Eddies: Continuously Adaptive Query Processing

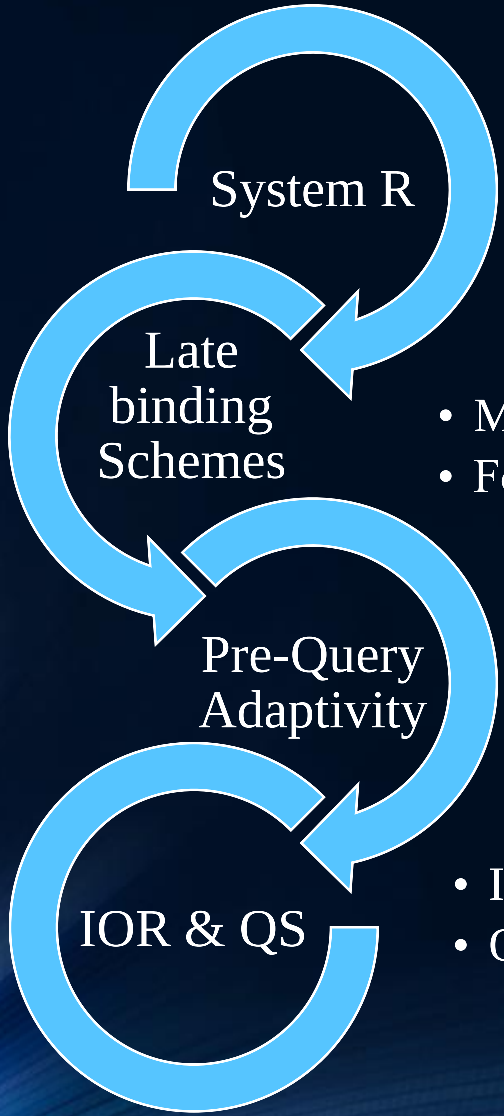
-RON AVNUR AND JOSEPH M. HELLERSTEIN, UC BERKELEY

-PRESENTED BY AADITYA SRINIVASAN

Outline

- Background
- Introduction
- Reorderability of plans
- Rivers and Eddies
- Routing Tuples in Eddies
- Conclusion and Future Work

Background



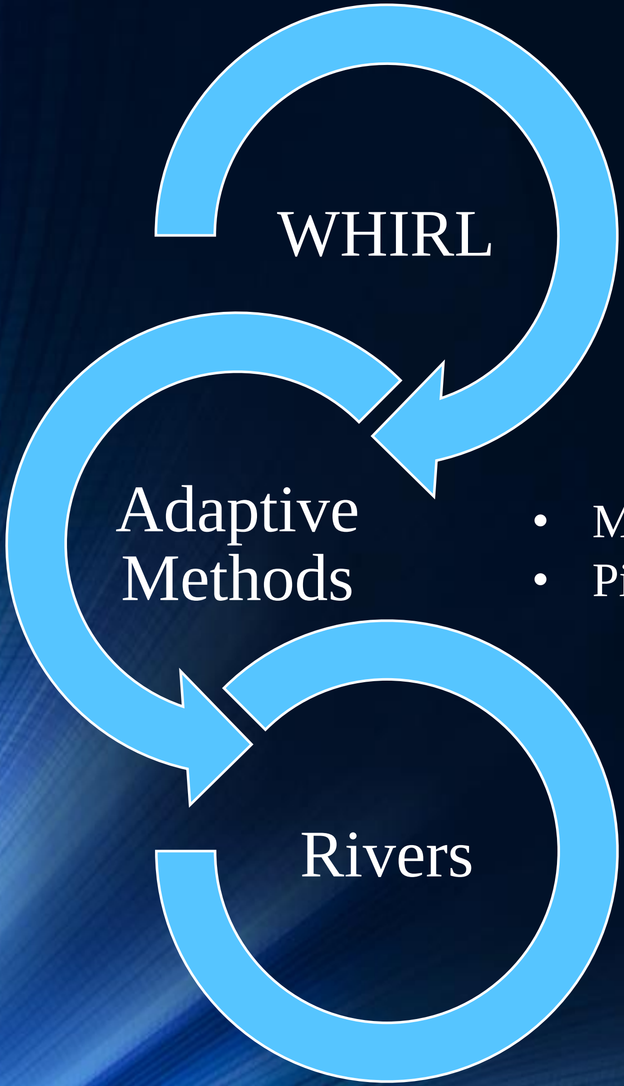
- First Prototype with minimal notion of adaptivity
- Uses *batch adaptivity* method

- More adaptive than system R
- Found weak with growing database size, in early 90’s

- Makes system R more adaptive
- Introduced *Adaptive Selectivity Estimation* scheme

- Inter-Operator Re-optimization
- Query Scrambling especially for WAN

Background(Contd.)



- Follows Intra-Operator adaptivity similar to INGRES
- Designed at AT&T Labs

- Memory adaptive sorting and hashing
- Pipelining and Ripple join algorithms

- Adaptive Partitioning
- Distributed *Queue mechanism* and *Graduated Declustering* scheme for data partitioning

Outline

- ✓Background
- Introduction
- Reorderability of plans
- Rivers and Eddies
- Routing Tuples in Eddies
- Conclusion and Future Work

Introduction

- Static Query Execution
  - Hardware and Workload Complexity
  - Data Complexity
  - User Interface Complexity
- Adaptive Query Processing
- A new system, “*Telegraph*”.

Introduction(Contd.)

- Run-Time Variations
  - Cost of operators
  - Selectivity of operators
  - Rates at which tuples arrive from input
- Architectural assumptions
- *Eddy*, the query processing operator for Telegraph.

Outline

- ✓ Background
- ✓ Introduction
- Reorderability of plans
- Rivers and Eddies
- Routing Tuples in Eddies
- Conclusion and Future Work

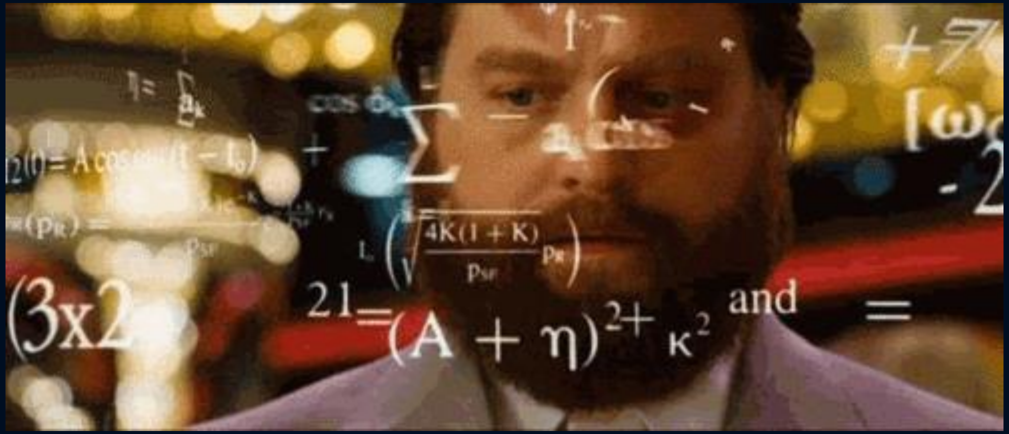
Reorderability of plans

- A challenge before we get to Eddy optimization.
- Changing operator order during run-time sounds cool, but is it easy?
- Adaptivity precedes best-case scenario in a variable environment.

Reorderability of plans(Contd.)

Synchronization Barrier:

- Where one operation hinders the speed of another operation.
- Constrains the order in which tuples are consumed.
- This is bad news for concurrency.



Reorderability of plans(Contd.)

<u>Extreme Example:</u>

- Merge join on two duplicate-free inputs. (slowlow and fasthi)
- At run-time the next tuple is always consumed from the relation that had the lowest values
- Slowlow is a slowly delivered external relation with many low columns in it’s bandwidth
- Fasthi is high bandwidth (i.e. delivers tuples fast) and has only high values in it’s column
- In this example fasthi is delayed when slowlow delivers tuples. This is the *synchronization barrier*.

Reorderability of plans(Contd.)

<u>Moments of Symmetry:</u>

- It is when the order of input can be changed without modifying the state in join.
- Merge join is a symmetric operator.
- How about nested loop joins?

Reorderability of plans(Contd.)

<u>Join Algorithms:</u>

- Index Join
- Nested Loop Join
- Merge Join
- Hybrid Hash Join
- Ripple Join(Ripples out from the corner of the join)

Reorderability of plans(Contd.)

<u>Join Algorithms:</u>

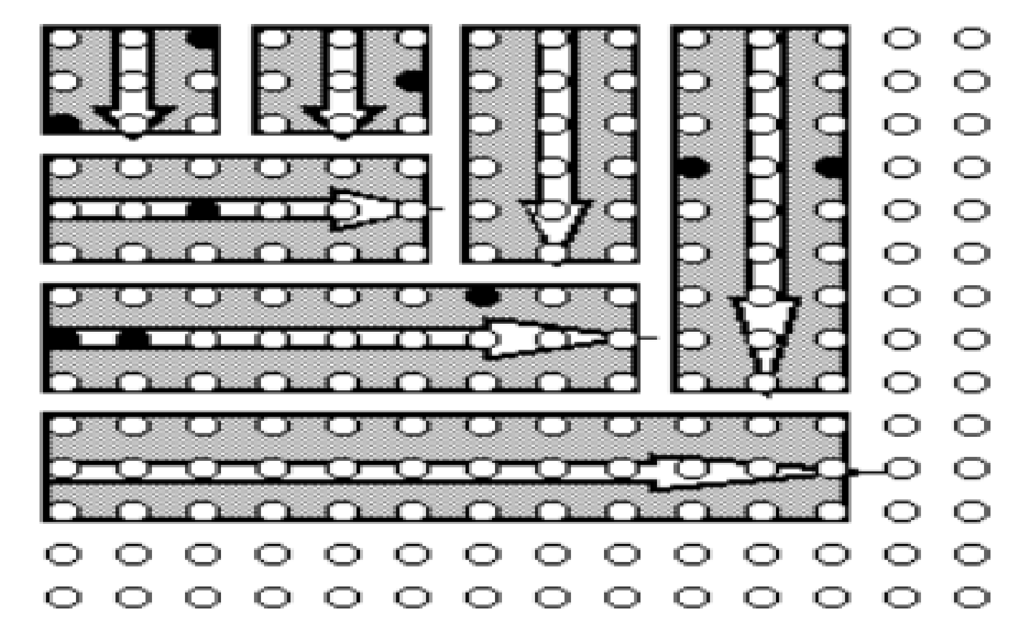
- Better adaptivity compared to other join algorithms.
- Adaptive/non-existent barriers.
- Frequent moments of symmetry.
- Minimal ordering constraints.
- So Who is the winner h

Ripple Join!!

Reorderability of plans(Contd.)

Variations of ripple join:

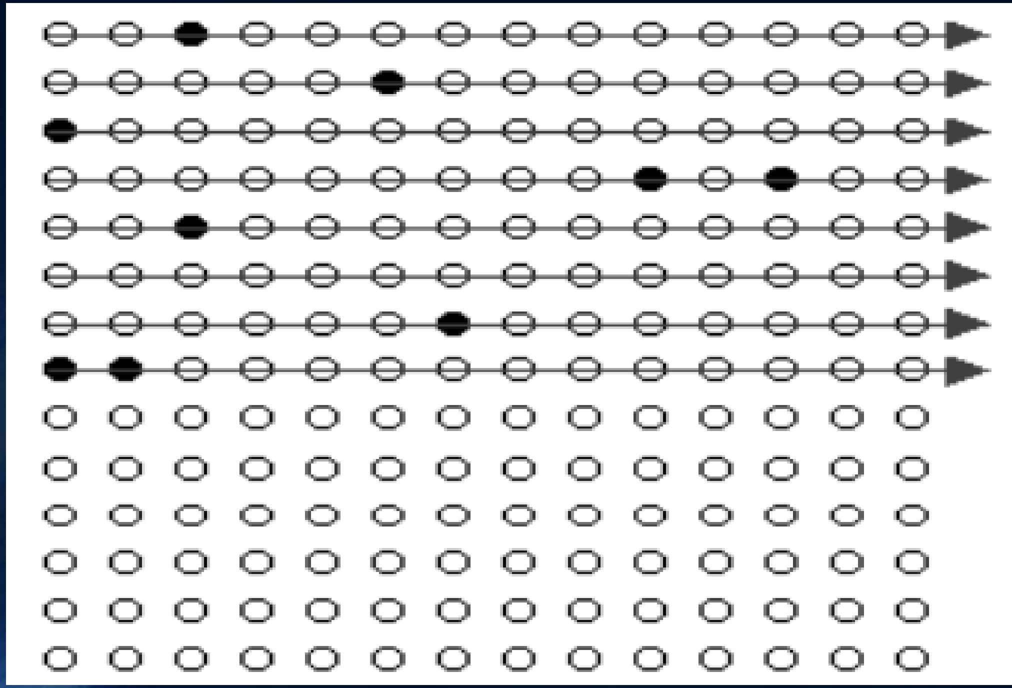
- **Block-** Obtain data *b* tuples at a time. For classic ripple join, *b*=1



Reorderability of plans(Contd.)

Variations of ripple join:

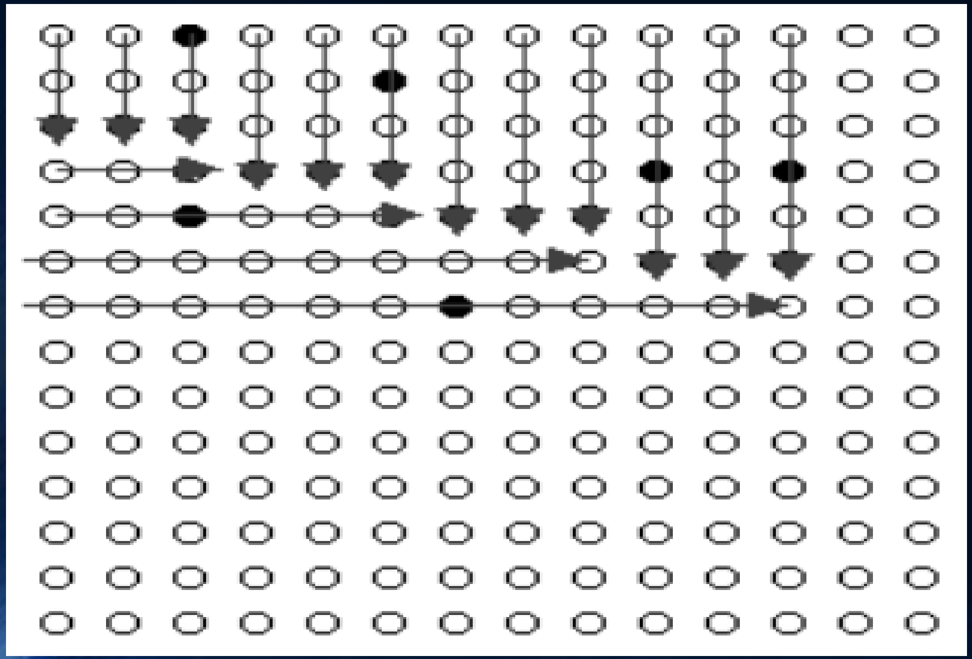
- **Index-** Identical to indexed-nested loop join.



Reorderability of plans(Contd.)

<u>Variations of ripple join:</u>

- **Hash-** Maintains hash tables of samples in memory. Used only for Equijoin queries.



Outline

- ✓Background
- ✓Introduction
- ✓Reorderability of plans
- Rivers and Eddies
- Routing Tuples in Eddies
- Conclusion and Future Work

Rivers & Eddies

River:

- *River* is an adaptive parallel dataflow infrastructure.
- *River* is multi-threaded, with efficient I/O mechanisms.

Pre-Optimization:

- Performed by Spanning tree of a query graph.
- Specific join algorithms chosen based on the edge being equijoin or non-equijoin.

Rivers & Eddies(Contd.)

An Eddy in the River:

- Implemented via a module in the *river*.
- Merges multiple unary and binary into single *n*-ary operator.
- Tuple entering eddy is identified with *Ready* and *Done* bits.
- Eddies are flexible in the shapes of the trees they can generate.
- They are flexible when operators are to be logically reordered.
- Eddies are so named because of this circular data flow within a river.

Outline

- ✓ Background
- ✓ Introduction
- ✓ Reorderability of plans
- ✓ Rivers and Eddies
- Routing Tuples in Eddies
- Related Work
- Conclusion and Future Work

Routing Tuples in Eddies

- Eddy module directs the flow of tuples from inputs to output through various operators.
- A priority queue is implemented to avoid clogging.
- The routing policy determines the efficiency of the system.

Routing Tuples in Eddies(Contd.)

Experimental Setup:

- Single-processor Sun Ultra-I workstation
- Solaris 2.6 Operating System
- 160MB RAM
- Hash Ripple Join
- Index Join with random I/Os within a file.

Routing Tuples in Eddies(Contd.)

So, How to route tuples to different Eddy operators?

- Naïve Eddy
- Fast Eddy

Routing Tuples in Eddies(Contd.)

Naïve Eddy:

- Best suited for handling operators with different costs but equal selectivity.
- The query operator with low cost processes its input tuple faster.
- Simple fluid dynamics makes naïve eddy effective.
- Most tuples are routed to the low cost operator first.

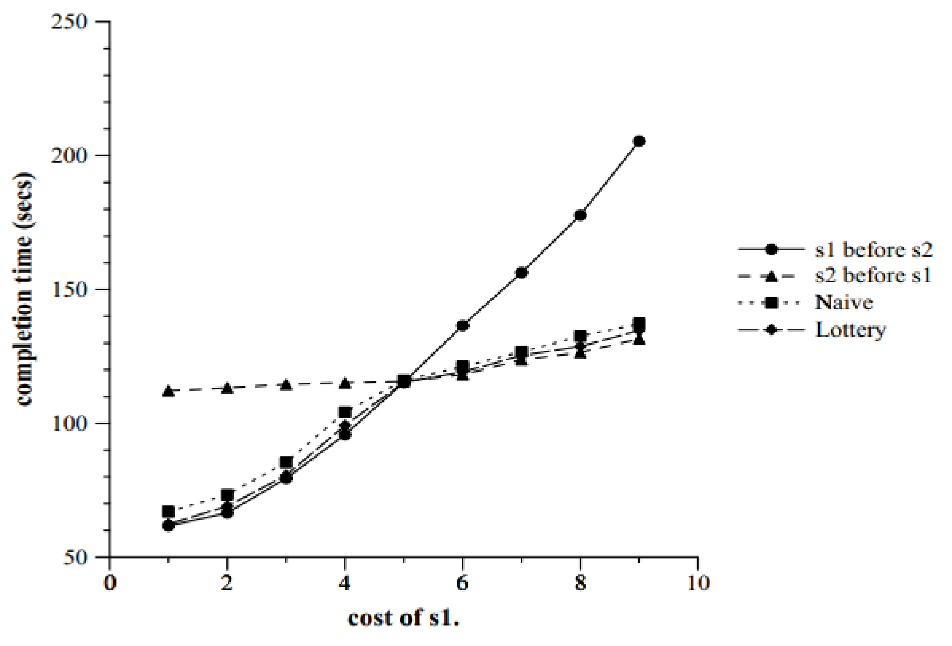
Routing Tuples in Eddies(Contd.)

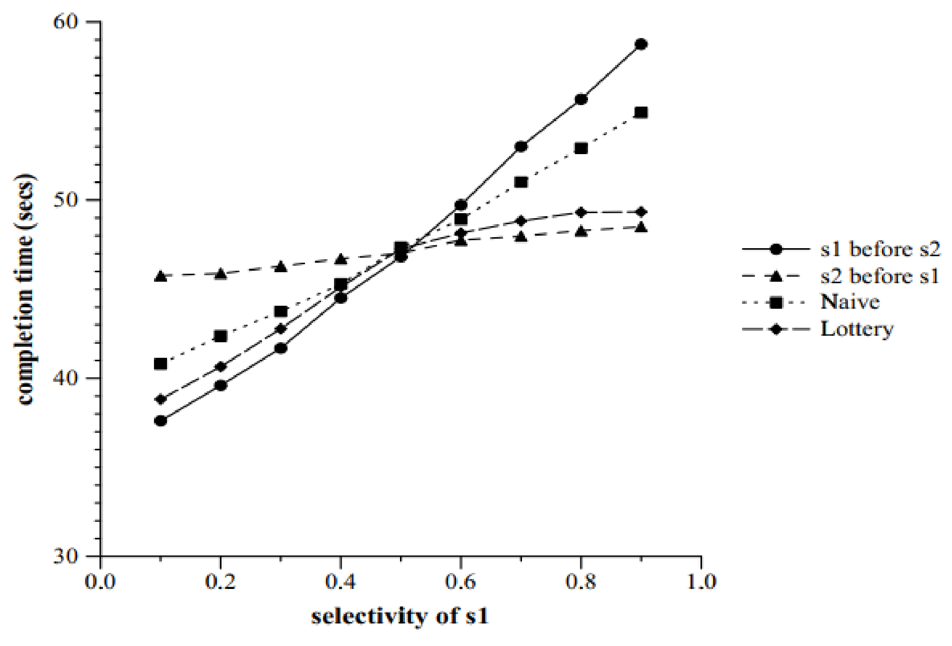
Fast Eddy:

- Naïve eddy was best suited for scenarios with operators of different cost but equal selectivity.
- *Lottery scheduling* is introduced to track both consumption and production over time, by maintaining *tickets*.

Routing Tuples in Eddies(Contd.)

Experiment results:

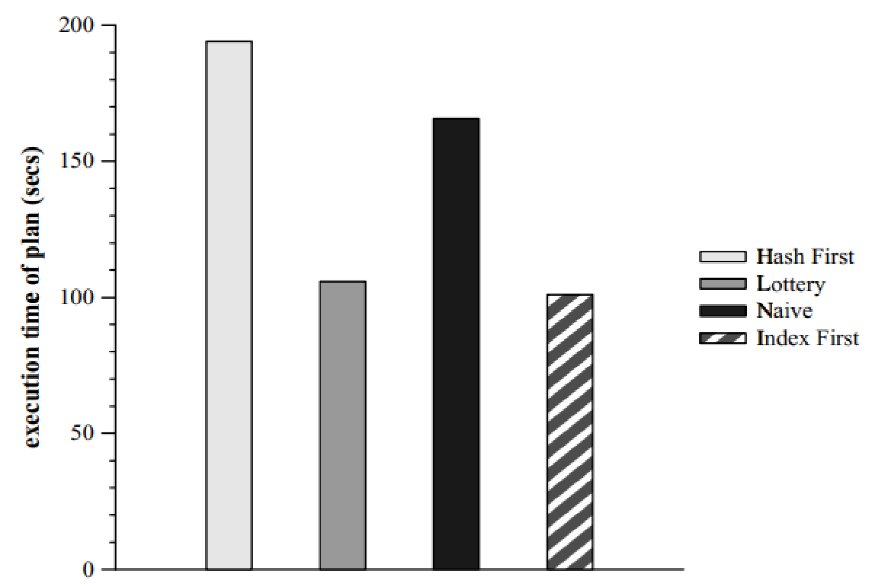




Routing Tuples in Eddies(Contd.)

<u>Performance of Joins:</u>

- Hash ripple join between R and S and index join between S and T.
- Eddy does well even in static scenarios.
- Lottery based eddy performs nearly optimal.



Routing Tuples in Eddies(Contd.)

Responding to Dynamic Fluctuations:

- Flaw in the lottery scheme.
- How to overcome this problem?
- *Window* Scheme comes to the rescue!!
- Time is partitioned into windows in this scheme.

Routing Tuples in Eddies(Contd.)

Responding to Dynamic Fluctuations: (Contd.)

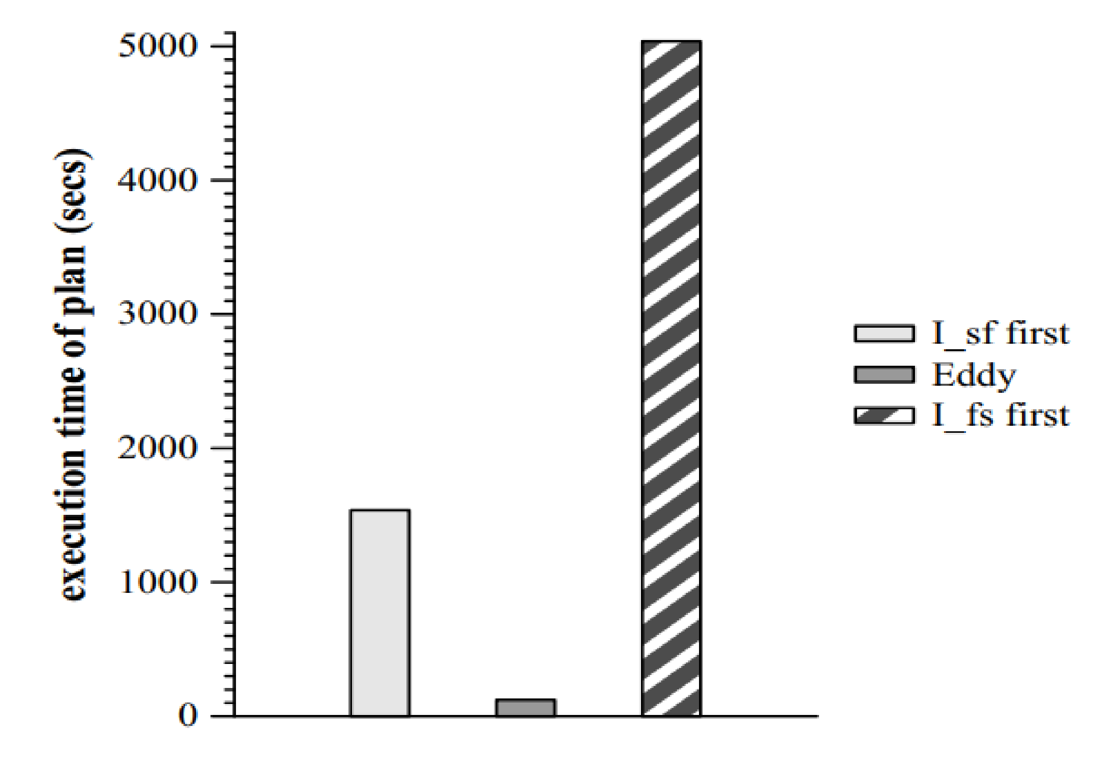
- Each operator holds *banked* tickets and *escrow* tickets, tracked by Eddy.
- Lets Experiment to check this theory.
- At the beginning, banked = escrow && escrow=0
- 3 table equijoin query, with 2 indexes, I_{fs} and I_{sf} experimented with 30,000 tuples.
- Indexes swapped after 30 seconds.

Routing Tuples in Eddies(Contd.)

Responding to Dynamic Fluctuations: (Contd.)

Experiment-1 Results:

- Eddy is way faster than both static plans I_{fs} and I_{sf}.
- Both indexes return a single matching tuple 1% of the time.



Routing Tuples in Eddies(Contd.)

Responding to Dynamic Fluctuations: (Contd.)

Experiment-2 & Results:

- Similar to first experiment, but more controlled.
- The two indexes return match 10% of the time.
- Eddy wins again!!

Routing Tuples in Eddies(Contd.)

Responding to Dynamic Fluctuations: (Contd.)

3rd experiment:

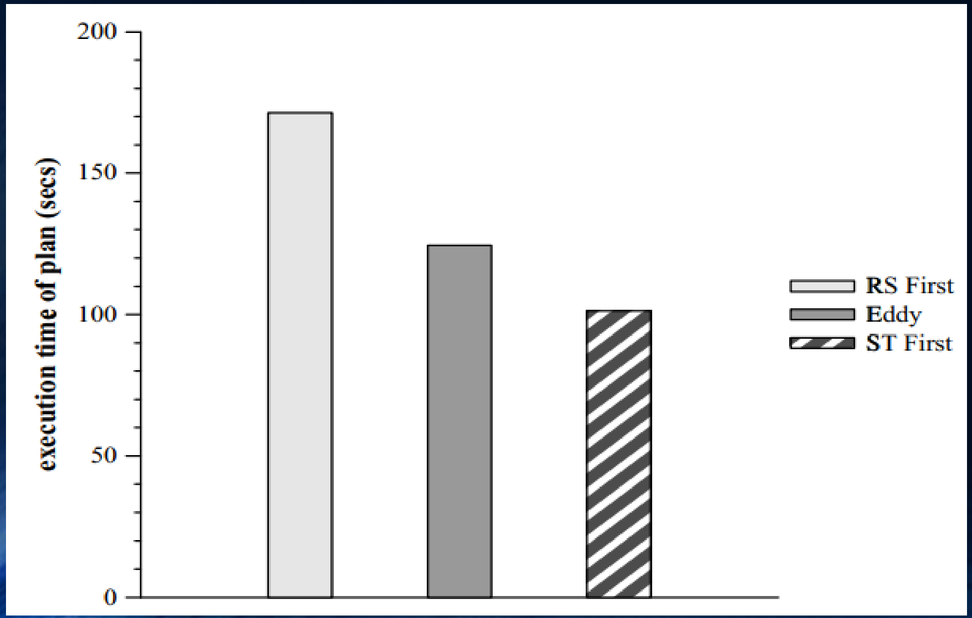
- Similar to Exp2 with fixed costs and modified selectivity with time.
- Similar results. But changing only the selectivity of the two operators result in less benefits for an adaptive scheme.

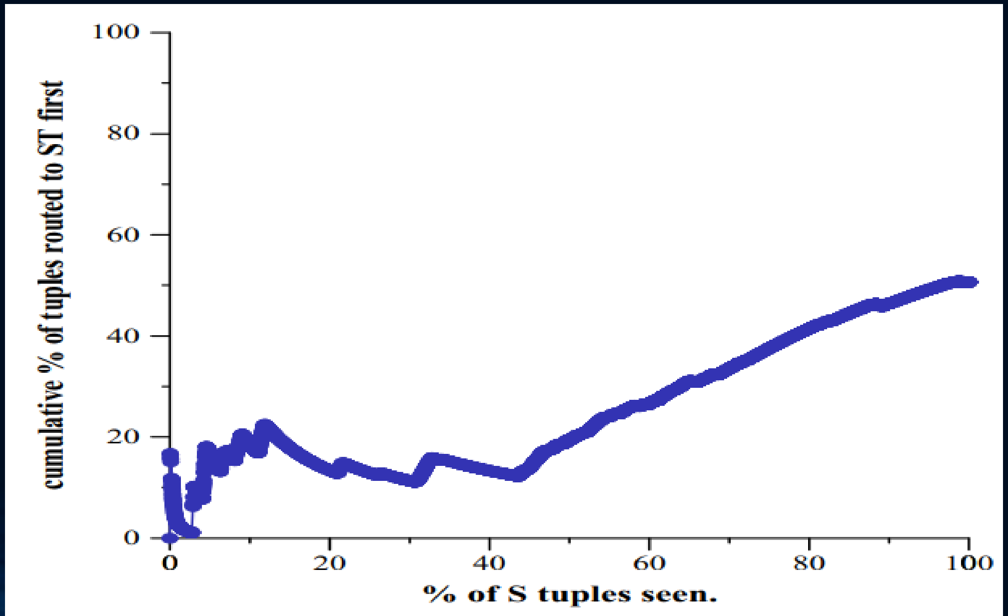
Routing Tuples in Eddies(Contd.)

<u>Delayed Delivery:</u>

Another experiment, to study the effect of initial delay on input relation.

Performance of eddy is not as expected.





Outline

- ✓Background
- ✓Introduction
- ✓Reorderability of plans
- ✓Rivers and Eddies
- ✓Routing Tuples in Eddies
- Conclusion and Future Work

Conclusion and Future work

- Eddies are beneficial in unpredictable environments.
- It can be used as a sole optimization mechanism.

But Challenges…

1. Develop eddy “ticket” policy.
2. Attack remaining static aspects of the schemes.
3. Harness parallelism and adaptivity available in rivers.

- Final goal is to make rivers, a generic parallel dataflow engine, and eddies the main scheduling mechanism.

Thank You!!

Questions???