Flow Algorithms for Parallel Query Optimization

Amol Deshpande, University of Maryland
Lisa Hellerstein, Polytechnic University, Brooklyn
Introduction and Motivation

• Motivation: Parallel Query Processing
  – Increasing parallelism in computing
    • Shared-nothing clusters, multi-core technology, Grid, P2P...
  – Two ways to exploit parallelism:
    • Partitioned parallelism
      – Operator copies run in parallel on partitions of data
      – Harder to set up, more communication overheads
    • Pipelined Parallelism
      – Each operator run on a different processor
      – Better cache locality, easier to reason about
      – May be the only option in some scenarios
      – Cannot exploit the parallelism fully
Motivation: Parallel Query Processing

Example query
select *
from R1, R2, R3, R4, R5
where R1.a = R2.a and
  R1.b = R3.b and
  R1.c = R4.c and
  R4.d = R5.d

A pipelined query plan

Tuple Throughput = 1000 tuples/sec

Driver relation

Processor 1
R1 → R2 → R2
1000/sec

Processor 2
R1 → R3 → R3
100/sec

Processor 3
R1 → R4 → R4
10/sec

Processor 4
R4 → R5 → R5
1/sec

sel = 0.1
Motivation: Parallel Query Processing

Example query
select *
from R1, R2, R3, R4, R5
where R1.a = R2.a and
  R1.b = R3.b and
  R1.c = R4.c and
  R4.d = R5.d

A pipelined query plan

Tuple Throughput = 1000 tuples/sec

Processor 1
R1 × R2

Processor 2
R1 × R3

Processor 3
R1 × R4

Processor 4
R4 × R5
Proposed Solution: Interleaving Plans

Example query
select *
from R1, R2, R3, R4, R5
where R1.a = R2.a and
    R1.b = R3.b and
    R1.c = R4.c and
    R4.d = R5.d

Plan 1 (50%)

Plan 2 (50%)

Tuple Throughput ≈ 1998 tuples/sec (max = 2790 tuples/sec)
Motivation: Selection ordering with precedence constraints

Given a driver relation, choosing a left-deep pipelined plan for a multi-way join query is equivalent to precedence-constrained selection ordering.

Example query

```sql
select *
from R1, R2, R3, R4, R5
where R1.a = R2.a and
    R1.b = R3.b and
    R1.c = R4.c and
    R4.d = R5.d
```

Cost of $O_4$: $c_4 = \text{average per-tuple cost of the join } R_1 \bowtie R_4$

Selectivity of $O_4$: $p_4 = \text{fanout of } R_1 \bowtie R_4$

= average number of $R_4$ matches for an $R_1$ tuple (may be > 1)
Introduction and Motivation

• Motivation: Selection ordering with precedence constraints

• Motivation: Query Processing over Web Services
  – Increasing abundance of web services and standardized APIs for querying them
    • Shopping, Web Search, Housing etc. ...
  – Similar issues as pipelined query processing
    • Each web service == a processor
    • Typically limited number of requests allowed per minute

• Motivation: Similar to many problems in other domains
  – Sequential testing (e.g. for fault detection) [SF’01, K’01]
  – Learning with attribute costs [KKM’05]
Prior Related Work

- Rich literature on parallel and distributed query processing
  - Didn’t consider interleaving plans
- Interleaving Plans for Selection Ordering [Condon et al., 2006]
  - Simpler types of queries
  - $O(n^2)$ algorithm for computing the optimal plan
- Query Optimization over Web Services [Srivastava et al., 2006]
  - Algorithm for choosing an optimal serial (single) plan
  - Considered cyclic queries and a larger plan space
- Eddies [Avnur and Hellerstein, 2000]? 
  - Interleaving plans are not adaptive
  - Distributed eddies [Tian and DeWitt, 2006]: Similar metrics, but they focus on adaptivity
Outline

• Introduction and Motivation

• Problem Definition

• Algorithms for finding Optimal Interleaving Plans
  – Selective operators
  – Non-selective operators

• Experimental Results
• Each operator runs on a different processor

Example query
select * from R1, R2, R3, R4, R5 where R1.a = R2.a and R1.b = R3.b and R1.c = R4.c and R4.d = R5.d

Processor
O2
r₂, p₂

Processor
O3
r₃, p₃

Processor
O4
r₄, p₄

Processor
O5
r₅, p₅

rᵢ = rate limit of operator Oᵢ
= Number of tuples it can process per unit time (also called capacity)

Can be computed using cᵢ
Interleaving Plans

• An interleaving plan defined by:
  – A set of permutations of the operators
  – A weight $w_i$ for each permutation ($\Sigma w_i = 1$)

Interleaving plan: $O_2 \rightarrow O_3 \rightarrow O_4 \rightarrow O_5, w = 0.5$

$O_4 \rightarrow O_5 \rightarrow O_3 \rightarrow O_2, w = 0.5$
Problem Definition

• Given:
  – $n$ selection operators $O_1, \ldots, O_n$
  – selectivity $p_i$ and rate $r_i$ for each operator $O_i$,
  – a precedence graph $G$ over the operators

Find the optimal interleaving plan that maximizes the tuple throughput \textit{(and hence total completion time)}

• Definition: $O_i$ is called selective if $p_i < 1$

• We assume tree-structured precedence constraints (correspond to queries with no cycles)
Outline

• Introduction and Motivation

• Problem Definition

• Algorithms for finding Optimal Interleaving Plans
  – Selective operators
  – Non-selective operators

• Experimental Results
Overview of Approach

• View an interleaving plan as a collection of tuple flows

• Definition: An operator is \textit{saturated} if it is processing at its rate limit

• Lemma: Saturation $\rightarrow$ Optimality
  – If all operators are saturated, we have an optimal solution

• Algorithm for when $G$ is a forest of chains

• Recursively reduce the general case to forests of chains

• Combine the solutions for sub-problems
Saturated Suffix Lemma [CDHW’06]

- Given an interleaving plan, IF:
  - A set of operators is saturated (processing at their rate limit), and
  - No flow from a saturated operator to an unsaturated operator
  THEN the plan is optimal.
- The actual permutations used irrelevant
- However not necessary when there are precedence constraints
Overview of Approach

• View an interleaving plan as a collection of tuple flows

• Definition: An operator is saturated if it is processing at its rate limit

• Lemma: Saturation $\Rightarrow$ Optimality
  – If all operators are saturated, we have an optimal solution

• Algorithm for when $G$ is a forest of chains

• Recursively reduce the general case to forests of chains

• Combine the solutions for sub-problems
**Algorithm: $G = \text{forest of chains}$**

**Example query**

```
select *
from R1, R2, R3, R4, R5
where R1.a = R2.a and
    R1.b = R3.b and
    R1.c = R4.c and
    R4.d = R5.d
```
Algorithm: $G = \textit{forest of chains}$

- Sort in non-increasing order by rate
- Start adding flow from left-to-right till:
  - Cond 1: A parent can exactly saturate a child (merge and recurse)
  - Cond 2: A node can exactly saturate its \textit{predecessor} (merge and recurse)
  - Cond 3: No more flow can be added ($\rightarrow$ saturation $\rightarrow$ optimality)

- Merging two operators:
  $O_2$ is merged into $O_1 \Rightarrow O_2$ is applied immediately after $O_1$

\[\begin{align*}
O_1 & \quad r_1 = 1000 \\
     & \quad p_1 = 0.5 \\
O_2 & \quad r_2 = 500 \\
     & \quad p_2 = 0.5
\end{align*}\]

$O_1$ exactly saturates $O_2$

\[\begin{align*}
O_{12} & \quad r_{12} = 1000 \\
       & \quad p_{12} = 0.25 \\
O_2 & \quad r_2 = 500 \\
     & \quad p_2 = 0.5
\end{align*}\]
Algorithm: $G = \textit{forest of chains}$

- Sort in non-increasing order by rate
- Start adding flow from left-to-right till:
  - Cond 1: A parent can exactly saturate a child (merge and recurse)
  - Cond 2: A node can exactly saturate its \textit{predecessor} (merge and recurse)
  - Cond 3: No more flow can be added (→ saturation → optimality)

\textbf{Step 1:}
Send 600 units along $O_4 \rightarrow O_2 \rightarrow O_3 \rightarrow O_5$

Cond 1 satisfied for $O_4$ and $O_5$
\textit{Merge $O_5$ into $O_4$}

Cond 2 satisfied for $O_2$ and $O_4$
\textit{Merge $O_4$ into $O_2$}
Algorithm: $G = \textit{forest of chains}$

- Sort in non-increasing order by rate
- Start adding flow from left-to-right till:
  - Cond 1: A parent can exactly saturate a child (merge and recurse)
  - Cond 2: A node can exactly saturate its predecessor (merge and recurse)
  - Cond 3: No more flow can be added ($\rightarrow$ saturation $\rightarrow$ optimality)

**Step 2:**
Send 240 units along $O_2 \rightarrow O_4 \rightarrow O_5 \rightarrow O_3$

Cond 2 satisfied for $O_3$ and $O_{245}$
$\text{Merge } O_{245} \text{ into } O_3$

<table>
<thead>
<tr>
<th>Node</th>
<th>Rate $r$</th>
<th>Probability $p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_4$</td>
<td>$r_4 = 900$</td>
<td>$p_4 = 0.5$</td>
</tr>
<tr>
<td>$O_{245}$</td>
<td>$r_{245} = 600$</td>
<td>$p_{245} = 0.125$</td>
</tr>
<tr>
<td>$O_3'$</td>
<td>$r_3' = 750$</td>
<td>$p_3 = 0.5$</td>
</tr>
<tr>
<td>$O_5$</td>
<td>$r_5 = 225$</td>
<td>$p_5 = 0.5$</td>
</tr>
</tbody>
</table>
Algorithm: $G = \textit{forest of chains}$

- Sort in non-increasing order by rate
- Start adding flow from left-to-right till:
  - Cond 1: A parent can exactly saturate a child (merge and recurse)
  - Cond 2: A node can exactly saturate its \textit{predecessor} (merge and recurse)
  - Cond 3: No more flow can be added (\(\rightarrow\) saturation \(\rightarrow\) optimality)

\textbf{Step 3:}
Send 720 units along
O3 $\rightarrow$ O2 $\rightarrow$ O4 $\rightarrow$ O5

Cond 3 satisfied for O3245

All operators are saturated
\(\rightarrow\) Optimality

\[r_4 = 900, \quad p_4 = 0.5\]

\[r_{245} = 600, \quad p_{245} = 0.125\]

\[r_{3245} = 720, \quad p_3 = 0.0625\]

\[r_5 = 225, \quad p_5 = 0.5\]
Algorithm: \( G = \text{forest of chains} \)

- Sort in non-increasing order by rate
- Start adding flow from left-to-right till:
  - Cond 1: A parent can exactly saturate a child (merge and recurse)
  - Cond 2: A node can exactly saturate its predecessor (merge and recurse)
  - Cond 3: No more flow can be added (\( \rightarrow \) saturation \( \rightarrow \) optimality)

**Final Interleaving Plan:**

\[ O_4 \rightarrow O_2 \rightarrow O_3 \rightarrow O_5, \quad w = \frac{600}{1560} \]

\[ O_2 \rightarrow O_4 \rightarrow O_5 \rightarrow O_3, \quad w = \frac{240}{1560} \]

\[ O_3 \rightarrow O_2 \rightarrow O_4 \rightarrow O_5, \quad w = \frac{720}{1560} \]
Algorithm: $G = \textit{forest of chains}$

- Sort in non-increasing order by rate
- Start adding flow from left-to-right till:
  - Cond 1: A parent can exactly saturate a child (merge and recurse)
  - Cond 2: A node can exactly saturate its \textit{predecessor} (merge and recurse)
  - Cond 3: No more flow can be added (→ saturation → optimality)

- Theorem: The algorithm runs in $O(n^2 \log n)$ time and finds an interleaving plan with at most $4n - 3$ distinct permutations.
Overview of Approach

- View an interleaving plan as a collection of tuple flows

- Definition: An operator is saturated if it is processing at its rate limit

- Lemma: Saturation $\rightarrow$ Optimality
  - If all operators are saturated, we have an optimal solution

- Algorithm for when $G$ is a forest of chains

- Recursively reduce the general case to forests of chains

- Combine the solutions for sub-problems
General Case

Eliminate one fork at a time using the Chains algorithm
Combine the solutions found for the sub-problems and the recursive problem.
**General Case**

- Theorem: The algorithm runs in $O(n^3)$ time and finds an interleaving plan with at most $4n$ distinct permutations.
Outline

• Introduction and Motivation

• Problem Definition

• Algorithms for finding Optimal Interleaving Plans
  – Selective operators
  – Non-selective operators

• Experimental Results
Non-selective Operators

• The saturated suffix lemma does not hold:
  – *Saturation does not imply optimality*

• Summary of results:
  – All non-selective operators and tree-structured precedence constraints
    • Can be solved using the same algorithm
  – Mixture of selective and non-selective operators
    • $O(n^2 \log n)$ algorithm for when $G$ is a forest of chains
  – General case still open
    • Known to be polynomial
Outline

• Introduction and Motivation

• Problem Definition

• Algorithms for finding Optimal Interleaving Plans
  – Selective operators
  – Non-selective operators

• Experimental Results
Performance Study

• Compared Techniques:
  – OPT-SEQ: Serial plan found using rank ordering
    • Optimal for centralized case
  – BOTTLENECK [Srivastava et al.; 2006]
    • Optimal serial plan for parallel execution
  – MTTC: Proposed algorithm

• Setup:
  – Synthetic datasets: costs and selectivities chosen randomly
  – Different query types: star, path, random

• Comparison metrics:
  – Response time (total time to execute the query)
  – Total work (across all processors)
Performance Study

• Star queries (1)

(i) Sel in [0, 1], Costs in [1, 1]

(ii) Sel in [0, 1], Costs in [1, 1]
Performance Study

• Star queries (2)
Performance Study

- Path queries, randomly generated query graphs

(i) Path Queries

(ii) Random Queries
Conclusions and Future Work

• Proposed *interleaving plans* to fully exploit parallelism in a database system
• Fast algorithms for finding optimal *interleaving plans*
  – Use few permutations, so easy to deploy

• Open questions:
  – Cyclic precedence constraints
  – Correlated predicates
  – Other types of queries (Bushy plans, MJoins)

• Thank you !!
**Reduction**

- Given a multi-way join query and a driver relation, choosing a left-deep plan is equivalent to precedence-constrained selection ordering.

**Example query**

```sql
select *
from R1, R2, R3, R4, R5
where R1.a = R2.a and
R1.b = R3.b and
R1.c = R4.c and
R4.d = R5.d
```
Algorithms for finding serial plans

• Rank ordering: Order the operators in the increasing order of rank = ci/(1 – pi)
  – Optimal in a centralized scenario
  – Oblivious to the parallelization

• Bottleneck [Srivastava et al.; 2006]: Order the operators in the decreasing order of ri
  – Finds the optimal serial plan in the parallel setting for selective operators
  – Different plan space considered for non-selective operators
Saturation $\rightarrow$ Optimality

- **Case: Full Saturation**
  - All operators are processing at their rate limit
  - Precedence constraints irrelevant
- **Proof:**
  
  \[
  K = \sum r_i (1 - p_i) / (1 - p_1 \cdots p_n) = \text{Constant}
  \]

  If throughput equal to $K$,
  then total tuples rejected in unit time
  
  \[
  = K (1 - p_1 \cdots p_n)
  \]

  Combined rejection probability

  Total tuples rejected in unit time
  Rejects $\sum r_i (1 - p_i)$ tuples