Eddies: Continuously Adaptive Query Processing
Ron Avnur, Joseph M. Hellerstein

Presented by:
Bojan Jovanovic

Eddies: Overview
- No rigid tree-structured query plan
- Adaptive: optimization during execution
- Continuously reorder operators in a query plan
- Schedule fast and selective operators first
- Each tuple scheduled separately
- Trade off between adaptivity and extra overhead

Eddies: Motivation
- Widely fluctuating data arrival rates – bursty performance
- Fluctuating data characteristics – selectivity estimation hard for data with no statistical summaries
- Continuous queries change while they execute
- Variations in cost and selectivity of operators
Eddies: Architectural Assumptions

- Non-distributed single site query executor
- Data and indexes possibly from external sources
- No shared processing – single query only
- Initial query plan constructed by a naïve preoptimizer

Join Operators: The Good, the Bad and the Ugly

The Bad:
- Hybrid hash join
  - State can be as large as the input
  - Requires re-computation if plan reordered
- Synchronization barriers limit concurrency
  - Example: merge join on two sorted duplicate-free inputs

The Good:
- Moments of symmetry
  - When synchronization barriers disappear
  - Example: nested loop joins
- Pipelined hash join has no barriers and constant symmetry
Join Operators: The Good, the bad and the ugly

The Ugly:

- In n-ary join un-indexed inputs must come before indexed inputs
- Physical property of an index relation: indexes can be probed but not scanned

Eddies: Ripple Joins

- Replace traditional iteration, hashing and indexing joins with their ripple join variants

Eddies: Pre Optimization

- Reordering tables among joins
- Picking spanning tree of a query graph
  - Form a chain of Cartesian products across small tables
  - Pick arbitrary equijoin edges (low selectivity)
  - Pick as many arbitrary non-equijoin edges as required
- For equijoin operators use index ripple join or hash ripple join. For non-equijoin use block.
**Eddies: What is and eddy?**

- N-ary tuple router between n-data sources and a set of query processing operators

**Eddies: An eddy in the river**

- Merge multiple unary and binary operators into a single n-ary operator
- Each tuple contains a vector or ready bits and done bits
- Tuples sent to operators whose ready bit set
- After a tuple returns appropriate done bit set
- When all done bits set, tuple sent to output

**Eddies: An eddy in the river**

- Ready bits are turned on according to constraints
- Binary operators
  - output tuples correspond to combinations of input tuples
  - Done and ready bits are ORed
Eddies: Routing tuples in eddies

- First, tuples given low priority
- After they return from operators priority increases
- Operator takes a ready tuple of highest priority
- Therefore, eddy is not clogged with new tuples

Eddy Test: Fluid dynamics and Operator Costs

- SELECT * FROM U WHERE s1() AND s2()
- Fluid dynamics arising from different operator costs

Eddy Test: Learning Selectivities

- Lottey Scheduling
  - Operators credited tickets when given tuples
  - Operators debited tickets when tuples returned
  - Chance of getting a tuple depends on number of tickets
Eddies Test: Joins

- SELECT * FROM R,S,T WHERE R.a = S.a AND S.b = T.b

Eddies Test: Responding to dynamic fluctuations

- Use window scheme to forget history
  - Banked tickets and escrow tickets
- Index Join – first I_Fast and I_Slow slow. After 30 sec I_Fast and I_Slow slow

Eddies Test: Delayed delivery

- Input on R initially delayed
Eddies: Extended to multi-way joins

  - Join operators decomposed into State Modules or SteMs
  - Motivation to decouple physical operations encapsulated within join modules
  - Data flow between SteMs is managed by an Eddy routing operator
  - Shared materialization point. Multiple access methods share results. Used by multiple competing join operators.

Eddies: Extended to multi-way joins

- Arbitrary routing not allowed. A set of constraints necessary to guarantee correctness.

Eddies: Applied to continuous queries

Eddies: Conclusions and Discussion

- Major Contributions / Good Points
  - First general query processing scheme for reordering in-flight operators
  - Novel characterization of synchronization barriers and moments of symmetry
- Bad Points
  - Extra overhead compared to static queries
  - Does not handle delayed data well
- Points for discussion:
  - Other scheduling policies?
  - How can delayed data be handled?
  - Better Pre-optimizer?