Eddies: Continuously Adaptive Query Processing

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

CPSC 504 Data Management
Presented by Hongrae Lee

Outline

- Introduction
- Reordability 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*
  - To allow system to adapt dynamically to fluctuations in computing resources, data characteristics, and user preferences

Group Discussion
(~4 per group)

(Credit to Jianhao)

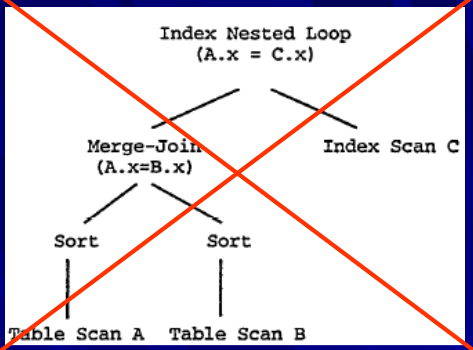
What are pros and cons of centralized query execution vs sending partial queries to the data sources? (Assuming query execution capability.)

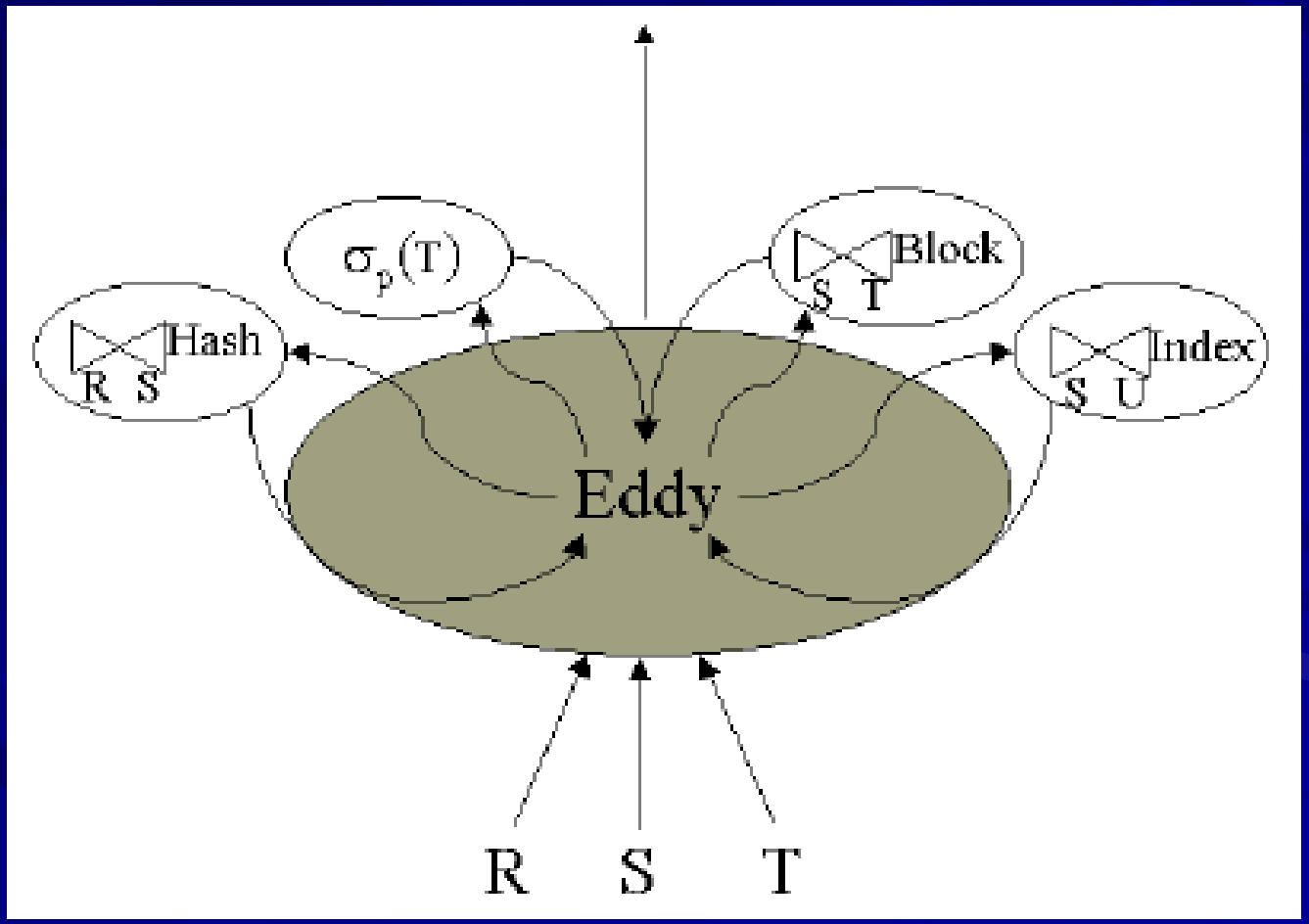
For example, data sources may have the stats for query optimization.

(Credit to Sid)

Conversely, what if the stats about the data were also sent from the data sources? What are some potential pitfalls of such an approach?

Eddy


Index Nested Loop
(A.x = C.x)
Merge-Join
(A.x=B.x)
Index Scan C
Sort
Sort
Table Scan A
Table Scan B

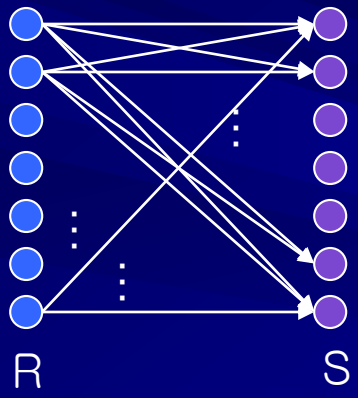

$\sigma_p(T)$
Block
S T
Hash
R S
Index
S U
Eddy
R
S
T

Two Challenges for This Scheme

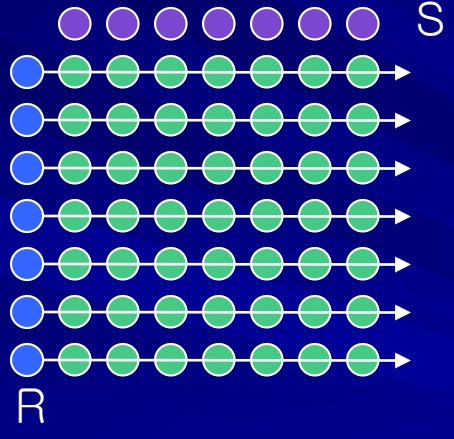
- How can we reorder operators?
  - Reorderability of plans
- How should we route tuples?
  - Routing tuples in Eddies

A Brief Review on Join

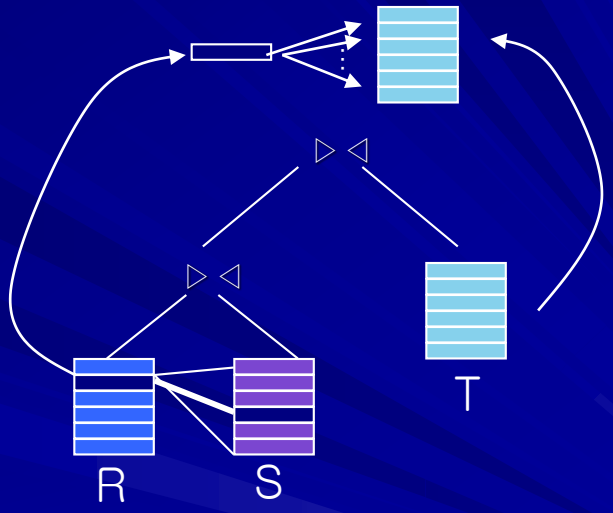
- R ⋈ S



Basic nested loop join



Grid view of nested loop join

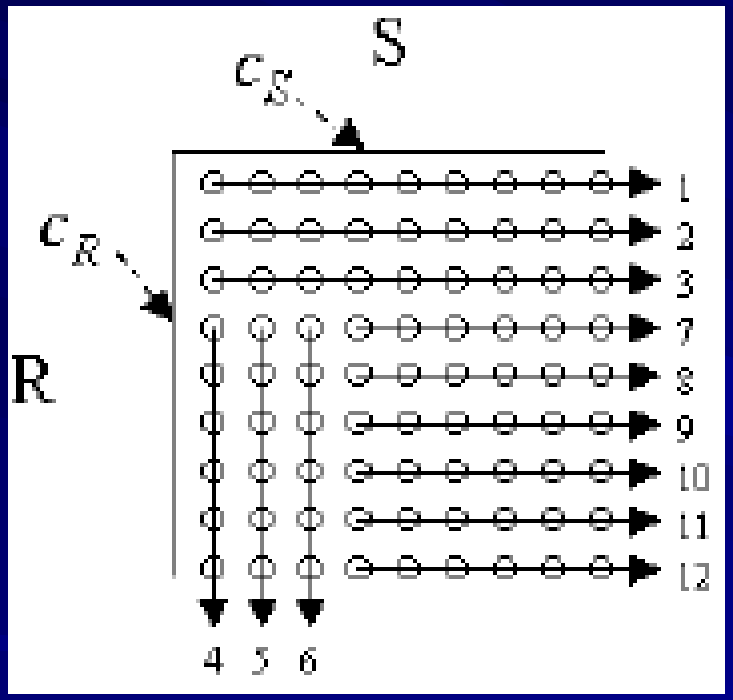


Pipelining

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



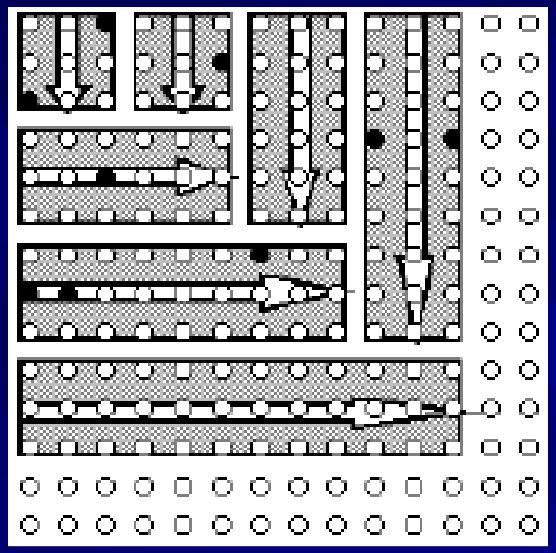
- Moments on symmetry
  - Allow reordering of the inputs to a single binary operator
    - $R \bowtie S \leftrightarrow S \bowtie R$
- Generalization
  - N-ary join view
  - $(R \bowtie_1 S) \bowtie_2 T \rightarrow (R \bowtie_2 T) \bowtie_1 S$
  - $\rightarrow (T \bowtie_2 R) \bowtie_1 S$
- Commutativity + moments of symmetry → aggressive reordering of a plan is possible

Join Algorithms and Reordering

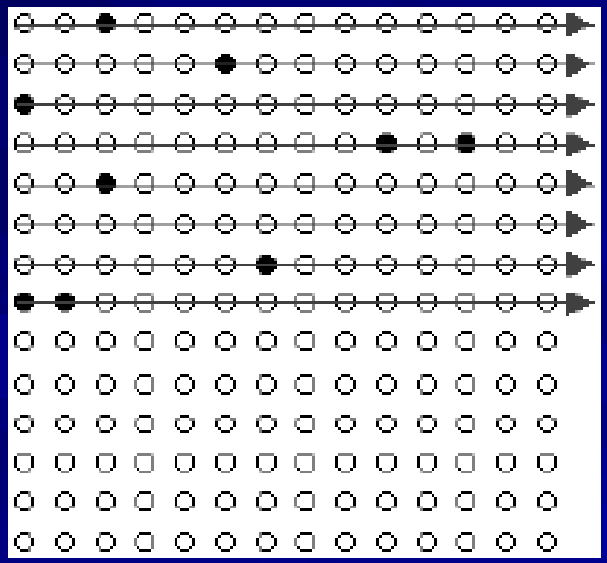
- Constraints on reordering
  - Unindexed join input is ordered before the indexed input
  - Preserving the ordered inputs
  - Some join algorithms work only for equijoins
- Join algorithms in Eddy
  - We favor join algorithms with
    - Frequent moments of symmetry
    - Adaptive or nonexistent barriers
    - Minimal ordering constraints
    - →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

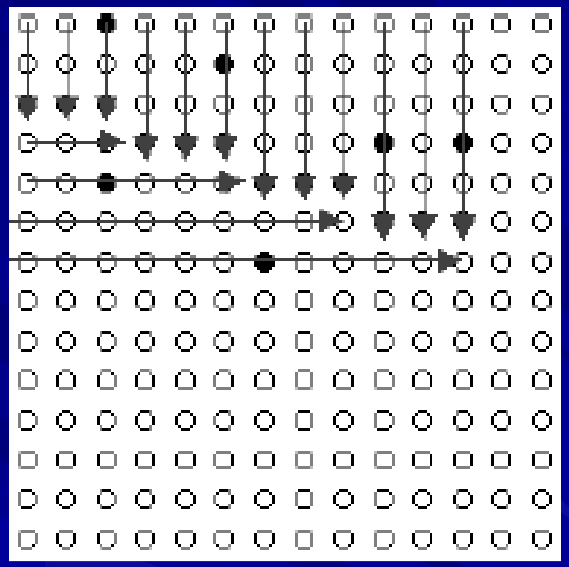
Block



Index



Hash



- 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

Pair Discussion
(with neighbours)

(Credit to Ji Tong Yin)

Rivers, eddies, ripples…

Many of the papers we've read have had friendly names. Tukwila is a city in Washington. An eddy is circular flowing water.

What are some interesting or effective strategies for naming systems, programs, or processes?

Routing Tuples in Eddies

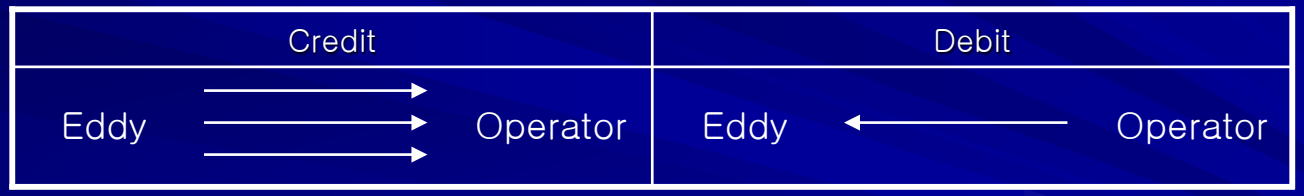
- An eddy module
  - Directs the flow of tuples from the inputs through the various operators to the output
  - Providing the flexibility to allow each tuple to be routed individually through the operators
  - The routing policy determines the efficiency

Naïve eddy

- Naïve eddy
  - Tuples enter eddy with low priority, and when returned to eddy from an operator are given high priority
    - → Tuples flow completely through 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

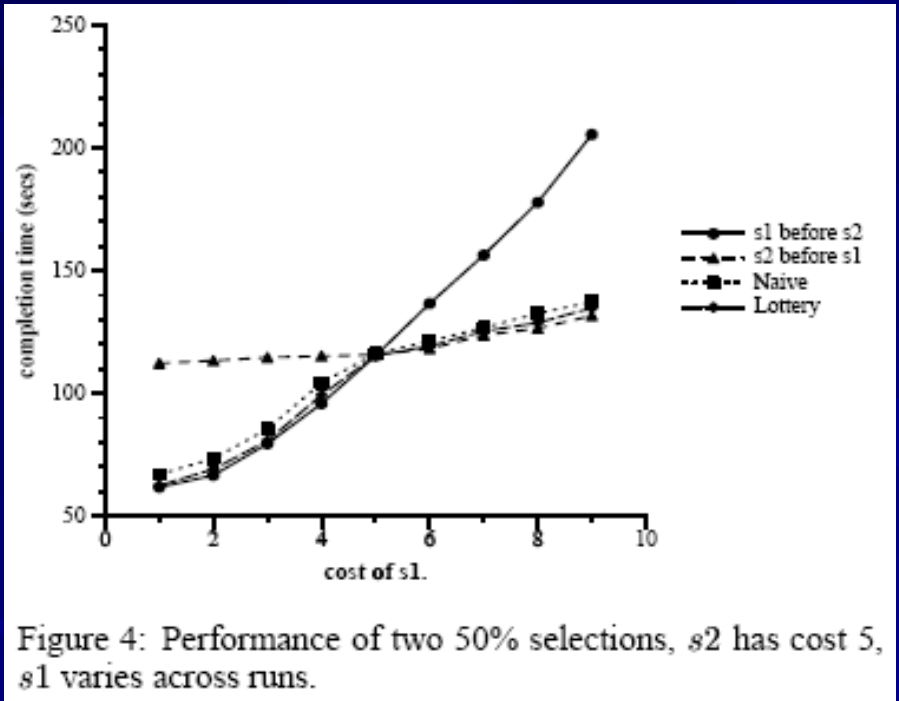


Figure 4: Performance of two 50% selections, $s2$ has cost 5, $s1$ varies across runs.

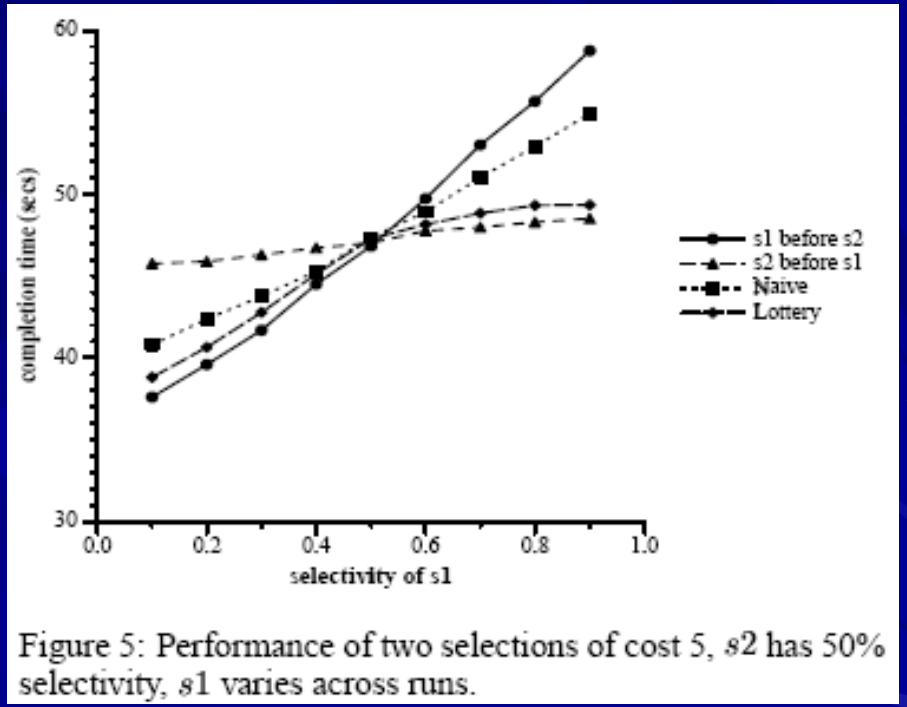


Figure 5: Performance of two selections of cost 5, $s2$ has 50% selectivity, $s1$ varies across runs.

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

Thank you

Group Discussion
(~4 per group)

(Credit style to Matt)

It's 2023 and data has exploded in quantity, with a population expecting to have instant access to all of it over a massive global network of connected computers.

Do we favour Tukwila or Eddies? Why?

Final Discussion

(Credit to Sarah)

The authors of "Eddies" mention that part of the aim was attempting to do away with traditional optimizers entirely. Considering research and innovation in general, what are some advantages and disadvantages of attempting to replace traditional approaches with entirely new methods? What are some instances where this may be necessary and others where it is unlikely to work?