Towards Scalable UDTFs in Noria

Justus Adam
Technische Universität Dresden
Chair for Compiler Construction
Dresden, Germany
justus.adam@tu-dresden.de

ABSTRACT

User Defined Functions (UDF) are an important and powerful extension point for database queries. Systems using incremental materialized views largely do not support UDFs because they cannot easily be incrementalized.

In this work we design single-tuple UDF and User Defined Aggregates (UDA) interfaces for Noria, a state-of-the art dataflow system with incremental materialized views. We also add limited support for User Defined Table Functions (UDTF), by compiling them to query fragments. We show our UDTFs scale by implementing a motivational example used by Friedman et al. [7].

ACM Reference Format:
Justus Adam. 2020. Towards Scalable UDTFs in Noria. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3318464.3384412

1 INTRODUCTION

Noria [8] is a novel database that implements incremental, *partially materialized views* [10, 17] that speed up reads significantly at the expense of immediate consistency. It leverages multicore parallelism and distributed deployment on clusters. This architecture is well suited for medium- to large-scale web applications where data access is read-heavy and non-uniform. Views in the Noria database are currently defined using a subset of SQL with no support for UDFs yet. This limits which kinds of processing tasks can be expressed and thus performed. For instance decoding or computing a distribution of a series is impossible to express.

In this work we try to remedy this by providing facilities for two basic kinds of UDF: single-tuple UDFs and User Defined Aggregations (UDA). Single-tuple UDFs can express decoding tasks or unit conversions, UDAs cover distribution or variance calculations. We apply known techniques [1, 15] for incrementalizing single-tuple UDFs and UDAs and partially automate the process for developer convenience.

We integrate the state needed by UDAs into Noria's materialization, which allows UDAs to be data parallelized and integrate with Noria's memory management.

```
fn main(clicks: RowStream<i32, i23, i32>)
       -> GroupedRows<i32, i32> {
    let click_streams = group_by(0, clicks);
    for (uid, group_stream) in click_streams {
        let sequences = IntervalSequence::new();
        for (_, category, timestamp) in group_stream {
            let time = deref(timestamp);
            let cat = deref(category);
            if eq(cat, 1) {
                sequences.open(time)
            } else if eq(cat, 2) {
                sequences.close(time)
            } else {
                sequences.insert(time)
            }
        };
        sequences.compute_average()
    }
}
```

Figure 1: Clickstream analysis as a UDTF

We further add support for a limited form UDTFs, which has not been done before. UDTFs pose a significant threat to performance. To the engine they are black boxes, preventing optimization such as parallelization. We apply insights from the domain of implicit parallel programming [4–6] and compile the UDTF to a query fragment which gives the engine a more fine grained view of the UDTF. To evaluate we implement a *clickstream analysis* UDTF (Figure 1) adapted from Friedman et al. [7]. They showed that this query is difficult to write in pure SQL and performs poorly. Our UDTF version is simpler to read and write and the experiments show the engine is able to parallelize it.

2 RELATED WORK

Simple materialized views [10, 13] recompute the entire query, which requires no changes to the UDF integration. Incrementally maintained materialized views [11, 17, 18] are more difficult because UDFs also need to be incremental.

DBToaster [1] does support UDFs, but only single-tuple ones, which are trivial to incrementalize. Mohapatra and Genesereth [15] support User Defined Aggregates only in so far as they have to also be defined in the query language Datalog, offering no effective integration for library code developed in other languages. Oracle [12] developed interfaces for incremental UDAs supporting foreign code. Our approach builds on a similar interface, but also automates part of incrementalizing the UDA. To the best of our knowledge there is no proposal yet for UDTFs in incremental materialized views.

There are approaches to scalable UDTFs, separate from materialized views, using MapReduce [7] and annotations [9]. Our approach uses a more general language than MapReduce [3] and is orthogonal to the annotation approach.

Stream processing systems, such as Apache Flink [2] also process data in a distributed and incremental fashion. However Database Management Systems (DBMS) also offers data-management capabilities [14] and leverage in-depth knowledge about size and structure [16] of the data when selecting processing strategies.

3 APPROACH

First we provide incremental single-tuple UDFs and UDAs, by applying techniques similar to DBToaster [1] and Oracle [12]. We also simplify the creation of incremental UDAs by noting that an aggregator state with reversible mutations is sufficient. We require the developer to provide this reversible state implementation and generate the rest of the UDA operator implementation automatically. UDTFs are incremental by construction, as they compile to query fragments, the constituents of which are all incremental.

To compile the UDTF we plug into the parallelizing language and compiler Ohua [5]. Our source language supports features that SQL does not, such as shared mutable state, enabling more efficient algorithms to be expressed. State sharing between dataflow operators however introduces potentially costly synchronization. It makes reasoning about the graph more complicated because the bidirectional state dependencies have different semantics to data dependencies.

Our plugin in fuses the manipulation of shared mutable state and generates Rust code which forms the core of a UDA operator. The state thus becomes operator-private, allowing safe, unsynchronized mutation. It also allows us to pair the state directly with Noria's materialization. With this pairing Noria manages the state memory, as well as partitioning and distributing it.

We also add a new backend to Ohua that targets Noria's query intermediate representation MIR. The main challenge here is the representation of procedural *control flow* in a query. Our source language supports iteration and conditionals. Conditional execution can be constructed using native `Filter` and `Join` operators and is omitted here for brevity. Iteration is different, because the type of data flowing between backend operators is restricted to database tuples. We therefore stream elements individually, tagged with indices.

The index facilitates state scoping. This occurs when state is used in a `for` loop (Figure 1). Scoping rules mean that each iteration of the loop needs a new state. We create a map of these states and dispatch for each tuple based on the sequence index. This lets us emulate scoped, procedural semantics using a cheap tagging mechanism.

4 RESULTS AND CONCLUSION

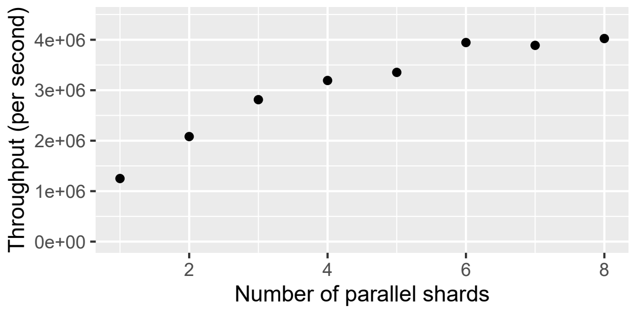

Figure 2: Scaling of the clickstream analysis UDTF over sharded data

We implement the efficient version of a clickstream analysis query from [7]. It shows that our approach allows intuitive, though restricted, procedural queries, such as in Figure 1, to run as a queries on Noria. They also parallelizes with Norias sharding capabilities, shown in Figure 2, without additional developer effort.

Our approach can express procedural UDFs as database query fragments. It allows for the use of state for efficient queries, correctly handling the scope of the state. These UDTFs are capable of leveraging parallelizing optimizations. Our approach could be expanded with annotations and analysis [9], giving the query engine even more information and further expanding the optimization potential. We also believe that a substantial part of the restrictions currently placed on the shape of the source UDTF could be lifted in the future.

5 ACKNOWLEDGEMENTS

This work was part of my Master's thesis at the Chair for Compiler Construction at the Technische Universität Dresden. I want to thank Sebastian Ertel, Malte Schwarzkopf, Dirk Habich and Jeronimo Castrillon for their invaluable input — this work would not have been possible without their help.

REFERENCES

[1] Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic. 2012. DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views. *Proc. VLDB Endow.* 5, 10 (June 2012), 968–979. https://doi.org/10.14778/2336664.2336670

[2] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. [n.d.]. Apache Flink: Stream and Batch Processing in a Single Engine. 36, 4 ([n. d.]).

[3] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (jan 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[4] Sebastian Ertel, Justus Adam, Norman A. Rink, Andrés Goens, and Jeronimo Castrillon. 2019. STCLang: State Thread Composition As a Foundation for Monadic Dataflow Parallelism. In *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell* (Berlin, Germany) *(Haskell 2019)*. ACM, New York, NY, USA, 146–161. https://doi.org/10.1145/3331545.3342600

[5] Sebastian Ertel, Christof Fetzer, and Pascal Felber. 2015. Ohua: Implicit Dataflow Programming for Concurrent Systems. In *Proceedings of the Principles and Practices of Programming on The Java Platform* (Melbourne, FL, USA) *(PPPJ '15)*. ACM, New York, NY, USA, 51–64. https://doi.org/10.1145/2807426.2807431

[6] Sebastian Ertel, Andrés Goens, Justus Adam, and Jeronimo Castrillon. 2018. Compiling for Concise Code and Efficient I/O. In *Proceedings of the 27th International Conference on Compiler Construction* (Vienna, Austria) *(CC 2018)*. ACM, New York, NY, USA, 104–115. https://doi.org/10.1145/3178372.3179505

[7] Eric Friedman, Peter Pawlowski, and John Cieslewicz. 2009. SQL/MapReduce: A Practical Approach to Self-describing, Polymorphic, and Parallelizable User-defined Functions. *Proc. VLDB Endow.* 2, 2 (aug 2009), 1402–1413. https://doi.org/10.14778/1687553.1687567

[8] Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, Lara Timbó Araújo, Martin Ek, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. 2018. Noria: dynamic, partially-stateful data-flow for high-performance web applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 213–231. https://www.usenix.org/conference/osdi18/presentation/gjengset

[9] Philipp Große, Norman May, and Wolfgang Lehner. 2014. A Study of Partitioning and Parallel UDF Execution with the SAP HANA Database. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management* (Aalborg, Denmark) *(SSDBM '14)*. ACM, New York, NY, USA, Article 36, 4 pages. https://doi.org/10.1145/2618243.2618274

[10] H. Gupta and I. S. Mumick. 2005. Selection of views to materialize in a data warehouse. *IEEE Transactions on Knowledge and Data Engineering* 17, 1 (Jan 2005), 24–43. https://doi.org/10.1109/TKDE.2005.16

[11] Himanshu Gupta and Inderpal Singh Mumick. 2006. Incremental maintenance of aggregate and outerjoin expressions. *Information Systems* 31, 6 (2006), 435 – 464. https://doi.org/10.1016/j.is.2004.11.011

[12] Ying Hu, Seema Sundara, and Jagannathan Srinivasan. 2010. Materialized views with user-defined aggregates.

[13] Ki Yong Lee and Myoung Ho Kim. 2005. Optimizing the Incremental Maintenance of Multiple Join Views. In *Proceedings of the 8th ACM International Workshop on Data Warehousing and OLAP* (Bremen, Germany) *(DOLAP '05)*. ACM, New York, NY, USA, 107–113. https://doi.org/10.1145/1097002.1097021

[14] S. Madden. 2012. From Databases to Big Data. *IEEE Internet Computing* 16, 3 (May 2012), 4–6. https://doi.org/10.1109/MIC.2012.50

[15] Abhijeet Mohapatra and Michael Genesereth. 2014. Incremental maintenance of aggregate views. In *International Symposium on Foundations of Information and Knowledge Systems*. Springer, 399–414.

[16] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. 2009. A Comparison of Approaches to Large-scale Data Analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Providence, Rhode Island, USA) *(SIGMOD '09)*. ACM, New York, NY, USA, 165–178. https://doi.org/10.1145/1559845.1559865

[17] Jingren Zhou, Per-Ake Larson, and Hicham G. Elmongui. 2007. Lazy Maintenance of Materialized Views. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (Vienna, Austria) *(VLDB '07)*. VLDB Endowment, 231–242. http://dl.acm.org/citation.cfm?id=1325851.1325881

[18] Yue Zhuge, Héctor García-Molina, Joachim Hammer, and Jennifer Widom. 1995. View Maintenance in a Warehousing Environment. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (San Jose, California, USA) *(SIGMOD '95)*. ACM, New York, NY, USA, 316–327. https://doi.org/10.1145/223784.223848