Flow Algorithms for Two Pipelined Filtering Problems

Anne Condon, University of British Columbia
Amol Deshpande, University of Maryland
Lisa Hellerstein, Polytechnic University, Brooklyn
Ning Wu, Polytechnic University, Brooklyn
Pipelined Filter Ordering

- Query processors commonly need to evaluate complex predicates against relations
  - Eg., on an employee relation:
    $((\text{s}alary > 120000) \ \text{AND} \ (\text{status} = 2)) \ \text{OR} \ (\text{educationlevel} = 3);$
    $(\text{age} < 30) \ \text{AND} \ \text{hasPatents}(\text{emp}) \ \text{AND} \ \text{hasWrittenBooks}(\text{emp});$

- Pipelined filter ordering problem
  - Decide the order in which to apply the individual predicates (filters) to the tuples

- We will focus on evaluating conjunctive predicates (containing only ANDs)

**Example Query**

```
select *
from employee
where age < 30 and
    salary < 100000 and
    zipcode = 20001
```
Classical Pipelined Filter Ordering

Given:
- A set of independent filters, \( \text{pred}_i \)
- cost of applying each filter, \( c_i \)
- selectivity of each filter, \( p_i \)

Find:
- the optimal serial execution order for applying the filters that minimizes the total execution cost on a single processor

Example Query

```sql
select *
from employee
where age < 30 and
  salary < 100000 and
  zipcode = 20001
```

Serial Plans Considered

```
age < 30 --> salary < ... --> zipcode = ...

salary < ... --> age < 30 --> zipcode = ...
```

3! = 6 plans considered
Classical Pipelined Filter Ordering

Given:
- A set of independent filters, \( \text{pred}_i \)
- \( \text{cost} \) of applying each filter, \( c_i \)
- \( \text{selectivity} \) of each filter, \( p_i \)

Find:
- the optimal serial execution order for applying the filters that minimizes the total execution cost on a single processor

Example Query
```sql
select *
from employee
where age < 30 and
  salary < 100000 and
  zipcode = 20001
```

Execution cost of a plan
- \( \text{costs} \): \( c_1, c_2, c_3 \)
- \( \text{selectivities} \): \( p_1, p_2, p_3 \)

Expected cost per tuple
\[
\text{Expected cost per tuple} = c_1 + p_1 c_2 + p_1 p_2 c_3
\]
Classical Pipelined Filter Ordering

- Algorithm for *independent filters* [KBZ’86]
  - Apply the filters in the increasing order of:
    \[(1 - p_i) / c_i\]
  - \(O(n \log (n))\)

- Correlated filters?
  - NP-hard under several formulations
    - E.g. when asked to find the best order for a given set of tuples
  - 4-Approx greedy algorithm [BMMNW’04]
Pipelined Filter Ordering

- Complex expensive predicates
  - E.g. `pointInPolygon(x, y, P), hasPatents(emp)`
- Many join queries reduce to this problem
  - E.g. queries posed against a *star schema*
- Increasing interest in recent years
  - Data streams [AH’00, BMMNW’04]
  - Sensor Networks [DGMH’05]
  - Web indexes [CDY’95, EHJMKW’96, GW’00]
  - Web services [SM’06]
- Similar to many problems in other domains
  - Sequential testing (e.g. for fault detection) [SF’01, K’01]
  - Learning with attribute costs [KKM’05]
Outline

- Introduction
- Problem 1: Max-throughput problem
  - Maximize *throughput* (tuples processed per unit time) in a parallel environment
- Problem 2: Adversarial type, Single Tuple
  - Minimize *multiplicative regret* in an adversarial setting
Problem 1: Max-throughput Problem

- \( n \) independent filters executed \textit{in parallel} by \( n \) operators, \( O_1, \ldots, O_n \)
- Given:
  - selectivities of the filters, \( p_i \)
  - rate limits of the operators, \( r_i \)
- Goal: Maximize the number of tuples processed per time unit

Parallel Databases

\begin{align*}
\text{proc 1} & \quad \text{age} < 30 \\
\text{proc 2} & \quad \text{salary} < \\
\text{proc 3} & \quad \text{zipcode} \\
\end{align*}

Queries over web services

- Person
  - John
  - Jane

- Query Processor

- Google
- Patents Database
- Amazon

\( R \)
Max-throughput Problem

\[ r_1 = 4 \quad \text{and} \quad p_1 = \frac{1}{2} \]
\[ r_2 = 4 \quad \text{and} \quad p_2 = \frac{1}{2} \]
\[ r_3 = 4 \quad \text{and} \quad p_3 = \frac{1}{2} \]

**Best “single” execution order?**

- 4 tuples/unit time
- 2 tuples/unit time
- 1 tuple/unit time

Throughput = 4 tuples

Idle
Max-throughput Problem

Use two orders: (1, 2, 3) and (3, 2, 1)

\[
\begin{align*}
    r_1 &= 4 \\
    p_1 &= \frac{1}{2} \\
    X &= \frac{16}{5} \\
    \text{Throughput} &= \frac{32}{5} \text{ tuples}
\end{align*}
\]

Send Y tuples through (1, 2, 3), (2, 3, 1), and (3, 1, 2)

\[
\begin{align*}
    Y &= \frac{16}{7} \\
    \text{Throughput} &= \frac{48}{7} \text{ tuples}
\end{align*}
\]

Best possible ??
Max-throughput Problem

- Definition:
  - Saturation = an operator is processing at its capacity

- Lemma: Full saturation $\rightarrow$ optimality

- Proof:

  If throughput equal to $K$, then total tuples rejected in unit time
  
  \[ r_{1}(1 - p_{1}) + r_{2}(1 - p_{2}) + \cdots + r_{n}(1 - p_{n}) = K (1 - p_{1}\cdots p_{n}) \]

  \[ K = \sum r_{i} (1 - p_{i}) / (1 - p_{1}\cdots p_{n}) = \text{Constant} \]
Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
- Adversarial type, Single Tuple
Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
    - Equal rates, unequal selectivities
    - Unequal rates and selectivities
      - Two operators
      - General case
  - Routing and queuing issues
- Adversarial type, Single Tuple
Summary of results

- Max-throughput problem
  - $O(n)$ algorithm to find the maximal achievable throughput, if rates given in sorted order
  - $O(n^2)$ algorithm to find the optimal solution
  - Previously known best algorithms [Kodialam’01]
    - From sequential testing literature
    - $O(n^2)$ and $O(n^3 \log n)$ respectively
Equal rates, unequal selectivities

- Closed form *saturating* solution using *cyclic permutations*
- Send:
  \[
  \frac{(1 - p_{j-1})}{(n - \sum p_i)}
  \]
  tuples through permutation:
  \[(j, j+1, j+2, \ldots, n, 1, \ldots, j-1)\]

\[
\begin{align*}
rate = 1
\end{align*}
\]

\[
\begin{align*}
p_1 & \quad p_2 & \quad p_3 & \quad \cdots & \quad p_n
\end{align*}
\]
Unequal rates: Two Operators

Two operators with unequal rates ($r_1 > r_2$)

Start adding flow through $(1, 2)$

Either:
- residual rates equalize
  - (if $r_1 p_1 > r_2$)

Or:
- $O_2$ saturates, but $O_1$ doesn’t
Unequal rates: Two Operators

Two operators with unequal rates ($r_1 > r_2$)

Start adding flow through $(1, 2)$

Either:
residual rates equalize

$r_1, p_1$

(if $r_1 p_1 > r_2$)

Use the equal-rates algorithm to solve the residual problem

Two part solution:
During every time unit,
1. Send $x$ tuples through $(1, 2)$
2. Send $y$ tuples total through cyclic permutations: $(1, 2)$ and $(2, 1)$ (appropriately divided)
Unequal rates: Two Operators

Two operators with unequal rates \((r_1 > r_2)\)

Start adding flow through \((1, 2)\)

\[ r_1, p_1 \quad \begin{array}{c} \text{O}_1 \\ \text{x} \end{array} \quad p_1 \begin{array}{c} \text{O}_2 \\ \text{x} \end{array} \quad r_2, p_2 \]

Optimal

1. \(\text{O}_2\) has saturated
2. No flow out of \(\text{O}_2\)

Solution:
During every time unit, send \(z\) tuples through \((1, 2)\)

Or:
\(\text{O}_2\) saturates, but \(\text{O}_1\) doesn’t

\(\text{O}_2\) is being applied last to all tuples. No way to lighten its load more.
Unequal rates: General Case

- Create equivalence groups of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if residual rates equalize

Partial solution:
1. send $x$ tuples along $(1, 2, 3, 4, 5)$
Unequal rates: General Case

- Create equivalence groups of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if residual rates equalize

Partial solution:
1. send $x$ tuples along $(1, 2, 3, 4, 5)$
Unequal rates: General Case

- Create equivalence groups of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if residual rates equalize
- Recurse using residual rates

Partial solution:
1. send x tuples along (1, 2, 3, 4, 5)

Use equal-rates solution to distribute tuples evenly
Unequal rates: General Case

- Create equivalence groups of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if residual rates equalize
- Recurse using residual rates

Use equal-rates solution to distribute tuples evenly:

Partial solution:
1. send $x$ tuples along $(1, 2, 3, 4, 5)$
2. send $y$ tuples total along:
   - $(1, 2, 3, 4, 5)$ and
   - $(1, 3, 2, 4, 5)$
   (appropriately divided)
Unequal rates: General Case

- Create equivalence groups of operators based on rates
- Incrementally add flow from higher-rate groups towards lower-rate groups
- Merge groups if residual rates equalize
- Recurse using residual rates

- After at most $n$ rounds
  - **Either**: All operators are equalized
    - Solve using equal rates solution
    - *Full saturation* $\rightarrow$ Optimality
  - **Or**: Rightmost equivalence group saturates
    - *No flow out of that group* $\rightarrow$ Optimality

- Running time: $O(n^2)$
Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
    - Equal rates, unequal selectivities
    - Unequal rates and selectivities
      - Two operators
      - General case
  - Routing and queuing issues
- Adversarial type, Single Tuple
Routing

- Ideal form of the final solution:
  - $\left( \pi_i, f_{\pi_i} \right)$: a list of permutations, and associated flows
  - Not feasible: The resulting solution is not necessarily sparse

- Can directly use the output of the algorithm (the equivalence groups and associated flows) for routing tuples

Positive flow assigned to $2^k$ permutations

$(1\ 2) \ (3\ 4) \ \ldots \ (2k-1\ 2k)$

![Diagram](image-url)
Queuing issues

- Rate limits only guaranteed in expectation

- [Kodialam’01]
  - If $K$ is an optimal routing scheme with max throughput $F$, then, for any $F^* < F$, there is a routing scheme $K^*$ with throughput $F^*$ that obeys the rate limits
Outline

- Introduction
- Max-throughput problem
  - Formulation in terms of flows
  - Algorithms
- Adversarial type, Single Tuple
Problem 2: Adversarial, Single Tuple

- Given $n$ predicates and their costs, $c_1, \ldots, c_n$
- For tuple $t$, let:

\[
\text{multiplicative regret}(t) = \frac{\text{actual cost of processing } t}{\text{minimum cost of processing } t}
\]

- Adversary knows algorithm used, and controls input tuples
- **Goal:** Minimize the expected multiplicative regret
Problem 2: Adversarial, Single Tuple

- Can be reduced to a problem similar to max-throughput problem
  - Details in the paper
  - Same algorithms can be used

- Naïve solution
  - Order in the increasing order of costs

- Theorem: Naïve solution within a factor of 2
Conclusions

- Two pipelined filter ordering problems
  - Maximize throughput in a parallel scenario
  - Minimize multiplicative regret in an adversarial scenario
- Very simple and efficient algorithms
  - Using a flow formulation
- Interesting open problems
  - Correlated predicates
  - Robustness of algorithms
  - Routing tuples of multiple types together
Thank you!!

Questions?