Eddies: Continuously Adaptive Query Processing

Ran Avnur, Jeseph M. Hellestein
University of California, Berkeley

CPSC 405 Data Management
Presented by Hongrae Lee

Outline

- Introduction
- Reorderability of plans
- Rivers and Eddies
- Routing tuples in Eddies
- Summary

Static Query Processing

- Traditional query processing scheme
  1. Optimizing a query
  2. Executing a static query plan

This traditional scheme is not appropriate for Large scale widely-distributed information resources or Massively parallel database systems!

New Requirements

- Increased complexity in large-scale system
  - Hardware and workload
  - Data
  - User interface
- We want query execution plans
  - To be reoptimized regularly during query processing
  - Allowing the system to adapt dynamically to fluctuations in computing resources, data characteristics, and user preferences

Discussion Question 1

The Philosophy: "We Favor adaptivity Over Best-case Performance"

Consider if adaptivity is needed only when the best-case is missing (unable established for lack of statistics, or non-existence because of changing environment) or could also be a general strategy in regular query processing. Do you think it is good or bad to apply it in the traditional query processing? Why? Please give reasons or use examples to support your opinions.
Two Challenges for This Scheme
- How can we reorder operators?
  - Reorderability of plans
- How should we route tuples?
  - Routing tuples in Eddies

Reorderability of Plans
- Synchronization Barriers
  - One task waits for other tasks to be finished
- Moments of Symmetry
  - The barrier where the order of the inputs to a join can be changed without modifying any state in the join

Reordering of Inputs Using Moments on Symmetry
- Moments on symmetry
  - Allow reordering of the inputs to a single binary operator
  - Generalization
    - N-ary join view
      - \((R \bowtie S) \bowtie T \rightarrow (R \bowtie T) \bowtie S\)
      - \((T \bowtie S) \bowtie R \rightarrow (T \bowtie R) \bowtie S\)
  - Commutativity + moments of symmetry \(\Rightarrow\) aggressive reordering of a plan is possible

Join Algorithms and Reordering
- Constraints on reordering
  - Unindexed join input is ordered before the indexed input
  - Preserving the ordered inputs
  - Some join algorithms work only for equijoins
- Join algorithms in Eddies
  - We favor join algorithms with
    - Frequent moments of symmetry
    - Adaptive or nonexistent barriers
    - Minimal ordering constraints
    - \(\Rightarrow\) Rules out hybrid hash join, merge joins, and nested loops joins
  - Choice: Ripple Join
    - Frequently-symmetric versions of traditional iteration, hashing and indexing schemes
  - Favors adaptivity over best-case performance
Ripple Joins

- Have moments of symmetry at each corner
- Are designed to allow changing rates for each input
- Offer attractive adaptivity features at modest overhead

Rivers and Eddies

River
- A shared-nothing parallel query processing framework
- Pre-optimization
  - Choose how to initially pair off relations into joins

An eddy in the River
- Is implemented via a module in a river
- Encapsulates the scheduling of its participating operators
- Explicitly merges multiple unary and binary operators into a single n-ary operator
- A tuple is associated a vector of Ready and Done bits

Routing Tuples in Eddies

An eddy module
- Directs the flow of tuples from the inputs through the various operators to the output
- Provides the flexibility to allow each tuple to be routed individually through the operators
- The routing policy determines the efficiency

Naïve eddy

- Tuples enter the eddy with low priority, and when they are returned to the eddy from an operator they are given high priority
  - Tuples flow completely through the eddy before new tuples
  - Prevents being ‘clogged’ with new tuples
- Fixed-size queue: back-pressure
  - Production along the input to any edge is limited by the rate of consumption at the output
  - Tuples are routed to the low-cost operator first
- Cost-aware policy
- Selectivity-unaware policy

Learning Selectivity: Lottery Scheduling

- To track both
  - Consumption (determined by cost)
  - Production (determined by cost and selectivity)
- Lottery Scheduling
  - Maintain ‘tickets’ for an operator
    - An operator’s chance of receiving the tuple
      - The counts of tickets
      - The eddy can track (learn) an ordering of the operators that gives good overall efficiency

Some Experimental Results
Summary

- Eddies are
  - A query processing mechanism that allow fine-grained, adaptive, online optimization
  - Beneficial in the unpredictable query processing environments

- Challenges
  - To develop eddy 'ticket' policies that can be formally proved to converge quickly
  - To attack the remaining static aspects
  - To harness the parallelism and adaptivity available to us in rivers
  - To explore the application of eddies and rivers to the generic space of dataflow programming

Discussion Question 2

Comparison among traditional query processing, Tukwila, and Eddy

The adaptivity and complexity of Eddy, Tukwila, and traditional query processing vary. Each of them has its beauties and can not be replaced by others.

As a designer of query processing and optimization, which one would you like to use? Why?

Thank you