Noria: Partially Stateful Data-flow for Read Heavy Web Applications

Jon Gjengset  Malte Schwarzkopf  Jonathan Behrens Lara Timbo Ara  Martin Ek Eddie Kohler M. Frans Kaashoek Robert Morris
Challenges of Read Heavy Web Apps

- Repeat reads for complex queries
- De-normalise a relational database: complicates writes, hard to maintain
- In-memory key-value cache (e.g. memcached), difficult to get efficient writes
- Stream processing system (e.g. Twitter’s Heron) not general, hard to reconfigure
Noria’s Solution

- Data-flow model with DAG composed of relational operators
- Noria introduces three innovations:
  
  A ‘partially stateful’ dataflow model

  Automatic merge and reuse of data-flow subgraphs over multiple queries

  Fast, dynamic transitions for data-flow graphs in the presence of new queries and schema changes
Dataflow Design

- Roots of the DAG are base tables
- External views are at the leaves
- Internal views are represented by relational operators
- Updates are first applied to the base table and then propagate through the data-flow graph as deltas
- Join operators use an upquery to process updates - better than just keeping windowed state
- Some operators (e.g. projection, filter) are stateless, while some (e.g. count, min/max) are stateful to avoid redundant recomputation
Partial State: Challenges and Opportunities

- Problem with stateful operators: leads to potentially unbounded state
- Partial state, based around *partially materialised views* in databases allow operators to only contain a subset of their overall state
- Introduces a new dataflow message: eviction notices
Partial State: Challenges and Opportunities

- If an operator is missing state, it will issue a recursive upquery
- Recursive upqueries introduce challenges around concurrency and correctness
- Start with empty state, lazily issue upqueries
- Only have partial state if can do index lookups
Dynamically Transitioning Dataflow

- Common for web applications to change query set overtime
- First stage of dataflow transition: plan what needs to be added to the dataflow graph, sharing and reusing operators wherever possible
- Then add operators into the graph to support new queries:
  - Stateless
  - Partially stateful
  - Fully stateful
Implementation

- 45k lines of Rust, RocksDB for persistent base tables
- Sharding on hash partition on key, TCP interconnect
- Two pools of worker threads: some to process updates, some to serve external views
- MySQL adapter
(a) With partial materialization and reuse (Zipfian).

(b) With partial materialization and reuse (uniform).

(c) No reuse or partial materialization (Zipfian).
Pros and Cons of the System

- Seems very easy to integrate with existing web apps
- Read performance very good for non-uniform
- See biggest performance benefits with Zipfian distributions: how representative is this of other applications?
- Recursive upqueries limit concurrency and complicate design
Questions