SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

Eddies: Continuously Adaptive Query Processing

Ron Avnur

Joseph M. Hellerstein

Presented by: Samaneh Khakshour

CMPT 843

February 2017

Outline

- Motivation
  - What are traditional query optimizers
  - What are the problems with these optimizers
- What is eddy
  - How to reorder operators
  - Routing tuples in eddy
  - Experimental results
- Summary

Static Query Processing

- Traditional query processing scheme:
  - Query optimization
  - Static query execution

- This scheme is not good for widely-distributed resources and massively parallel database systems.

Why not Static Query Processing

- Large-scale systems should function robustly in an unpredictable environment with:
  - Hardware and workload complexity
  - Data complexity
  - User Interface complexity

- Query processing should be adaptive
  - Allowing the system to adapt dynamically to fluctuations in:
    - Computing resources
    - Data characteristics
    - User preferences

Run- Time Fluctuations

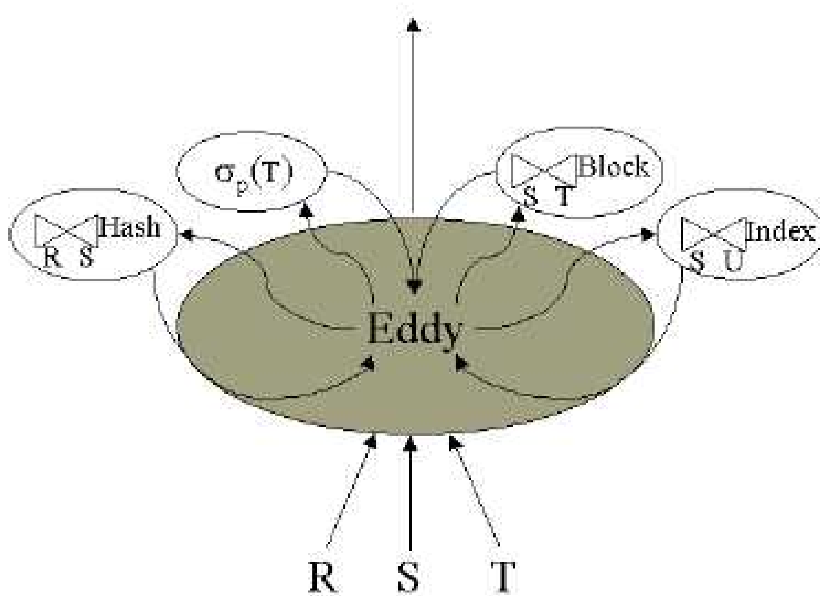
- Three properties can vary during query processing:
  - Cost of Operators
  - Operator Selectivity
  - Rate tuple arrive from input

- Example on run-time variation in selectivity:

- Consider an employee table clustered by ascending age, and a selection “salary>10000”:
  - Initially the predicate “salary>10000” will be very selective
  - Selectivity rate will change as the older employees are scanned.

Eddies

- Two Challenges with Eddies:
  - How can we reorder operators (Reorderability of plan)
  - How should we route tuples?



Reorderability of Plans

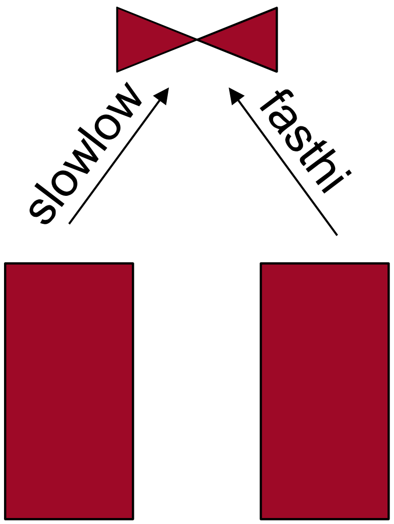
- Reoptimizing a query execution pipeline on the fly requires significant care in maintaining query execution state.

- Some things that Eddies must consider:
  - Adaptive or non existent synchronizations barriers
  - Frequent moment of symmetry

Synchronization Barriers

- One Operation hinder the speed of other operations

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

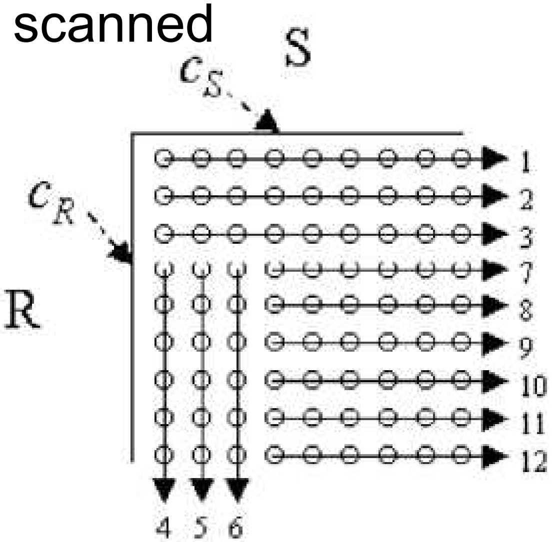


- Desirable to minimize the number of synchronization barriers

Moments of Symmetry

- When query is executed to a point that optimizer can change the query plan without affecting the way the query plans predicates are performed.

- Example:
- Consider a nested-loop join
- Outer relation R, and inner relation S
- Reordering the input (re-optimization):
  - Can happen when S is completely scanned
  - $R \bowtie S = S \bowtie R$



- Combination of
  - Commutativity of operators
  - Moment of symmetry
  - Reordering of a plan

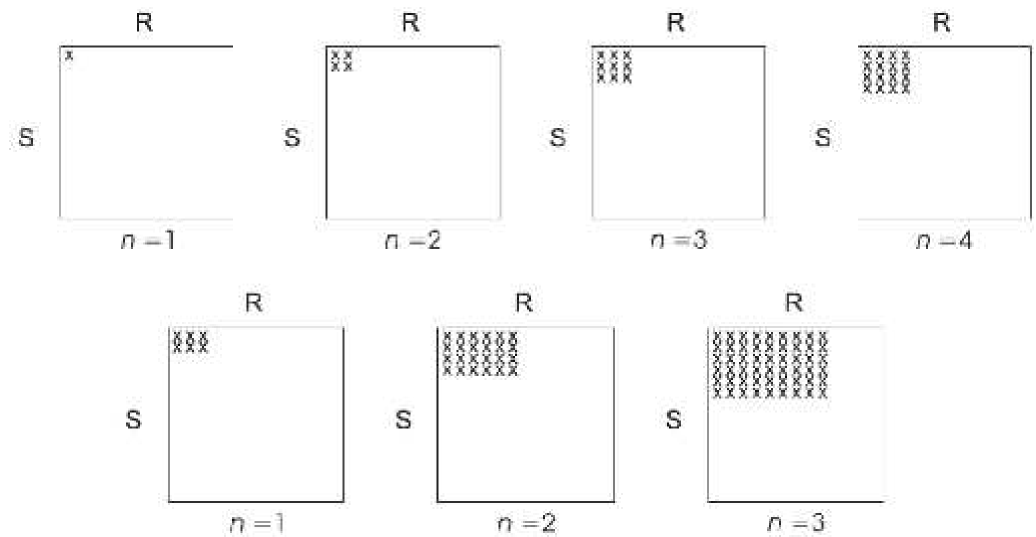
Join Algorithms and Reordering

- Join ordering constrains:
  - Unindexed join input is ordered before the indexed input
  - Preserving the ordered inputs
  - Some join algorithm work only for equijoins

- In order for Eddies to be most effective:
  - Frequent moment of symmetry
  - Adaptive or non existent barriers
  - Minimal ordering constrains

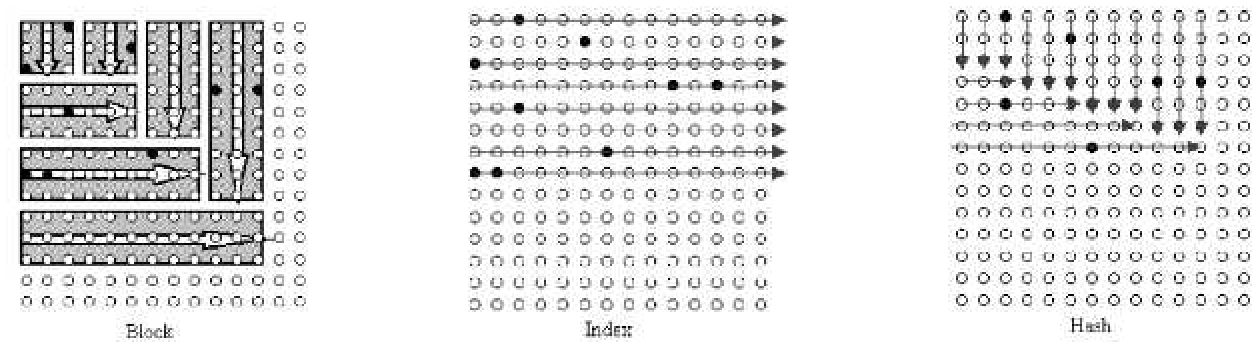
- Ripple joins are having frequent moments of symmetry and attractive adaptivity.

Ripple Join



- Get tuples from each relation
- Compare them with tuples seen until now

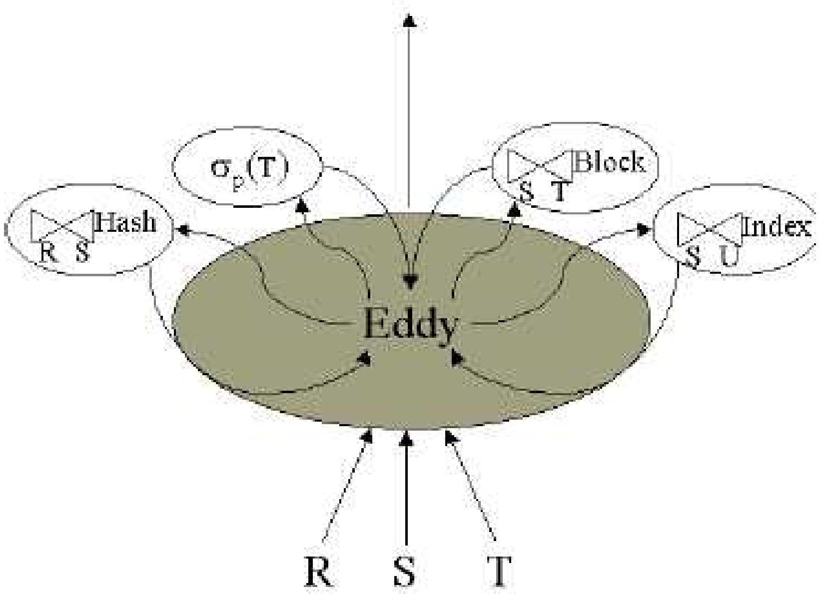
Ripple Join



- Ripple joins:
  - Have moments of symmetry at each corner
  - Are designed to allow changing rates for each input
  - Offer attractive adaptivity feature

Eddy

- River: A shared nothing parallel query processing framework.
- Eddy is implemented via a module in River, containing:
  - arbitrary number of input relations
  - A number of participating unary and binary modules
  - A single output relation
- Eddy was designed to dynamically reoptimize queries.
- Eddy maintains a fixed size buffer of tuples that need to be processed



Eddy

Motivation
- Traditional query optimizers
- Problems

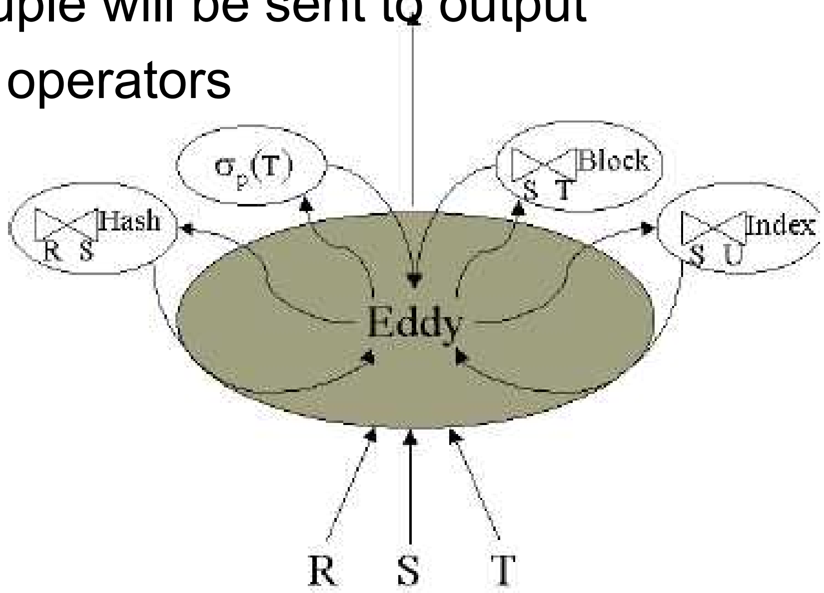
What is eddy
- Reordering operators
- **Eddies**
- Routing tuples
- Experimental results

Summary

- A tuple in eddy is associated with a descriptor:
  - A vector of Ready bits and Done bits

- The eddy forward the tuple to operators that have Ready bits turned on

- After an operator is processed, its Done bits are set
  - If all Done bits are set, tuple will be sent to output
  - Otherwise, sent to other operators



Routing Tuples in Eddies

- Eddy module:
  - Directs the flow of tuples from inputs through the various operators to the output
  - Providing the flexibility to allow each tuple to be routed individually through the operators

- Routing policy used in eddy determines the efficiency of the system

- In this paper multiple different ways of routing is studied.

Naïve Eddy

- Eddy's buffer is implemented as priority queue
- When a tuple enters buffer, its priority is set low
- After it is processed by an operation in Eddy and returned to buffer, its priority is set to high
- This ensure that tuples do not get clogged in the Eddy
- Eddy's queue size is fixed: back-pressure
  - Production along the input to any edge is limited by the rate of consumption at the output
  - Tuples are routed to low cost operators first
  - Consumption rate of low cost operator is higher than that of high cost operator.

- Cost aware policy
- Selectivity unaware policy

Lottery Scheme

- Gives priority to operators with low cost and low selectivity.
- Each time the eddy gives a tuple to an operator, it credits the operator one ticket.
  - Favor low cost (Track consumption)

- Each time the operator returns a tuple to eddy, one ticket is debited

  from eddy's running count for that operator
  - Favor low selectivity (Track production)

- Operators chance of receiving a tuple corresponds to its count of tickets.
- Eddy can track an ordering of operators that give good overall efficiency.

Window Scheme

- Problem with lottery scheme is that it uses too much of past information:
  - An operator gained a lot of tickets initially but then became slow
- Window scheme: the lottery scheme is modified such that the lottery only looks at tickets gained by an operator in a fix window
  - Keep track of two types of tickets:
    - Banked tickets: Used when running the lottery
    - Escrow tickets: Used to measure efficiency during the window
    - At the end of the window:
    - Banked tickets=Escrow tickets
    - Escrow tickets=0

- Ensures operators re-ensure themselves each window

Experimental Results

Motivation
- Traditional query optimizers
- Problems

What is eddy
- Reordering operators
- Eddies
- Routing tuples
- **Experimental results**

Summary

- ❖ Naïve approach naturally adjusts based on cost of operators
- ❖ Lottery also adjusts based on cost very well.

```
SELECT   *
  FROM   U
 WHERE   s1() AND s2();
```

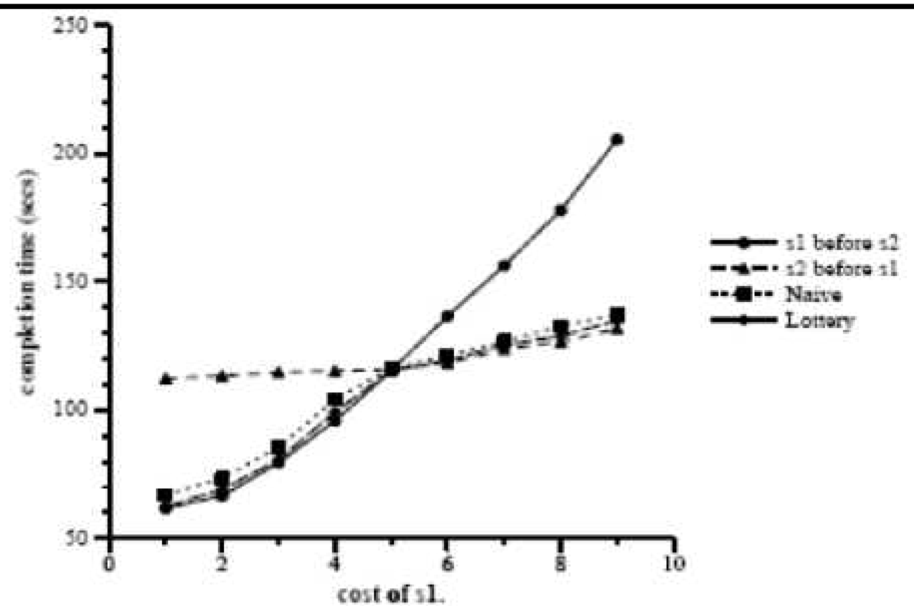


Figure 4: Performance of two 50% selections, $s2$ has cost 5, $s1$ varies across runs.

Experimental Results

- ❖ Naïve approach does not adjust based on selectivity
- ❖ Lottery also adjusts based on selectivity

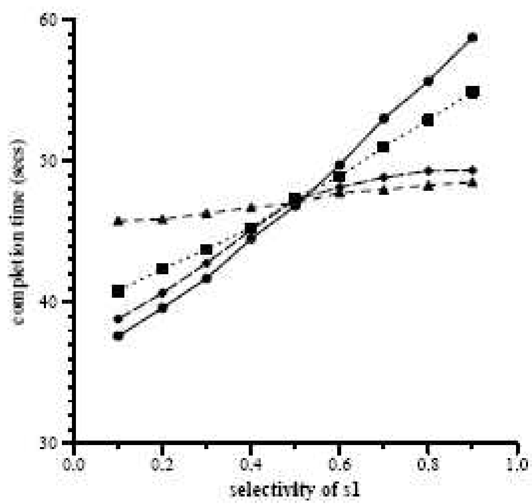


Figure 5: Performance of two selections of cost 5, $s2$ has 50% selectivity, $s1$ varies across runs.

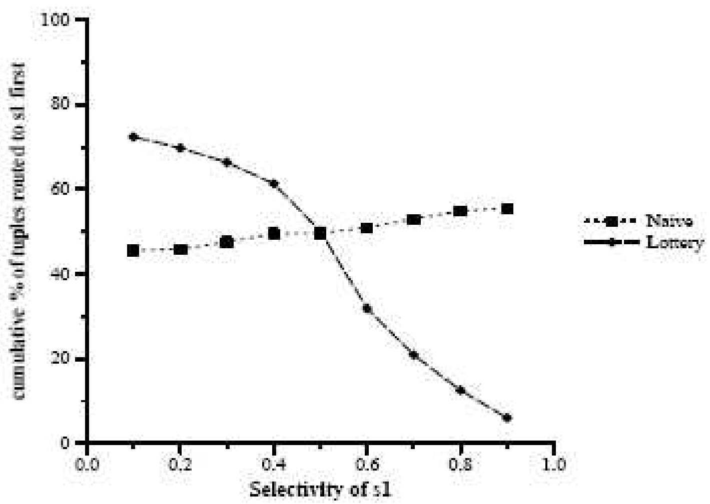


Figure 6: Tuple flow with lottery scheme for the variable-selectivity experiment(Figure 5).

Performance of Two Joins

- Naïve perform between best and worst
- Lottery performs nearly optimal

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

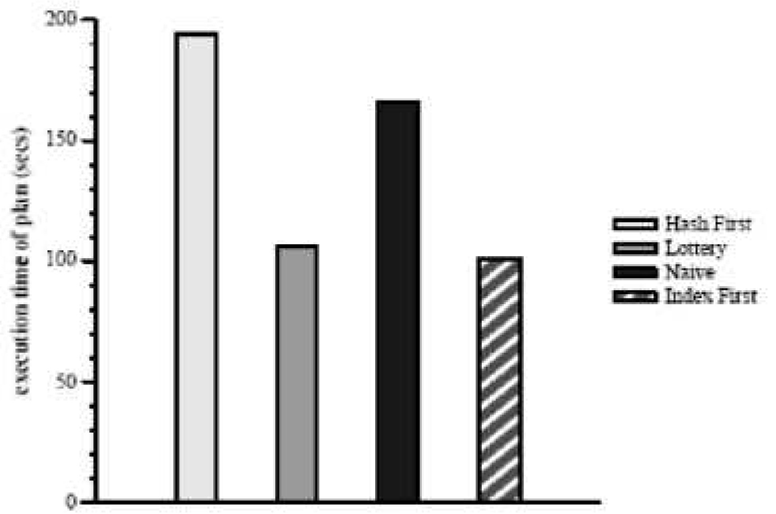


Figure 7: Performance of two joins: a selective Index Join and a Hash Join

Performance of Hash Joins

Motivation
- Traditional query optimizers
- Problems

What is eddy
- Reordering operators
- Eddies
- Routing tuples
- **Experimental results**

Summary

- All joins are ripple joins
  - Change selectivity of join predicate
  - In all cases eddy with lottery is close to optimal.

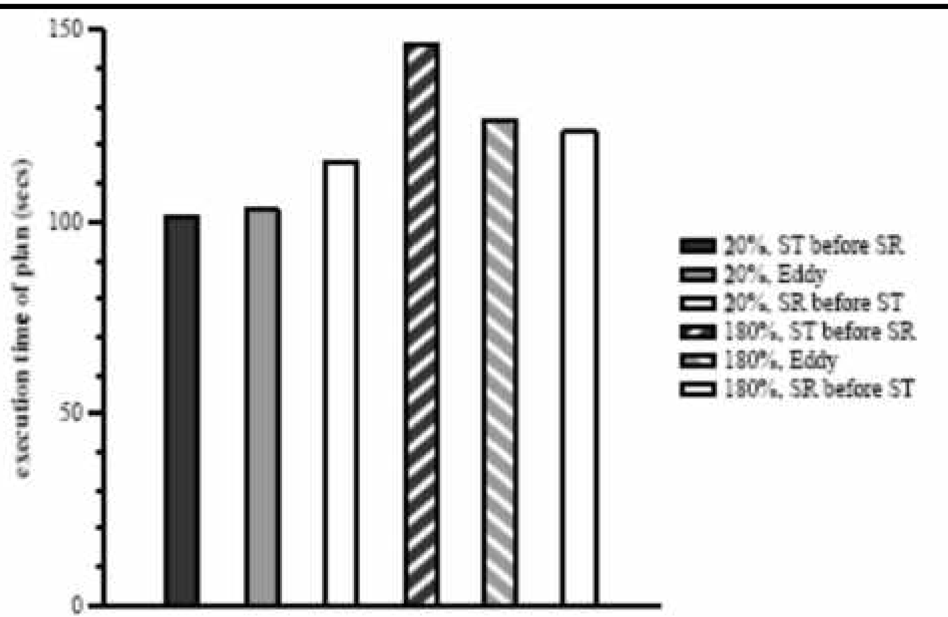


Figure 8: Performance of hash joins $R \bowtie S$ and $S \bowtie T$. $R \bowtie S$ has selectivity 100% w.r.t. S, the selectivity of $S \bowtie T$ w.r.t. S varies between 20% and 180% in the two runs.

Performance of another Join

- 3-table equijoin query, two tables are external and used as "inner" relation by index join.
- Third relation has 30,000 tuples.
  - Eddy outdoes both static plans

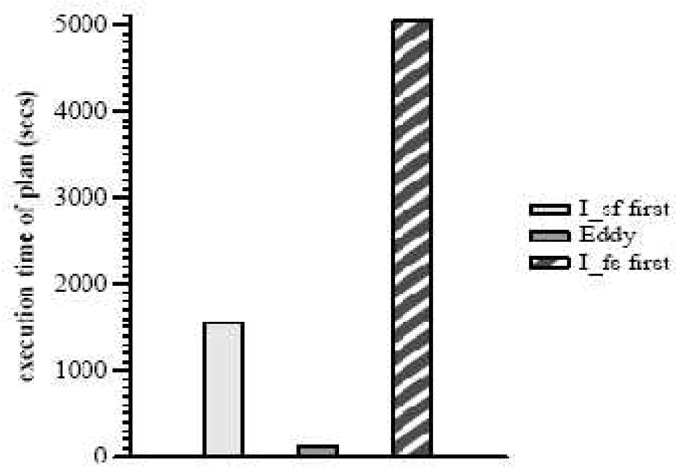


Figure 10: Adapting to changing join costs: performance

Adaptability

Motivation
- Traditional query optimizers
- Problems

What is eddy
- Reordering operators
- Eddies
- Routing tuples
- **Experimental results**

Summary

- Toggle the cost of operator 3 times throughout experiment
  - Eddy switches how many tuples are first processed by that operator

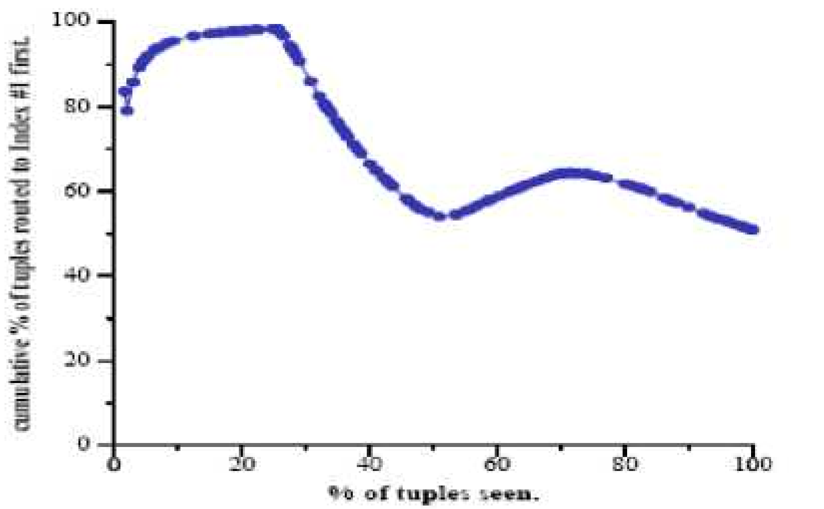


Figure 11: Adapting to changing join costs: tuple movement.

Delayed Delivery

Motivation
- Traditional query optimizers
- Problems

What is eddy
- Reordering operators
- Eddies
- Routing tuples
- **Experimental results**

Summary

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

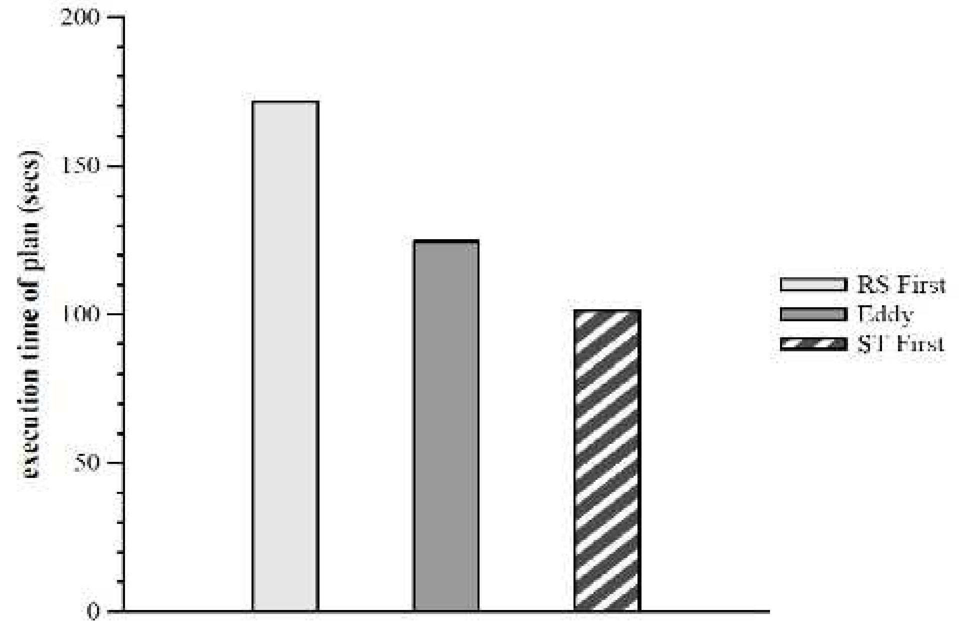


Figure 12: Adapting to an initial delay on R: performance

Problems

- Does not work well when there is initially a long delay at an operator
- Eddy gives all tuples to operator because operator is not returning tuples that satisfy the join predicates criteria (not until after the delay)
- Eventually this problem is figured out by eddy (Just may take some time).

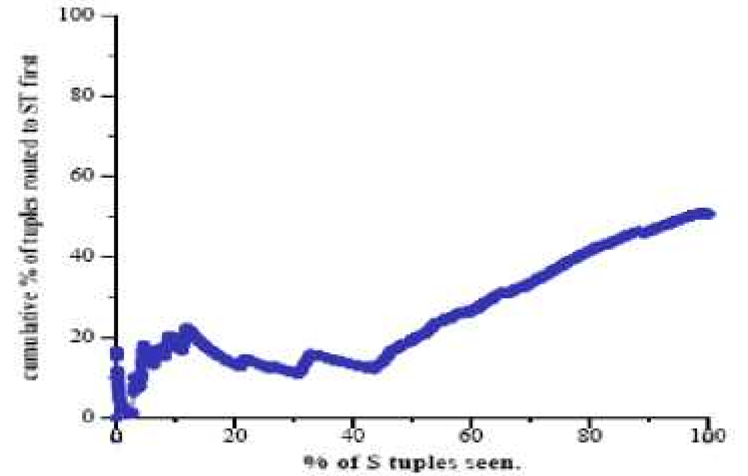


Figure 13: Adapting to an initial delay on R: tuple movement.

Re-optimization vs. Eddies

- Re-optimization (Kabra, Dewitt):
  - reordering queries at the end of pipelines
- Eddies:
  - Adaptively reorder pipelined operators on the fly
  - Learning algorithms that adaptively learns how to route the tuples to the pipelined operators

Strength of Eddy

- Per tuple adaptivity: Be beneficial for rapidly changing, unpredictable environments

- Can be used in concert with existing optimizers to improve adaptability within pipelines.

Summary

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

- Challenges:
  - Develop ticket policies that can formally proved to converge quickly
  - To attack the remaining static aspects
  - To harness the parallelism and adaptivity available to us in rivers

Questions and Comments

Thank you very much!