Dynamic Scheduling of Complex Distributed Queries

Luc Bouganim - Françoise Fabret
C. Mohan - Patrick Valduriez

Outline

• Architecture
• Classical Query Execution
• Problem
• Dynamic Query Execution
• Performance results
• Conclusion

System Architecture

Query Engine Architecture
Query Execution Plan (QEP)

(a) Query

(b) Logical execution plan

(c) Physical execution plan

- Dependency constraints
- Memory constraints

(b) Logical execution plan

(c) Physical execution plan

Query Plan Scheduling

- Execute A and C.
- When A finishes, execute B.
- When B and C finish, execute D

Dynamic Query Engine Architecture

Dynamic QEP Optimizer (DQO)

Dynamic Query Scheduler (DQS)

Dyn. Query Processor (DQP)

Communication Manager (CM)

Dynamic Query Scheduling

• Execute A and C.
• When A finishes, execute B.
• When B and C finish, execute D

Problems

• **Accuracy of estimates**: Estimates can be inaccurate
  ⇒ sub-optimal execution plan [KD98], [IFF+99]

• **Query engine stalling**: data is temporarily inaccessible
  (i) the remote data is the result of a complex sub-query;
  (ii) the remote site is overloaded and/or is not powerful enough;
  (iii) the network is slow or not reliable.

• **Memory limitation**: Less memory than needed
  ⇒ Thrashing of the system because of paging [BKV98], [ND98].
Dynamic Query Execution Process

**Scheduling Plan** SP = [ABC]

- **Dynamic Query Processor (DQP)**
  - Interleave the execution of the QF's according to SP

- **Planning**
  - DQO
  - DQS
  - DQP

- **Executing**
  - QEP_1
  - QEP_2
  - SP_1
  - SP_2
  - SP_3
  - SP_4

- New stats

**Dynamic Query Scheduler (1)**

- Use **Benefit Materialization Indicator** for materialization

  - **BMI (q): Profitability criterion for materialization**
  
    \[ \text{Low BMI} \quad \text{BMT} \quad \text{High BMI} \]

    \[ \Rightarrow \text{Apply Materialization for a PC } p \text{ if } BMI(p) > BMT \]

- Use **Critical Degree** to assign priorities

  - **Critical Degree (p): Total idle time when p is computed alone.**
  
    \[ \Rightarrow \text{Schedule QF's with high critical degree as soon as possible} \]

**Dynamic Query Scheduler (2)**

- **C-schedulable**
  - p is C-schedulable when it has no more dependency constraint

- Insert a **materialization** in p to obtain a C-schedulable fragment

- Materialization fragment MF(p)
Dynamic Query Optimizer (DQO)

- Mem(p) = X + Y
- p is M-schedulable if Mem(p) < available memory
- apply M-fragmentation to obtain M-schedulable fragments [BKV98]

Scheduling Plan Example

- Critical degree order: B, E, D, F, C
- bmi(B) < bmt ⇒ Do not materialize
- bmi(E) > bmt ⇒ Materialize
- bmi(F) < bmt ⇒ Do not materialize

Scheduling Plan Example

- Critical degree order: B, E, D, F, C
- bmi(B) < bmt ⇒ Do not materialize
- bmi(E) > bmt ⇒ Materialize
- bmi(F) < bmt ⇒ Do not materialize

Experiments ….

- One query
- Experiments
  - One slowed-down relation
  - Vary the average delivery rate
  - Several slowed-down relations

- Strategies
  - Classical iterator model (SEQ)
  - Materialize All (MA)
  - Dynamic Scheduling Execution (DSE)
  - Computed Lower Bound (LWB)
Performance Results

One slowed-down relation: F

<table>
<thead>
<tr>
<th>Total retrieval time of F (s)</th>
<th>0.50</th>
<th>1.00</th>
<th>1.50</th>
<th>2.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSE</td>
<td>2.00</td>
<td>2.50</td>
<td>3.00</td>
<td>3.50</td>
</tr>
<tr>
<td>SEQ</td>
<td>1.50</td>
<td>2.00</td>
<td>2.50</td>
<td>3.00</td>
</tr>
<tr>
<td>MA</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
<td>2.50</td>
</tr>
<tr>
<td>LW B</td>
<td>0.50</td>
<td>1.00</td>
<td>1.50</td>
<td>2.00</td>
</tr>
</tbody>
</table>

Varying the basic_w_avg

<table>
<thead>
<tr>
<th>Performance gain of DSE/SEQ</th>
<th>0.00</th>
<th>0.50</th>
<th>1.00</th>
<th>1.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic_w_avg (µs/tuple)</td>
<td>0.00</td>
<td>50</td>
<td>100</td>
<td>150</td>
</tr>
</tbody>
</table>

Conclusion

• A solution to Data Availability and Memory Overflow problems
  – Follows the optimizer decision
  – Allows parallel execution of the wrappers
  – Monitors the data arrival rates and adapt the execution strategy
  – Handles delays, bursty and slow delivery in the same way
  – Is based on simple cost formulas
  – Do not induce high overheads if no problem occurs

• Extensions
  – Multi-query scheduling

Related works

• Scrambling [PDIS96, DAPD98, SIGMOD98] from Urban, Amsaleg, Franklin, Tomasic
  – Uses the iterator model and relies on time-out
  – Solve Initial Delays [PDIS, SIGMOD] or Bursty [DAPD]
  – Use cost based formulas and re-optimization [SIGMOD]

• Tukwila [SIGMOD99] from Ives, Florescu, Friedman, Levy and Weld
  – Uses the iterator model and relies on time-out
  – Relies on Double pipelined hash join

• Our previous work [CIKM98] from Bouganim, Kapitskaia, Valduriez
  – Centralized context
  – Solve memory overflow only