Optimizing Continuous Queries Using Update Propagation with Varying Granularities

Andreas Behrend, Ulrike Griefahn, Hannes Voigt, Philip Schmiegelt
SSDBM 2015
Fast Temporal In-Database Analytics

New tasks for database systems:

- Scalable stream processing
- Reliable storage
- Total recall
- Retrospective
- Real-time
- Predictive
- Continuous analysis
- Customized analytics
- Evolving knowledge
Limitations of Datastream Systems (DSMS)

- DSMS are limited systems (main memory-based, no transactions, no multi-user access, no history, ...)
- DSMS provide only limited query optimizations (Magic Sets for joining streams with views?)
- DSMS offer almost no high-level analytics
Synopsis

What?

- Temporal data management is provided by most DBMS by now (bitemporal data).
- Database vendors are integrating stream processing technology (window expressions).
- Main-memory computation allow for reasonable performance.

Problems?

- How to evaluate continuous queries (CQ) in a database system? => Incremental Evaluation!
- How to optimize CQs referencing views over static domains data? => Magic Sets!
- How can these strategies be effectively combined? => ?

- A framework for optimizing CQs in a DBMS
- Showing its benefits for optimizing graph queries
Airspace Monitoring System (AIMS)

- Which aircrafts approach each other critically?
- Which aircrafts are landing?
- Which planes are late?
- Which flight are entering areas with bad weather conditions?
- Which planes are changing their course unexpectedly?
- What is the average number of flights for a given area?
An aircraft is considered to be **landing** if

- its vertical velocity is negative
- its flight level is below zero
- it is currently approaching an airport less than 20 miles away.

```sql
CREATE CONTINUOUS QUERY landings AS
SELECT f.*
FROM flights AS f [NOW], airports a
WHERE f.Incline < 0 AND
  f.altitude < 'FL0' AND
  getDist(f.Pos,a.Pos)<20000
```

<table>
<thead>
<tr>
<th>Flight Code</th>
<th>Time</th>
<th>Speed</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACA876</td>
<td>17:03</td>
<td>-4.2</td>
<td>52.51</td>
<td></td>
</tr>
<tr>
<td>ACA876</td>
<td>17:07</td>
<td>-4.8</td>
<td>51.33</td>
<td></td>
</tr>
<tr>
<td>ACA876</td>
<td>17:11</td>
<td>-4.9</td>
<td>50.21</td>
<td></td>
</tr>
<tr>
<td>ACA876</td>
<td>17:13</td>
<td>3.9</td>
<td>15.21</td>
<td></td>
</tr>
</tbody>
</table>

**Input Stream**

**Output Stream**

ACA876 landing [17:03]
ACA876 landing [17:07]
ACA876 landing [17:11]
SQL query:

```
CREATE CONTINUOUS QUERY landings AS
SELECT f.*
FROM flights AS f [NOW], airports a
WHERE f.Incline < 0 AND
  f.altitude < 'FL0' AND
  getDist(f.Pos,a.Pos)<20000
```

Datalog rules:

```
Landings(ID)@t ← flights(ID,Incl,_,Alt,Pos_1)@t,
  Incl < 0, Alt < 'FL0',
  airports(_,Pos_2),getDist(Pos_1,Pos_2)<20
```
Hierarchies of CQs allow for describing complex analysis criteria:

- **Input data stream**
  - `landings(ID)@t ← flights(ID,...)@t, Incl < 0, ...`
  - `LAX_Approach(ID)@t ← landings(ID)@t, LAXpos(Pos), ...`
  - `LAX_NearMiss(ID1,ID1)@t ← LAX_Approach(ID)@t, encounter(ID1,ID2), ...`

Views over static domain data
Evaluating CQs incrementally by **pushing up** new tuples covered by the time window:

- `+landings(ID)@t ← flights(ID,...)@t, Incl < 0, ...`
- `+LAX_Approach(ID)@t ← landings(ID)@t, LAXpos(Pos), ...`
- `+LAX_NearMiss(ID1,ID1)@t ← LAX_Approach(ID)@t, encounter(ID1,ID2), ...`
A transformation-based approach for obtaining incremental versions of the CQs:

\[
A \leftarrow L_1, L_2, \ldots, L_n
\]

\[
A^+ \leftarrow L^+_1, L_2, \ldots, L_n
\]

... 

\[
A^+ \leftarrow L_1, L^+_2, \ldots, L_n
\]

Classical push approach in stream processing!

=> not goal-directed?

=> costly side evaluation of referenced views?
Determining dynamic selection conditions for an efficient evaluation of side paths using Magic Sets:

- `+LAX_NearMiss(ID1,ID1)@t ← LAX_Approach(ID)@t, encounter(ID1,ID2), …`
- `+LAX_Approach(ID)@t ← landings(ID)@t, LAXpos(Pos), …`
- `+landings(ID)@t ← flights(ID,...)@t, Incl < 0, …`

Only relevant parts of the views are needed!
A transformation-based approach for optimizing incremental CQ versions using Magic Sets:

- Push approach with intermediate pulls
- Many new joins introduced this way

)=> Can we do better?
Idea: Determine potential results and use them as global selection conditions:

**Small set of hints for top CQs**

- Time window
  - Flights (ID, ...): landings (ID) at time `t`, Incl < 0, ...
  - LAX Approach (ID): LAX landings (ID) at time `t`, LAXpos (Pos), ...
  - LAX Near Miss (ID1, ID1): LAX Approach (ID1) at time `t`, encounter (ID1, ID2),...

**Query relevant?**

```
+LAX_Approach(ID) at t ←
  landings(ID) at t, LAXpos(Pos), ...
```

```
+LAX_NearMiss(ID1, ID1) at t ←
  LAX_Approach(ID1) at t, encounter(ID1, ID2), ...
```

```
+landings(ID) at t ←
  flights(ID, ...) at t, Incl < 0, ...
```
A transformation-based approach for incorporating hints into incremental CQ:

\[
\begin{align*}
A & \leftarrow L_1, L_2, \ldots, L_n \\
A^+ & \leftarrow L_{+1}, L_2, \ldots, L_n \\
A^+ & \leftarrow L_1, L_{+2}, \ldots, L_n \\
\vdots & \leftarrow L_1, L_2, \ldots, L_{+n} \\
\end{align*}
\]

Small set of hints

\[
\begin{align*}
H^+ & \leftarrow L_{+1}, L_2, \ldots, L_n \\
H^+ & \leftarrow L_1, L_{+2}, \ldots, L_n \\
\vdots & \leftarrow L_1, L_2, \ldots, L_{+n} \\
\end{align*}
\]

\[
\begin{align*}
A^+ & \leftarrow H^+, L_{+1}, a_{L_2}, \ldots, a_{L_n} \\
A^+ & \leftarrow H^+, L_{+2}, a_{L_1}, \ldots, a_{L_n} \\
\vdots & \leftarrow H^+, L_n, a_{L_1}, \ldots, a_{L_n} \\
A^+ & \leftarrow H^+, L_{+n}, a_{L_1}, \ldots, a_{L_n} \\
\end{align*}
\]
The task is to detect **disconnections** between backbone connections stored in $b$:

**Monitoring Disconnections:**

\[
\begin{align*}
di (X, Y) & \leftarrow b (X, Y), \neg p (X, Y) \\
p (X, Y) & \leftarrow e (X, Y) \\
p (X, Y) & \leftarrow e (X, Z), p (Z, Y)
\end{align*}
\]
Either pull or push will generate many irrelevant intermediate results:

\[ +di(X,Y) \leftarrow -p(X,Y), b(X,Y) \]
The hints produced by the PaQ Approach considerably simplify the reasoning:

Hints $X=7, b_3$ or $b_4$

- $-p(b3,b4)$

Time window

- $-e(b_3,6)$
Hints sometimes allow for early ending the entire answering process:

Hints $X=1$
A hint for variable X is computed using the most simplified version of the CQ containing X:

\[ a(X,Y,Z) \leftarrow c(Y,Z), \neg p(Y), s(X,Y), X<>2, Y>4, Z<>Y \]

Hints about X values for new tuples in a stream s:

- drop all side literals
- drop all selection cond.
- project on hint variables
- reverse sign for negations

\[ +h_{bff}(X) \leftarrow +s(X,Y), X<>2 \]
Hints are added to the incremental versions of the CQs which allows for **simplifying** their structure:

Original incremental CQ:

\[ +a(X,Y,Z) \leftarrow +s(X,Y), c(Y,Z), \neg p(Y), X\neq 2, Y>4, Z\neq Y, \neg a_{\text{old}}(X,Y,Z) \]

Incremental CQ containing hints:

\[ +a(X,Y,Z) \leftarrow +h_{\text{bff}}(X), c(Y,Z), a_{\text{new}}(X,Y,Z) \]

- hints are evaluated first
- collect further conditions, e.g., \( c(Y,Z) \)
- add effectiveness test, e.g., \( a_{\text{new}}(X,Y,Z) \)
A transformation-based framework for Optimizing CQs in DBMS

- Computing hints for supporting a pull approach
- Hints allows for reducing the number of intermediate results
- The resulting incremental rules have simpler structure such that ...
  - ... less joins become necessary
  - ... the introduction of recursion can be avoided
  - ... the introduction of unstratifiability can be avoided

Future Work
Refining PaQ for optimizing CQs over graphs databases.

Thank you for your attention!
Backup Slide
Hints sometimes lead to considerable computation overhead:

Hints $X=1,3,4,5,6,7, b_1,b_2,b_3,b_4$

+di(X,Y) ← -p(X,Y), b(X,Y)

Time window

- e(3,4)