Noria

Dynamic, partially-stateful data-flow for high-performance web applications


Venue: OSDI 2018
Background and motivation

The problem addressed

• **Web applications** – long lived, low latency and often with changing queries.

• **Pre-computation** – difficult for both writes and reads.

• **Eventual consistency** – often sufficient.

• **Downtime at change** – needed in most data-flow systems, undesirable for web.
Noria’s novelty
Contributions of the paper

1. A partially-stateful data-flow model.
2. Techniques to automatically merge and reuse data-flow subgraphs.
3. Quick, dynamic response to a change of schema without downtime.
4. Prototype implementation and evaluation.
Noria data-flow design

- SQL interface, data-flow underneath.
- Directed acyclic graph of operators with:
  - Root — persistent store; on disk.
  - Leaves — derived external views; on server.

Noria: stateful data-flow operators pre-compute data for reads incrementally; data-flow change supports new queries.
Partial statefulness
Definition and properties

- **Partially-stateful model**: operators maintain only a subset of their state.
- **Missing records**: derived when needed via upqueries.
- **New operator**: initially empty, but starts processing immediately due to upqueries.
- **Descendants**: partial-state operators cannot have full-state descendants.
- **Rarely-used states**: evicted to reduce size and write load.
Eviction and upqueries
Mechanisms to ensure invariants hold

- **Eviction notices** — state entries that will no longer be updated.
  - Updates for evicted entries are dropped by operators.
  - Issued at random when approaching the memory ceiling.

- **Recursive upqueries**
  - Requests for records from stateful ancestors.
  - Eventually-consistent results.
Implementation
Noria’s development and usage

- **Rust-based** + RocksDB
- **Server setup** — runs on 1+ multicore servers.
- **Sharded data-flow** — across operators; no global coordination.
- **Easy integration** — MySQL adapter.
- **Noria-native applications** — best performance.

Source: https://github.com/mit-pdos/noria/graphs/contributors
Lobsters is a news aggregator, where users vote for stories.

Noria outperforms other (realistic) systems.

Uniform is not realistic…

**Figure 8:** For a uniformly-distributed, read-heavy (95%/5%) workload on Figure 2, Noria performs similarly to the (unrealistic) memcached-only setup.
Evaluation on Lobsters

Zipf-distributed story ID

- 95/5% representative for many web applications.
- Up to 70x higher throughput compared to realistic systems.

(a) Read-heavy workload (95%/5%): Noria outperforms all other systems (all but memcached at 100–200k requests/sec).

(b) Mixed read-write workload (50%/50%): Noria outperforms all systems but memcached (others are at 20k requests/sec).
Limitations
Design and prototype problems

- Requires a centralised timestamp signer.
- Lacks support for parameterised range queries.
- Lacks support for multi-column joins.
- Only suits apps compatible with eventual consistency.
- Is inefficient for sharded queries that require shuffles.