Optimization of Query Streams Using Semantic Prefetching

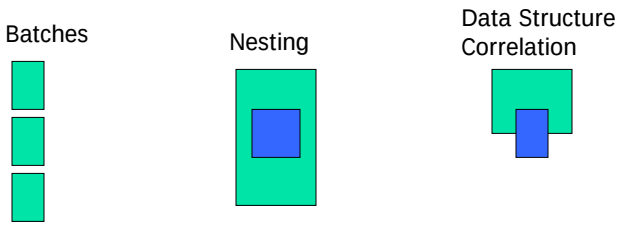
Ivan T. Bowman
School of Computer Science
University of Waterloo

Kenneth Salem
School of Computer Science
University of Waterloo

Presented By : *Ahmed A. Soror*

1. Behind The Scenes

- Query Streams
  - A request stream of open, fetch, close.
  - Single stream/connection.
- Query Patterns



1. Behind the Scenes

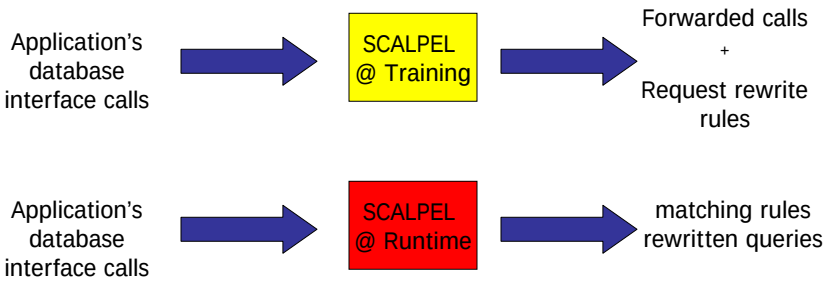
- Optimization
  - Rewriting Semantically related query.
  - Predictively execute new queries.
  - Reduce communication and system interface layers latency and overhead.
- Why not manually ..?
  - Dynamically changing semantics.
  - Manual tuning costs.
  - Decoupling implementation from application logic.

1. Behind the Scenes

- Semantic Prefetching
  - Execute predicted semantic (context related) queries before actual call.
- Updates after prefetches
  - Same connection : invalidate prefetches
  - Other connections : serializable isolation level

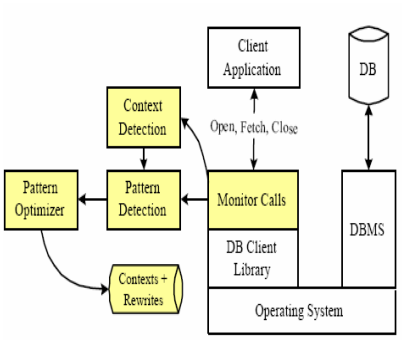
2. Introducing .. SCALPEL

- A system for detecting and optimizing patterns of repeated requests within a query stream.



3. Training Scalpel

- Context Detection
  - Monitor application requests stream.
  - Track evolving request context.
- Pattern detection
  - Detect correlation between queries and their context
- Pattern optimizer
  - Rewrite cost efficient patterns



3. Training Scalpel (Example)

```
function get_openinvoices( cust_id, currency )
    // this is Q1
    c1 = open( SELECT id, curr, transdate
               FROM   ar
               WHERE  customer_id = :cust_id
                 AND  ar.curr = :currency
                 AND  NOT ar.amount = ar.paid
               ORDER BY id )
    while( r1 = fetch(c1) )
        ...
        if( currency != defaultcurrency )
            rate = get_exchangerate( r1.curr,
                                     r1.transdate );
end

function get_exchangerate( curr, transdate )
    // this is Q2
    r2 = fetch( SELECT exchangerate FROM exchangerate
                WHERE  curr = :curr
                  AND  transdate = :transdate )
    return r2.exchangerate;
end
```

Training Process

```
SELECT id, curr, exchangerate, ...
FROM ar LEFT JOIN exchangerate er
     ON ar.curr = er.curr
     AND ar.transdate=er.transdate
WHERE  customer_id = :cust_id
       AND  ar.curr = :currency
       AND  NOT ar.amount = ar.paid
ORDER BY id
```

Initial application side Query

Optimized Combined Query

3. Training Scalpel (Example)

Trace
OPEN(Q_1(cust1,CDN$)) → c1
FETCH(c1) → (3305,CDN$,30/10/03)
OPEN(Q_2(CDN$,30/10/03)) → c2
FETCH(c2) → 1.4304
CLOSE(c2)
FETCH(c1) → (3307,CDN$,30/10/03)
OPEN(Q_2(CDN$,30/10/03)) → c2
FETCH(c2) → 1.4304
CLOSE(c2)
FETCH(c1) → (3308,CDN$,31/10/03)
OPEN(Q_2(CDN$,31/10/03)) → c2
FETCH(c2) → 1.4522
CLOSE(c2)
CLOSE(c1)

Context Detection

Query Context	Parameter Context
-	-
Q_1	(cust1,CDN$,-,-,-)
Q_1	(cust1,CDN$,3305,CDN$,30/10/03)
Q_1,Q_2	(cust1,CDN$,3305,CDN$,30/10/03),(CDN$,30/10/03,-)
Q_1,Q_2	(cust1,CDN$,3305,CDN$,30/10/03),(CDN$,30/10/03,1.4304)
Q_1	(cust1,CDN$,3305,CDN$,30/10/03)
Q_1	(cust1,CDN$,3307,CDN$,30/10/03)
Q_1,Q_2	(cust1,CDN$,3307,CDN$,30/10/03),(CDN$,30/10/03,-)
Q_1,Q_2	(cust1,CDN$,3307,CDN$,30/10/03),(CDN$,30/10/03,1.4304)
Q_1	(cust1,CDN$,3307,CDN$,30/10/03)
Q_1	(cust1,CDN$,3308,CDN$,31/10/03)
Q_1,Q_2	(cust1,CDN$,3308,CDN$,31/10/03),(CDN$,31/10/03,-)
Q_1,Q_2	(cust1,CDN$,3308,CDN$,31/10/03),(CDN$,31/10/03,1.4522)
Q_1	(cust1,CDN$,3308,CDN$,31/10/03)

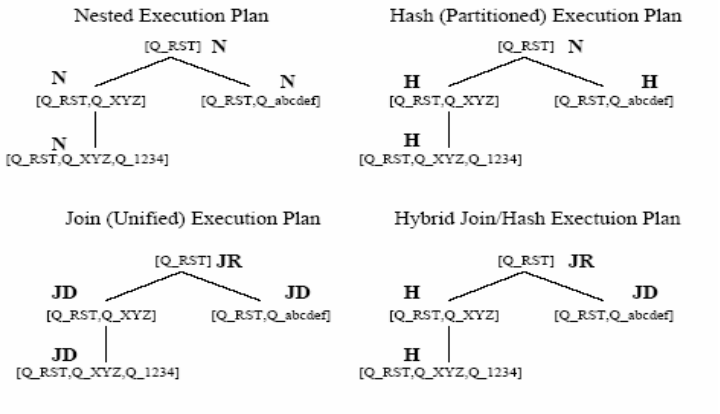
P.D

$[Q_1],Q_2$

3. Pattern Optimizer

- Plan generator
  - Contexts identified may be related to on another.
  - Enumerate all possible axecution alternatives
    - Nested Execution (as-is)
    - Partitioned Execution (rewritten inner query hash joined with outer query at client)
    - Unified execution (join inner query with its context)
- Ranking Module
  - Response time cost based ranking

3. Pattern Optimizer (Plan generator)



3. Pattern Optimizer (Ranking Module)

$$\textsc{Context-Cost}(C) = \textsc{Opens}(C) \times \textsc{Cost}(Q')$$

$$\textsc{Context-Cost}(C) = P_0(C)\textsc{Opens}(C_0)\left[\textsc{Cost}(Q') + |Q'|H_{\text{add}}\right] + H_{\text{find}}\textsc{Opens}(C)$$

$$\textsc{Opens}(C) = \begin{cases} 1 & : \; C = [Q] \\ P(C)\,|Q_0|\,\textsc{Opens}(C_0) & : \; C = [C_0, Q_0] \end{cases}$$

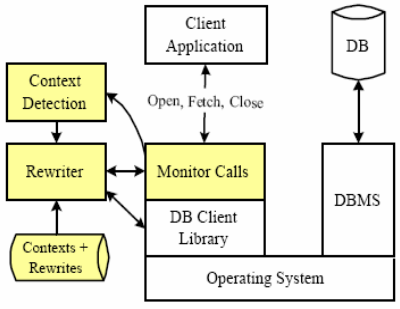
$$\textsc{Cost}(Q) = U_0 + \textsc{Server-Cost}(Q) + \textsc{Comm-Cost}(|Q|, \textsc{Bytes}(Q))$$

$$P(C) = P_0(C)P_1(C)$$

Quantity	Source	Description
$\textsc{Server-Cost}(Q)$	RDBMS	Server costs for Q in seconds
$\|Q\|$	RDBMS	Rows returned by Q
$\textsc{Bytes}(Q)$	RDBMS	Average row length for Q
$\textsc{Comm-Cost}(N, B)$	Scalpel	Communication latency for N rows of B bytes
U_0	Scalpel	Overhead of a single request
H_{add}	Scalpel	Cost of adding to hash table
H_{find}	Scalpel	Cost of finding in hash table
P_0, P_1	Scalpel	Selectivity of client predicates

4. Running Scalpel

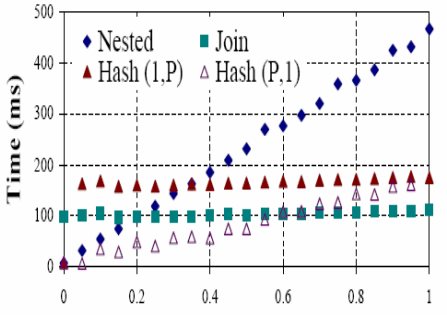
- On context detection Rewriter issues new optimized query.
- Further, Scalpel intercepts predicted calls and replies with prefetched results



5. Experiments

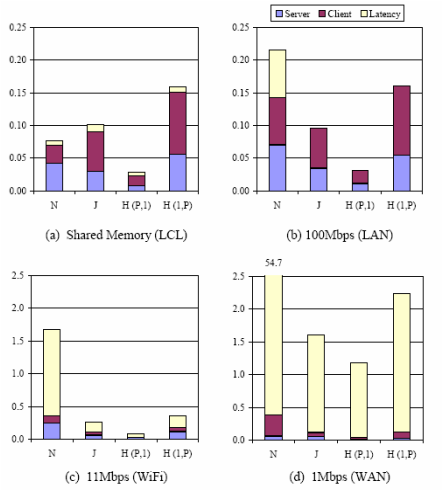
- Effect of client predicate selectivity

- For small selectivity, nested is most efficient.
- Hash strongly depends on P_0 selectivity.



Experiments

- Execution Costs
  - Nested execution is optimal for local connection.
  - Server costs are 50% lower with joined variants.
  - Nested outperformed in deployments with higher network latency.



Concluding remarks

- Tolerated overhead.
- Optimizing stream instead of individual requests.
- Providing more flexibility to the optimizer.
- Multi-query optimization.
- Is Nesting + stored procedure competitive..?
- Different cost models (resource consumption)
- Why scalpel..?
- Other partitioning, join techniques
- Why log the most recently fetched query results..?
- How long does it take to train it..?

Thank You