Report date 07/02/2025

Paper report:

**Overview:**

The need for incremental computation is introduced, the the main feature of differential dataflow is explained: partially ordered set of versions rather that totally order set of versions, this is due to the fact that there is not only one index to order the versions (version with starting data=0,first data update=1, second data update=2…). Differential Dataflow uses due indexes (i, j), where i is still the round of data input while j is the iteration of a loop in the code, since DD also allows loops. Therefore computation at (i, j) can be supported by the results obtained at (i-1, j) and (i, j-1) or (i-1, j-1). In this way it is possible to reuse everything we can to reduce the computing time.

Differently from IVM where every update is squashed into the temporary result, in DD every update of the output is stored sequentially and every partial result is stored so that data parallelization and reuse of previously computed states is maximized.

The benefits of DD are: ability to handle incremental computation, ability to handle data-parallel computations, ability to handle custom functions with a standard way to compute incremental updates (in the worst case it reconstruct all the current output and performs the operation in a standard non incremental manner).

Section 3 carefully explains the details of differential computation and how increments work and how all of them are stored.

Then 3 capabilities of Differential Computation are explained: iterative computation (manage multiple for loops), prioritized computation according to prioritization rules, composability and nesting thanks to loops and prioritization.

The general logic for implementing any operator is explained.

Then DD is specifically explained:

* Data parallel operations: operations can be parallelized by keying the values and splitting the data over different machines, the final data merge is then harder or easier based on the specific operator.
* Pipelined operators: Select, Where, Concat, Except are linear operators that need no state and therefore can be pipelined with other operators.
* Join: the logic behind the Join operator is explained
* Aggregations: aggregation operators in the context of data parallel operations are explained.

Then more details are reported about the implementation but those are out of scope for us.

Comparisons are made with similar tool belonging to these fields:

* IVM: main difference being maintaining the partially ordered set of increments over squashing all the updates (like IVM does).
* Incremental Dataflow: MapReduce and Dryad have been extended to support incremental computation, none of these supports iterative computations tho.
* Iterative dataflow: the tools shown here do not allow retraction of data, which is a massive no-no-no for us since we are gonna be working on data removal.

The paper concludes saying that DD is unique in allowing arbitrary nester iterative computations with general incremental updates. The paper also shows clear benefits in real life applications.

My doubts remain regarding how general can this solution be.

**Experimental metrics and scenarios:**

The tests and evaluation are run on computing a strongly connected component structure of Twitter whole graph of user mention in Tweets over 1 sec increment. The task consists in finding how many independent (disconnected from the others) subgraphs are found in the whole graph created by the user mentions in the tweets.

In the paper they often mention hops/iterations. These refer to how many hops are performed from a node to eventually find out that is does stop having neighbours at some point, finding a graph border and potentially identifying the subgraph in which that point is as one of the independent components that we're trying to count.

A second series of tests have been run on a Category B web-graph from ClueWeb using different unspecified graph algorithms. In this case they’ve also leveraged data-parallelism.

**Benchmarked against:**

The differential dataflow implementation is compared against:

* Stateless traditional computation
* Incremental computation (IVM)
* Incremental with prioritized computation (IVM with prioritization over computation, something related to this usecase)
* Differential Dataflow with 1 sec intervals (different from the 2 previous since stores every partial result, both for iteration and input data update)

**Limitations:**

**Repo:**

**Authors:**

**Year:**