Incremental Datalog Prototype in Soufflé

David Zhao, Pavle Subotić, Bernhard Scholz

1 Introduction

Datalog is a declarative logic programming language with wide uses in areas including static program analysis, network analysis, and security analysis. With recent parallel Datalog engines such as Soufflé, the performance of Datalog evaluation is on par with hand-crafted tools, while maintaining the ease of use and rapid prototyping capabilities [1]. However, Datalog engines like Soufflé do not support incremental computation, and marginally changed input relations cause the whole logic evaluation to be performed again.

A scalable incremental evaluation strategy has high utility in various Datalog applications, including static analysis where static analyzers are increasingly used in modern software development workflows [2]. In such scenarios, small, frequent incremental changes in source code are performed during development. A full re-evaluation of large code bases poses an unreasonable bottleneck during development. Thus to improve the practicality of such a workflow, an incremental evaluation strategy is paramount.

In this presentation, we propose a new incremental evaluation strategy for Datalog programs. This technique is inspired by early work on incremental Datalog evaluation, such as Delete-Rederive and the Counting algorithms [3], and its various improvements [4, 5, 6], as well as ideas from Differential Dataflow [7]. The advantage of our technique over previous Datalog approaches is that it eliminates the need to 're-derive' tuples that were over-deleted, and instead computes a precise set of changed tuples.

The key insight of our solution is to *instrument the Datalog evaluation* such that tuples are annotated with the *number of distinct derivations* for the tuple. To overcome the limitations of recursion for such an approach [3], we record additional information such as the iteration number of an evaluation step. The resulting scheme can be seen as coercing a recursive evaluation into a non-recursive evaluation by 'unrolling' the evaluation.

The contributions of this presentation are as follows:

1. Using a counting method for effective incremental evaluation of recursive Datalog programs
2. Reporting on the implementation of an incremental evaluation strategy in Soufflé
3. Showing preliminary experimental results using the incremental evaluation strategy

2 Comparison with Related Work

The main body of work in incremental Datalog evaluation is related to the Delete-Rederive (DRed) algorithm [3]. The main weakness of this and related approaches concerns the over-approximation when incrementally updated tuples are removed. A new Datalog program is constructed that uses the removed tuples as an input. All tuples generated by the incremental evaluation are considered 'over-deleted', and some of these tuples may remain computable in the final result. Therefore, a re-derivation procedure is required to maintain correctness.

Meanwhile, the Counting algorithm presented in [3] is optimized for non-recursive Datalog programs. For this approach, each tuple is associated with a *count* of the number of different derivations that exist for that tuple. When removing or inserting a new tuple, that count is decremented or incremented respectively, and a tuple may be removed if the count reaches 0. However, with recursive Datalog programs, deleting tuples may cause the recursive decrement of the count, thus again leading to over-deletion.

More recent developments include the Backward/Forward algorithm [4] and DRed^c [5], which are both optimizations of the DRed algorithm. The aim of these approaches is to reduce the approximation induced by the over-deletion step. Backward/Forward uses a form of *backwards evaluation* to check if over-deleted tuples still have a proof from the remaining input, while DRed^c maintains separate recursive and non-recursive counters to track the number of derivations of each tuple. While these approaches indeed reduce the over-deletion of DRed, they are still approximations and worst-case scenarios may exhibit poor performance.

Meanwhile, the Differential Dataflow system [7] implements incremental evaluation for Dataflow programming. The approach is similar to the Counting algorithm, with each tuple being associated with a count for the number of derivations for that tuple. However, Differential Dataflow permits recursive programs by storing a count *per iteration* of recursive evaluation. Its advantage is that it computes a precise result for an incremental update. However, the setting of Dataflow programming is different from Datalog, and more similar to stream programming, and thus, stopping conditions for recursive programs are not considered in the Differential Dataflow system.

Our approach aims to adapt ideas from the previous work to construct a practical incremental evaluation strategy in Soufflé. We instrument semi-naïve Datalog evaluation with counts per iteration of evaluation, thus enabling the counting schemes to be applied to recursive Datalog. By doing this, our scheme computes the precise result of an incremental update, without the need to re-evaluate over-deleted tuples, however, with some storage cost required for maintaining the per-iteration counts.

3 Incremental Evaluation

For incremental Datalog evaluation, the formal setup is as follows. Let P be a Datalog program, E_0 be a set of input tuples, and $P(E_0)$ be the result of evaluating P given input E_0. This forms our initial result for epoch number 0. In each subsequent epoch i, we have a set of tuples to be added E_i^+ and a set of tuples to be removed E_i^-. We wish to compute the result $P(E_i) := P((E_{i-1} \cup E_i^+) \setminus E_i^-)$ without fully re-evaluating the program.

As discussed above, our approach performs incremental evaluation by computing a *count* for each iteration of semi-naïve evaluation. Each distinct derivation for a tuple with all positive body tuples increments the count, and each distinct derivation where at least one body tuples has non-positive count (i.e., a deletion) decrements the count. These counts are separated for each round, essentially *unrolling* the recursive Datalog program into a series of non-recursive strata, thus allowing us to adapt the counting approaches. We also keep track of the epoch that the tuple is most recently updated in, which allows for further optimizations such as early stopping conditions for incremental updates.

Our technique is implemented as a prototype in the Soufflé Datalog engine. A user may perform incremental evaluation by using a command line interface to `insert` or `remove` a tuple. Each time an insertion or removal is performed, the incremental evaluation strategy is run, evaluating the incrementally updated Datalog program. Internally, the iteration, epoch, and count annotations are computed as extra fields in each relation, which incurs a slight overhead for memory consumption. Their correct computation is handled by instrumented Datalog rules, which use functors and constraints to ensure that the values of the annotations is correct after evaluation. The internal data structures of Soufflé are also adapted for incremental evaluation, allowing for the efficient maintenance of annotations.

References

[1] H. Jordan, B. Scholz, and P. Subotić, "Soufflé: On synthesis of program analyzers," in *International Conference on Computer Aided Verification*, pp. 422–430, Springer, 2016.

[2] P. W. O'Hearn, "Continuous reasoning: Scaling the impact of formal methods," in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, (New York, NY, USA), pp. 13–25, ACM, 2018.

[3] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining views incrementally," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 157–166, 1993.

[4] B. Motik, Y. Nenov, R. E. F. Piro, and I. Horrocks, "Incremental update of datalog materialisation: the backward/forward algorithm," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[5] P. Hu, B. Motik, and I. Horrocks, "Optimised maintenance of datalog materialisations," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[6] P. Hu, B. Motik, and I. Horrocks, "Modular materialisation of datalog programs," 2019.

[7] F. McSherry, D. G. Murray, R. Isaacs, and M. Isard, "Differential dataflow.," in *CIDR*, 2013.