What’s the Problem?
- “optimize-then-execute” does not work well for data integration applications
- Why?

Discussion 1
- Their approach is to "favor adaptability over best-case performance."
- Is this a good approach? Did they succeed?
- How does this compare with other approaches we've seen?
Synchronization Barriers

- What are synchronization barriers?
  - When table-scan A has to wait for table-scan B to finish or reach something before A can continue
- Synchronization barriers limit concurrency

Example

- Merge join - extreme case
  - Recall: merge join always takes next tuple from relation which had lowest value most recently

Moments of Symmetry

- What are moments of symmetry?
  - Moment of symmetry = synchronization barrier where the join algorithm can swap inputs without modifying any state in the join

Example

- Nested-loops join

Moments of Symmetry

- We switched roles of R and S two times during the join!
  - But we still end up “matching up” all the tuples (just in a different order)
  - And we didn’t have to change any internal state (except for remembering \( C_R \) and \( C_S \))
Outline
- What’s the Problem?
- Synchronization Barriers
- Moments of Symmetry
  - Choosing a Join Algorithm
    - Eddies
    - Lottery Scheduling
    - Results

Choosing a Join Algorithm
- What do we want in a join algorithm?
  - Frequent moments of symmetry
  - Adaptive/Non-existent synchronization barriers
  - Minimal ordering constraints
  - So we can’t use hybrid hash join, merge join, nested loop join...

Choosing a Join Algorithm
- Instead, we choose the Ripple Join family:
  - Block Ripple Join
  - Index Ripple Join
  - Hash Ripple Join
  - ...and relatives

Choosing a Join Algorithm
- Example
  1) Get tuple \( r \) from \( R \), or tuple \( s \) from \( S \)
  2) Compare \( r \) and \( s \) with every tuple in \( r \times s \) rectangle

Choosing a Join Algorithm
- Ripple Join is basically like nested loops join except that moments of symmetry occur much more often
  - In fact, at every corner of a rectangle there is a moment of symmetry (i.e. between every tuple for hash and index ripple joins)
  - Offers adaptivity at modest overhead in performance and memory
Outline

- What’s the Problem?
- Synchronization Barriers
- Moments of Symmetry
- Choosing a Join Algorithm
- Eddies
- Lottery Scheduling
- Results

Eddies

- What are eddies?
  - Eddy = n-ary tuple router interposed between n data sources and a set of query processing operators
  - Basically, eddy
    1) takes n tuples as input
    2) feeds tuples to operators
    3) operators feed result back to eddy
    4) eddy sends result
       - to output; or
       - to other operators

Eddies

- Each tuple entering eddy has a tuple descriptor
  - Eddy only sends tuple to operators with corresponding Ready=1
  - Eddy sets corresponding Done=1 when operator sends tuple back, and updates Ready bits
  - When all Done=1, eddy sends tuple to output

<table>
<thead>
<tr>
<th></th>
<th>Join_1</th>
<th>Select_1</th>
<th>Select_2</th>
<th>Join_2</th>
<th>Select_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Done</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Eddies

- Naïve Eddy
  - Operator s1 lower-cost than s2
  - So s1 consumes input faster than s2
  - Equal selectivity
    - higher selectivity --> more likely to return tuples to eddy
  - Back-pressure effect
    - s1 consumes fast, s2 produces slow, so most tuples end up being routed to s1 first
  - Desired effect, even though costs were not explicitly exposed or tracked!

Outline

- What’s the Problem?
- Synchronization Barriers
- Moments of Symmetry
- Choosing a Join Algorithm
- Eddies
- Lottery Scheduling
- Results
Lottery Scheduling
- Need a learning algorithm to track both consumption and production over time
  - i.e. take into account selectivity (as well as cost)
- Lottery Scheduling
  - Each time eddy gives tuple to operator, operator gets 1 ticket
  - Each time operator returns tuple to eddy, operator loses 1 ticket
  - Operator must use possessed tickets to win "lottery" to get new tuples
  - Therefore more "efficient" operator -> more tickets
  -> more likely to win lottery -> more likely to get tuples

Outline
- What's the Problem?
- Synchronization Barriers
- Moments of Symmetry
- Choosing a Join Algorithm
- Eddies
- Lottery Scheduling
- Results

Results

Figure 4: Performance of two 50% selections, x2 has cost 5, x3 versus selectivity

Figure 5: Performance of two selections of cost 5, x2 has 50% selectivity, x3 versus across range

Figure 6: Tuple flow with lottery scheme for the variable-selectivity experiments (x3 x3)
Discussion 2

- Compare and contrast this system to Mariposa's bidding approach.
  - Is one better than the other?
  - Could you combine the two?
- Compare and contrast Eddies and Tukwila.
  - Which is better?