Report date 27/11/2024

Paper report:

**Overview:**

The paper presents a language to describe computations on stream (updates to a view in an IVM scenario can be seen as stream elements). They first explain how DBSP work and show the theorems that sit behind it in a rather theoretical way and then they explain how it can be used to implement relational algebra and easily obtain IVM pipelines for SQL views.

With DBSP we can incrementally express all the SQL, Datalog and streaming computations components.

The idea is based on the fact that a database can be represented as a stream of transformations and for every database (source) transformation, we can obtain a view (output) transformation that incrementally updates the view whenever the database gets updated.

Given a query Q that gives as the view V starting from the database DB (V = Q(DB)) we can extract a lifted (↑) version of the query that works with single increments of the database (ΔQ).

The pipeline looks like this (arrows are streams and letters are the operators or source/sink):

ΔDB → I → Q() → D → ΔV . Where ΔDB is the change in the source, I is the integral function since a view computation is affected by all the data ever come (using an integral gives us that “memory”), Q() is the actual function we use to model the source into the view, D is the derivative since we do want the incremental update, ΔV is the final incremental update of the view. The structure I → Q() → D is referred to as Q()Δ .

Q and Q()Δ have a whole set of interesting properties upon which DBSP is built on top.

The issue is that I and D are defined for abelian groups but relational databases are not abelian groups, so they exploit the Z-set (taken from another work), which consists in a map where the tuples are the key and the value is the number of time that tuple is found in the database, a negative value means that we want to delete that tuple (perfect for the cases in which we run a DELETE query on a table and want to update the VIEW dependent from that table).

The *distinct* operator is also used in order to produce sets, it's necessary for some operators only.

In order to get an IVM algorithm we need to:

* Translate Q (the query that gives us the view) into a DSBP circuit (they provide a table for that)
* Apply *distinct* where needed (often only at the end)
* Lift the whole circuit to make it able to work on stream values instead of the whole DB (they developed the code to do it)
* Make the circuit incremental by placing I to integrate just after every source and D to derive before the output
* “Reduce” the circuit by removing I and D and replacing them with the QΔ version of the operator (they provide the replacement for every operator).
* Now you have the circuit and the operators and you can get the ΔV given a ΔDB!

As previously mentioned they do provide the circuit for every SQL operator (composition,union, projection, filtering, join, intersection, difference).

A quick study on complexity aimed at showing that the space and time complexity is better than recomputing the whole view.

They do also explain how to incrementalize recursive queries.

Chapter 7 talks about how to use DBSP not only for relational algebra but also for SQL constructs.

They also explain how they implement some SQL functionalities such as GROUP BY, SUM, COUNT, MIN, MAX.

Everything is implemented in Rust and creates a rustc files that given a stream of DB changes retrieves a stream of queries that implement the IVM on the view.

It is an interesting tool to generalize construction of IVM pipelines and being fully open source and implemented appears to be “easy” to use and easy to further analyze.

**Experimental metrics and scenarios:**

The correctness of the tool has been tested. They theoretically verified the lemma and theorems via the Lean theorem prover. And empirically tested via the 7Mln SQL Logic Test.

**Limitations:**

* Not mentioned
* Performances are not reported and without looking the implementation/usage I cannot give an informed opinion

**Repo:**

<https://github.com/vmware-archive/database-stream-processor>

**Presentations:**

[Building a streaming incremental view maintenance engine with Calcite](https://www.youtube.com/watch?v=iT4k5DCnvPU)

[DBSP: Automatic Incremental View Maintenance for Rich Query Languages](https://www.youtube.com/watch?v=J4uqlG1mtbU)

**Authors:**

Mihai Budiu et ali.

**Year:**

2023