Eddies: Continuously Adaptive Query Processing

Jae Kyu Chun

Feb. 17, 2003

Query in Large Scale System

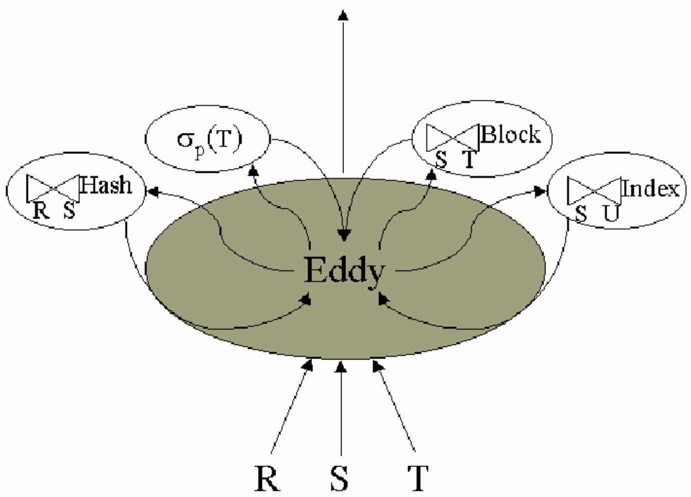
- Hardware and Workload Complexity
  - heterogeneous hardware mix
  - unpredictable hardware performance
- Data Complexity
  - Static cost estimates become unreliable
- User Interface Complexity

➢ *Query Processing should be adaptive.*

Steps for a typical Query Processor

- Express query as algebra expression (set of operators)
- Enumerate alternative plans for evaluating the expression using *equivalence rules, access methods, and implementation algorithms*
- For each alternative plan, estimate the cost of each enumerated plan
- Choose the plan with the least estimate cost

Eddies



- Continuously reorders the application of pipelined operators in a query plan, on a tuple-by-tuple basis.
- Data flows into the eddy from input relations R, S and T
- The eddy routes tuples to operators: the operators run as independent threads, returning tuples to the eddy
- The eddy sends a tuple to the output only when it has been handled by all the operators
- The eddy adaptively chooses an order to route each tuples through the operators

Eddies

- Traditional Query Processor (System R)
  - Frequency : batch (daily/weekly)
  - Effect : all aspects of Query Processing

- Eddies
  - Frequency : per tuple
  - Effect : reordering of pipelined operators

Eddies

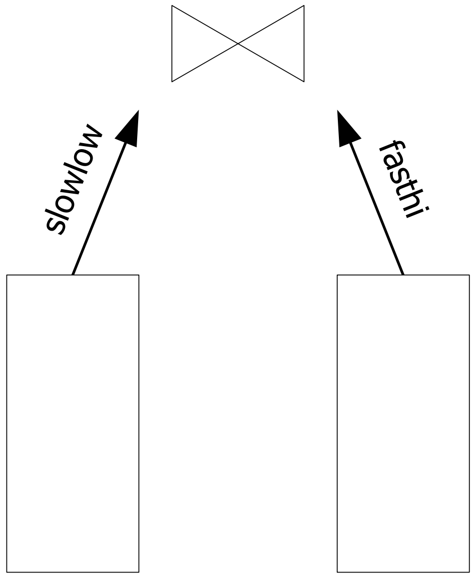
- Reordering operators during runtime sounds cool.

- Will it always work?
  - Synchronization Barrier
  - Moments of Symmetry

Synchronization Barriers

- The processing of fasthi is postponed for a long time while consuming many tuples from slowlow.

- Favor minimal barriers.

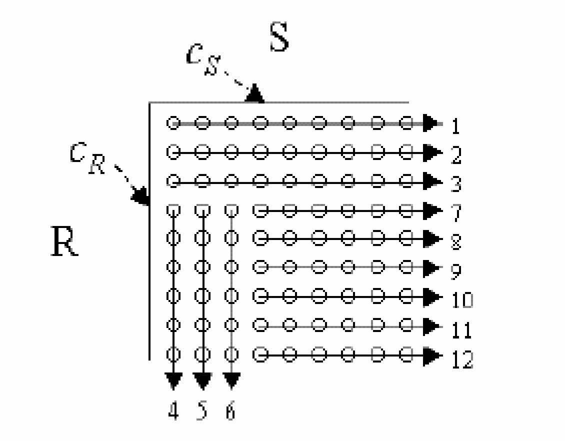


Moments of Symmetry

- The order of the inputs to the join can often be changed without modifying any state in the join.
- Commutativity of operator

  \+ Moments of symmetry

  ⇒ reordering of a plan tree



Join Algorithms

- Nest Loop Join
  - Moments of Symmetry : End of each inner loop
  - Synchronization Barrier : End of each inner loop
- Merge Join
  - Moments of Symmetry : Symmetric
  - Synchronization Barrier : data dependent
- Hybrid Hash Join
  - Moments of Symmetry : none
  - Synchronization Barrier : none

Join Algorithms and Reordering

- In order for an eddy to be most effective
  - Frequent moments of symmetry
  - Adaptive or non-existent barriers
  - Minimal ordering constraints
- Ripple joins offer very frequent moments of symmetry and attractive adaptivity.

Routing Tuples in Eddies

- An eddy's tuple buffer is implemented as a priority queue with a flexible prioritization scheme.
- An operator is given the highest-priority tuple in the buffer that has the corresponding Ready bit set.

Naïve Eddy

- The query operator with low cost will quickly process its input tuple and is ready to process another.

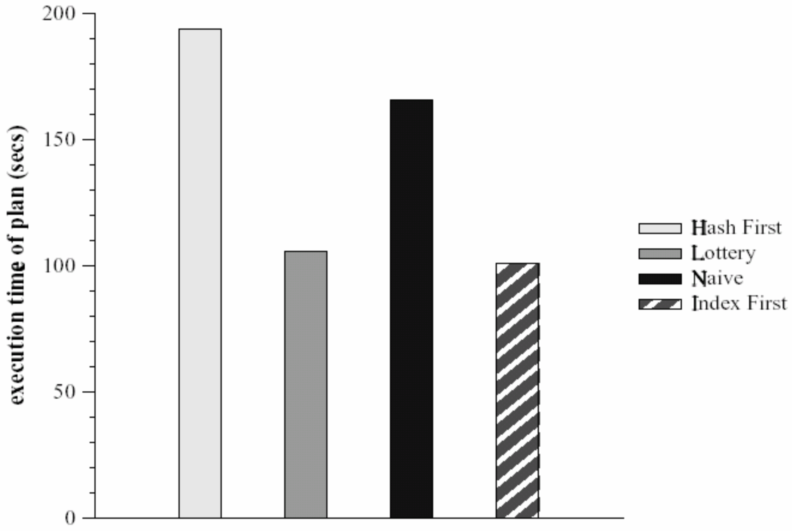
  -> The consumption rate of low cost operator is higher than that of high cost operator.
- In case where the cost of the query varies, the eddy performs better then the possible static plans.
- Not suitable for the cases where selectivity of the operators are varied.

Fast Eddy

- Gives priority to operators with low cost and low selectivity.
  - Lottery Scheduling
- Each time the eddy gives a tuple to an operator, it credits the operator one “ticket”.
  - -> favor low cost
- Each time the operator returns a tuple to the eddy, one ticket is debited from the eddy’s running count for that operator.
  - -> favor low selectivity

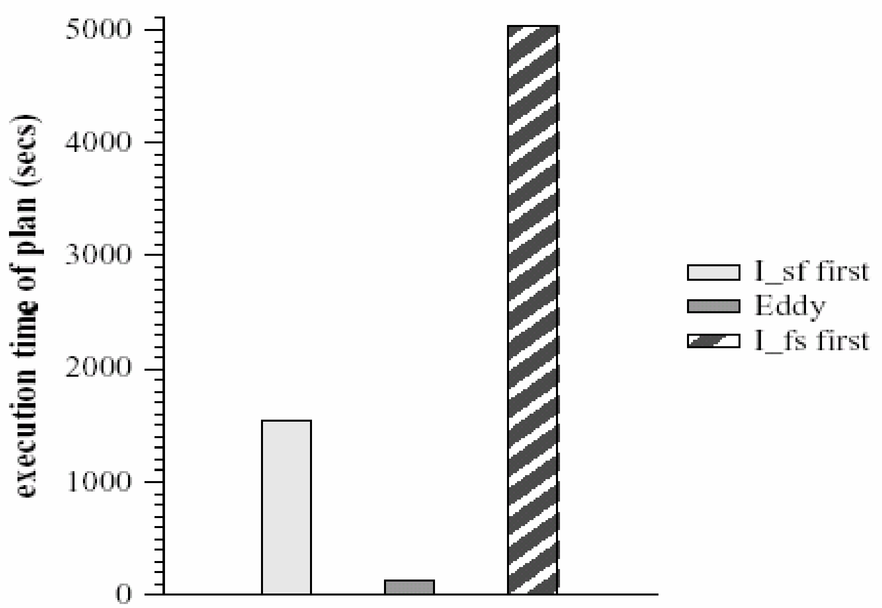
Performance of two Joins

- 3 table query: hash ripple join between R and S, and an index join between S and T
- Eddy do well in even in static scenarios
- Perform nearly optimally

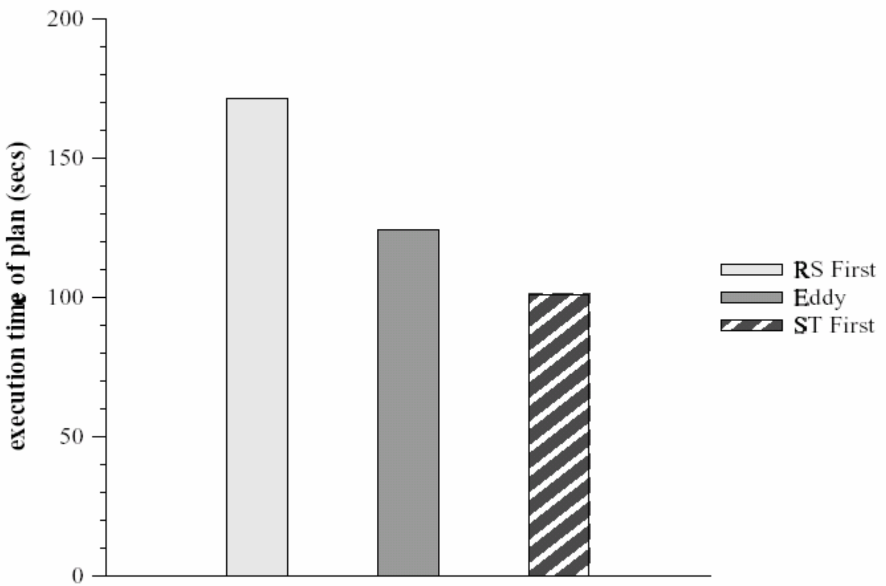


Changing Join Cost

- Two index joins
  - Slow: 5 second delay, Fast: no time delay
  - Swap speeds after 30 seconds



Delayed Delivery



Delay in delivery of R by 10 Seconds
RS selectivity at 100 %,
ST selectivity at 20 %

- Eddy does not adapt to initial delays of R
- RS join does not produce any output tuples during the early part of processing
  - Eddy awards most S tuples to the RS join initially
- Ticket scheme does not capture the growing selectivity inherent in a join with delayed input

Re-Optimization vs. Eddies

- Re-Optimization (Kabra, Dewitt) : Reordering queries at the end of pipelines
  - Reordering operators only after temporary results are materialized
- Eddies
  - Assumption : The choice of spanning tree, join algorithms and access methods are pre-determined.
  - Adaptively reorder pipelined operators on-the-fly.
  - Learning algorithm that adaptively learns how to route the tuples to the pipelined operators

Strength of Eddies

- Per tuples adaptivity : Be beneficial for rapidly changing, unpredictable environments
- Can be used in concert with existing optimizers to improve adaptability within pipelines.

Future Work

- Routing policy: adaptively converge quickly to optimal execution when conditions change
- Make adaptive the choices of spanning tree, join algorithms, and access methods.