



Network Query Engines

Craig Knoblock

USC Information Sciences Institute



Overview

- Network Query Engines
 - Tukwila, Telegraph, Niagara
 - Dataflow & pipelining similar to Theseus
 - Execution system with support for efficient query execution from remote data sources
 - Automatically generate query plans from XML queries
 - No support for loops, conditionals, or external interactions
 - Designed for querying only, not monitoring (except for NiagaraCQ)



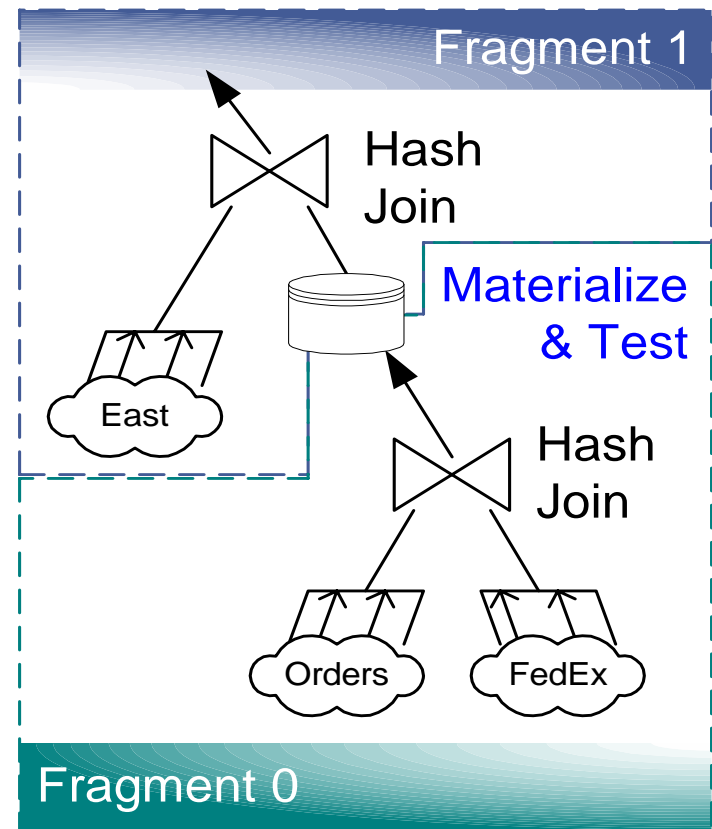
Tukwila (Ives et al. 1999)

- Adaptive network query processing for XML data
 - Interleaved execution and optimization
 - Inter-operator adaptivity
 - Dynamic operator re-ordering based on events
 - Memory overflow, wrapper timeout
- Notable new operators
 - X-SCAN: Efficient querying of streaming XML docs
 - JOIN: Double pipelined hash (probe is LHS or RHS)
 - DYNAMIC COLLECTOR: Efficient unioning of sources

Tukwila – Interleaved Planning and Execution

- Generates initial plan
- Can generate partial plans and expand them later
- Uses rules to decide when to reoptimize

From Ives et al., SIGMOD'99

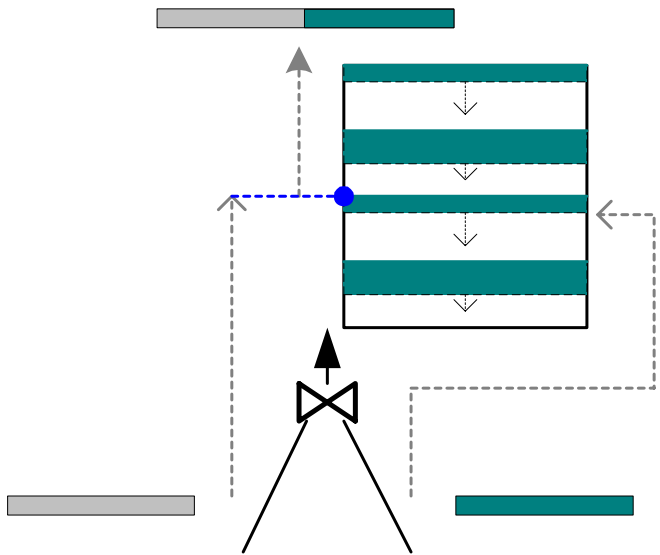


Fragment 0

```
WHEN end_of_fragment(0)
  IF card(result) > 100,000
  THEN re-optimize
```

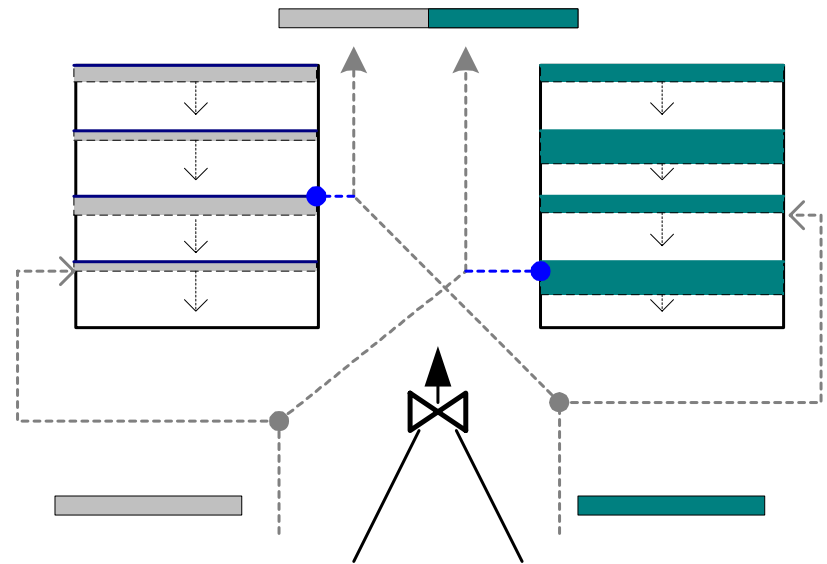
Tukwila – Adaptive Double Pipelined Hash Join

From Ives et al., SIGMOD'99



Hybrid Hash Join

- No output until inner read
- Asymmetric (inner vs. outer)



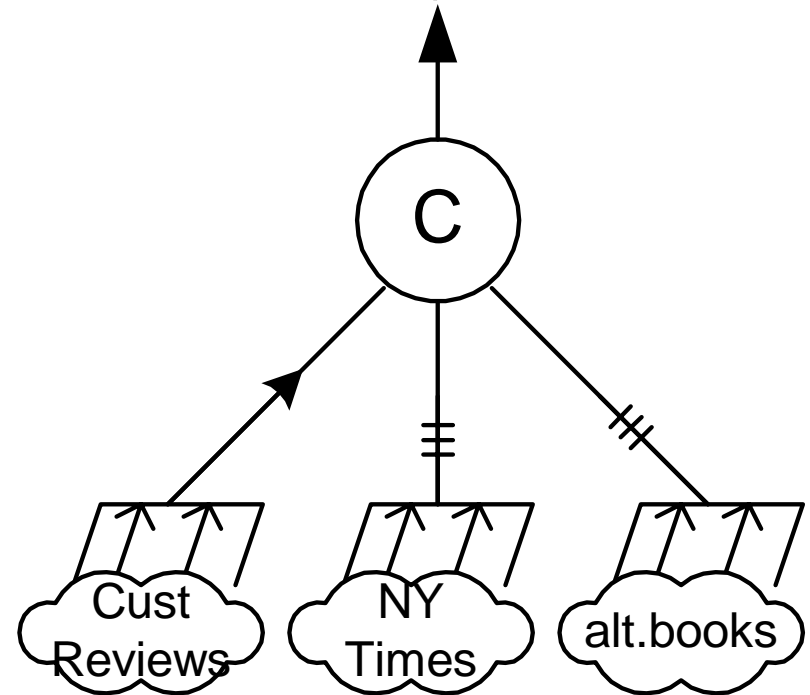
Double Pipelined Hash Join

- Outputs data immediately
- Symmetric
- More memory

Tukwila – Dynamic Collector Op

From Ives et al., SIGMOD'99

- Smart union operator
- Supports
 - Timeouts
 - slow sources
 - overlapping sources



```
WHEN timeout(CustReviews)
DO activate(NYTimes),
   activate(alt.books)
```



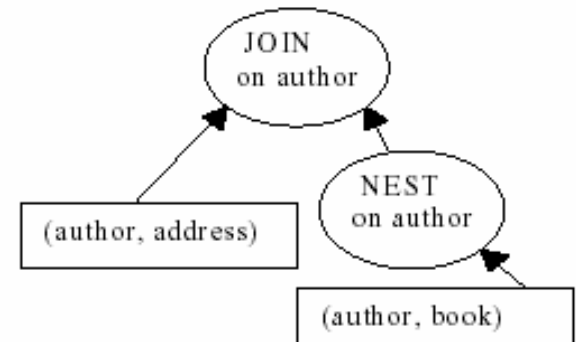
Niagara (Naughton, DeWitt, et al. 2000)

- Adaptive network query processing for XML data
 - Interleaved execution + document search
 - Supports streaming over blocking operators
 - Synchronization by re-evaluating operators or by propagating the differential result

Execution with partial results

[Shanmugasundaram et al. 2000]

- Niagara uses partial results to reduce the effects of blocking operators
 - Reduces blocking nature of aggregation or joins
- Basic idea
 - Execute future operators as data streams in, refine as slow operators catch up
 - Execution is driven by the **availability of real data**
 - Results are refined as additional data are processed





Approaches to Refining Results

- Re-evaluation
 - As new data becomes available, the operators re-output the results and the downstream operators are re-executed
 - Can be costly, but simple to implement
- Differential Algorithm
 - Each operator must support additions, deletes, and updates
 - Changed results must then be propagated to downstream operators

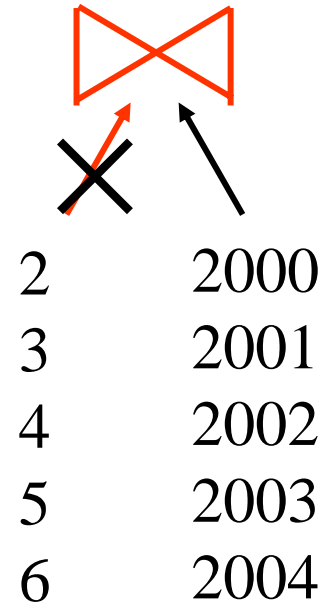


Telegraph (Hellerstein et al. 2000)

- Tuple-level adaptivity
- **Rivers** (optimize horizontal parallelism)
 - Adaptive dataflow on clusters (ie, data partitioning)
- **Eddies** (optimize vertical parallelism)
 - Leverage commutative property of query operators to dynamically route tuples for processing

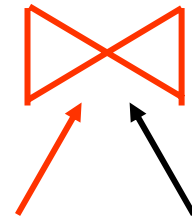
Adaptable Joins, Issue 1

- Synchronization Barriers
 - One input frozen, waiting for the other
 - Can't adapt while waiting for barrier!
 - So, favor joins that have:
 - no barriers
 - at worst, adaptable barriers



Adaptable Joins, Issue 2

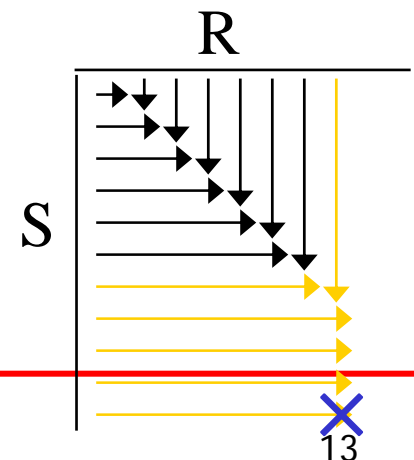
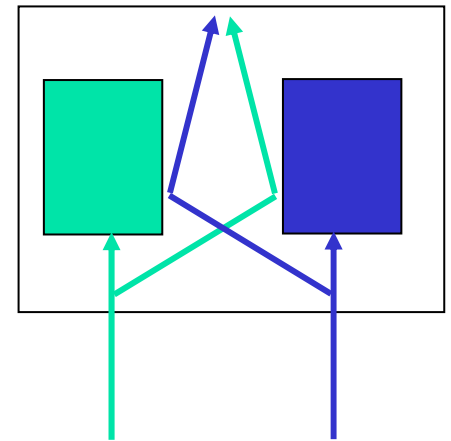
- Would like to reorder *in-flight* (pipelined) joins
- Base case: swap inputs to a join
 - What about per-input state?
- Moment of symmetry:
 - inputs can be swapped w/o state management
- E.g.
 - Nested Loops: at the end of each inner loop
 - Merge Join: any time*
 - Hybrid or Grace Hash: never!
- More frequent moments of symmetry
→ more frequent adaptivity



Ripple Joins: Prime for Adaptivity

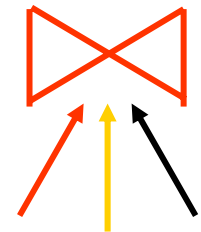
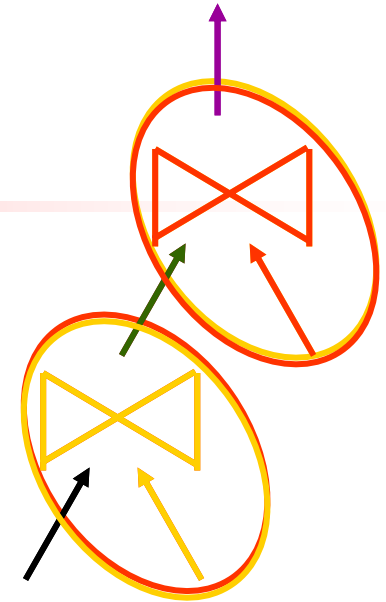
- Ripple Joins

- Pipelined hash join (a.k.a. hash ripple, Xjoin)
 - No synchronization barriers
 - Continuous symmetry
 - Good for equi-join
- Simple (or block) ripple join
 - Synchronization barriers at “corners”
 - Moments of symmetry at “corners”
 - Good for non-equi-join
- Index nested loops
 - Short barriers
 - No symmetry

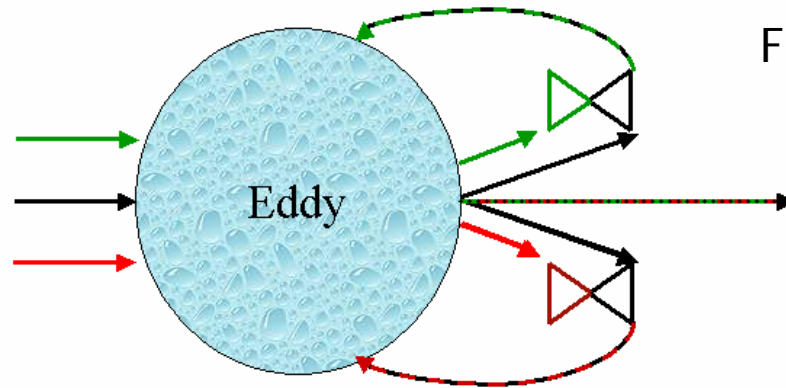


Beyond Binary Joins

- Think of swapping “inners”
 - Can be done at a global moment of symmetry
- Intuition: like an n-ary join
 - Except that each pair can be joined by a different algorithm!
- So...
 - Need to introduce n-ary joins to a traditional query engine



Telegraph – Beyond Reordering Joins



From Avnur & Hellerstein,
SIGMOD 2000

Eddy

- A pipelining tuple-routing iterator (just like join or sort)
- Adjusts flow adaptively
 - Tuples flow in different orders
 - Visit each op once before output
- Naïve routing policy:
 - All ops fetch from eddy as fast as possible
 - Previously-seen tuples precede new tuples



Discussion

- Theseus, Tukwila, Telegraph, Niagara are all:
 - Streaming dataflow systems
 - Targeting network-based query processing
 - Large source latencies
 - Unknown characteristics of sources
 - Proposed various techniques for improving the efficiency of processing data
 - More efficient operators (e.g., double-pipelined join)
 - Tuple-level adaptivity
 - Partial results for blocking operators
 - Speculative execution