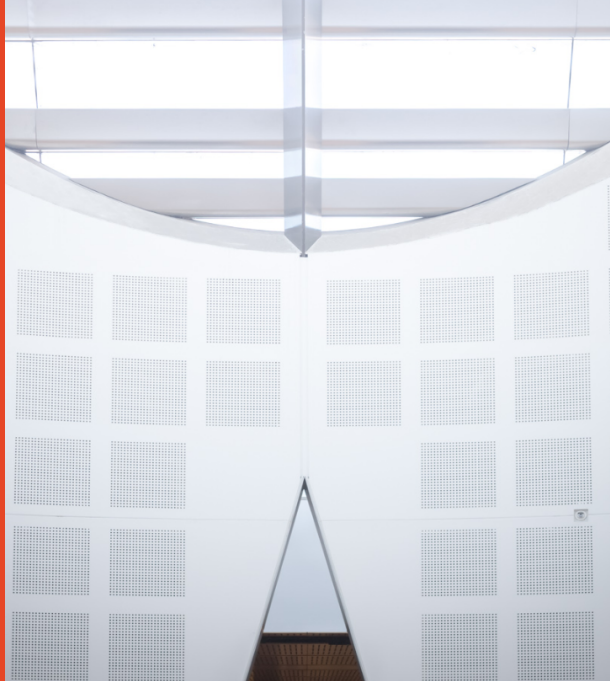
Fast Provenance in a Bottom-Up Evaluation

An Approach for Debugging Large-Scale Datalog

David Zhao, Pavle Subotic, Bernhard Scholz

THE UNIVERSITY OF SYDNEY



Introduction

Introduction

- Logic programming (e.g. Datalog) is popular [Aref et al., 2015]
  - Static program analysis
  - Declarative networking
  - Security analysis
- Evaluate at large scale, e.g. hundreds of millions of tuples
- Current debugging approaches do not scale well

We present a new approach to debugging that scales to super large sizes

Datalog

Declarative programming language - logical rules define computation

Example

```
path(x, y) :- edge(x, y).
path(x, z) :- edge(x, y), path(y, z).
```

Example Input

```
edge(1, 2), edge(2, 2), edge(2, 3)
```

Example Output

```
path(1, 2), path(2, 2), path(2, 3), path(1, 3)
```

Debugging in Datalog

Debugging Example

Program produces unexpected output `path(1, 4)`
Where does output come from?

- Debugging in Datalog is difficult
- Imperative language debugging:
  - Inspect values of variables at certain points in program
- In Datalog, we only get the output
  - No notion of variables
  - No notion of time

Provenance as a Debugging Tool

The answer is provenance!

Data Provenance

A way to explain the origins and derivations of data

- Previous approaches for provenance are expensive [Deutch et al., 2015, Köhler et al., 2012]

How do we compute provenance efficiently?

Provenance Computation

Proof Trees

A form of provenance - a complete explanation for a tuple

Definition (Proof Trees)

A *proof tree* for a tuple describes how that tuple is derived
The root is the tuple itself, tree explains which rules are applied and which tuples are used

Proof trees for `path(1, 3)`

$$\dfrac{edge(1,2) \qquad \dfrac{edge(2,3)}{path(2,3)}\,(r_1)}{path(1,3)}\,(r_2)$$

$$\dfrac{edge(1,2) \qquad \dfrac{edge(2,2) \qquad \dfrac{edge(2,3)}{path(2,3)}\,(r_1)}{path(2,3)}\,(r_2)}{path(1,3)}\,(r_2)$$

Fundamental Question

How do we compute a proof tree?
Apply one step of computation repeatedly

One step of computation

- Given a concrete tuple $R(a)$ and rule $R(X) :- R_1(X_1), \ldots R_k(X_k)$
- Want *subproof* for $R(a)$ - tuples for each atom $R_i(X_i)$ which generate $R(a)$

If we can do one step of computation, we can apply it recursively to get the full proof tree

Naïve Encoding

Directly store the subproof and rule for each tuple

Path program

```
path(x, y) :- edge(x, y).
path(x, z) :- edge(x, y), path(y, z).
```

Path	Subproof	Rule
$(1, 2)$	$edge(1, 2)$	r_1
$(2, 3)$	$edge(2, 3)$	r_1
$(1, 3)$	$edge(1, 2), path(2, 3)$	r_2

Naïve Encoding

Directly store the subproof and rule for each tuple

- Can directly query for a subproof
- Storing full provenance is expensive

Guided SLD

What information do we actually need for a subproof?

- Tuples matching the body of a rule
- Form the next level up in a proof tree

Guided SLD

What information do we actually need for a subproof?

- Tuples matching the body of a rule
- Form the next level up in a proof tree

So, we need

- The rule generating the tuple
- Its level in the proof tree

Guided SLD

A better method - generate annotations for each tuple

- Rule which generated tuple
- Level in proof tree for tuple

Path program

```
path(x, y) :- edge(x, y).
path(x, z) :- edge(x, y), path(y, z).
```

Path	Rule	Level
$(1, 2)$	r_1	1
$(2, 3)$	r_1	1
$(1, 3)$	r_2	2

Finding a subproof

Search for tuples matching the rule with lower level number

Guided SLD

Advantages:

- Only store 2 extra numbers per tuple
- Finds minimum height proof tree - optimality

Guided SLD

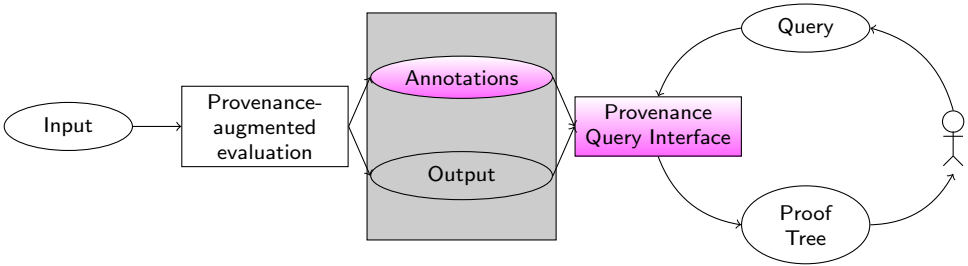


Figure: Diagram of guided SLD provenance system

Implementation in Soufflé

Soufflé

- Soufflé [Jordan et al., 2016] is a high-performance, compilation based Datalog engine - used in large-scale real-world applications

Implementation

- Datalog-to-Datalog transformation
- Guided SLD
  - Soufflé evaluation modification - standard set enforcement fails with annotations
  - Modified existing Soufflé machinery for subproof search

Provenance Query System

On-demand query interface

Demo

Figure: Provenance Query Interface

Experiments and Results

Overhead vs Normal Soufflé on Doop

Industry standard Doop DaCapo benchmarks

- Points-to analysis framework for Java
- Hundreds of millions of output tuples

Overhead vs Normal Soufflé on Doop

Industry standard Doop DaCapo benchmarks

- Points-to analysis framework for Java
- Hundreds of millions of output tuples

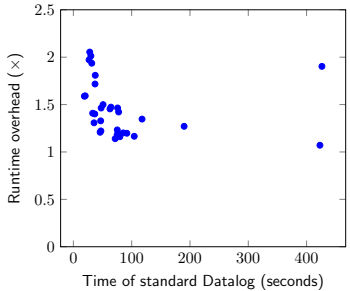


Figure: Runtime overhead of guided SLD

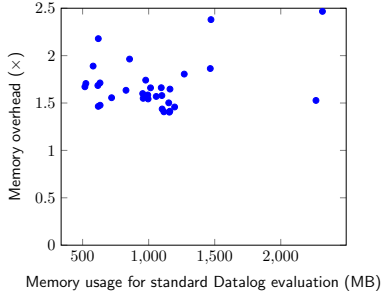


Figure: Memory usage overhead of guided SLD

Comparisons

Compared to state-of-the-art method (top-k [Deutch et al., 2015])

- Instrument Datalog for single query, and run on Soufflé

Comparisons

Compared to state-of-the-art method (top-k [Deutch et al., 2015])

- Instrument Datalog for single query, and run on Soufflé

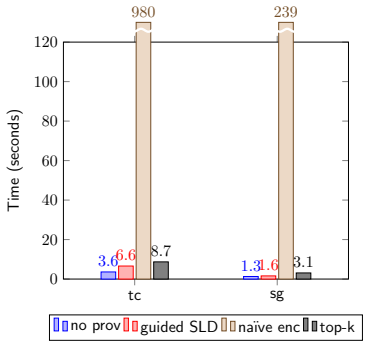


Figure: Results of Datalog evaluation time

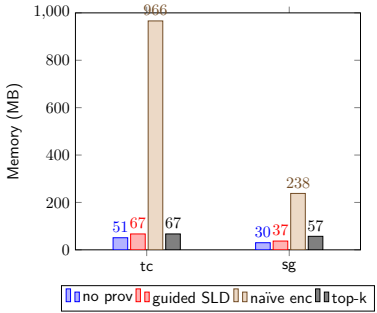


Figure: Results of Datalog evaluation memory usage

Proof Construction Time

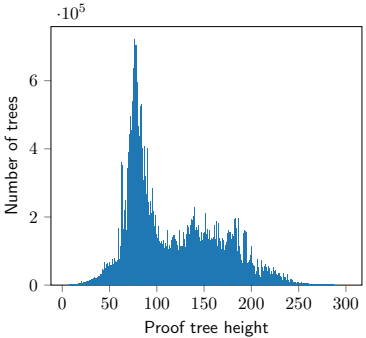


Figure: Distribution of proof tree heights for DaCapo

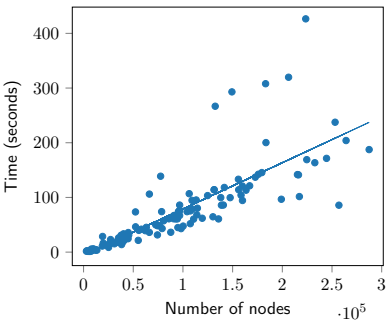


Figure: Proof tree construction time vs. size

Conclusion

Conclusion

- Debugging in Datalog is difficult
- Developed a solution to efficiently generate provenance information
- Demonstrated viability with large-scale real world data

Future Work

- Optimise Soufflé for guided SLD
- Provenance for negated Datalog

The End

References

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley Publishing Company.

Aref, M., ten Cate, B., Green, T. J., Kimelfeld, B., Olteanu, D., Pasalic, E., Veldhuizen, T. L., and Washburn, G. (2015). *Design and Implementation of the LogicBlox System*, pages 1371–1382. SIGMOD '15. ACM, New York, NY, USA.

Deutch, D., Gilad, A., and Moskovitch, Y. (2015). Selective provenance for datalog programs using top-k queries. *Proceedings of the VLDB Endowment*, 8:1394–1405.

Jordan, H., Scholz, B., and Subotic, P. (2016). Soufflé: On synthesis of program analyzers. *Proceedings of Computer Aided Verification*, 28:422–430.

Köhler, S., Ludäscher, B., and Smaragdakis, Y. (2012). Declarative datalog debugging for mere mortals. *Lecture Notes in Computer Science*, 7494:111–122.