DBToaster
Higher-Order Delta Processing for Dynamic, Frequently Fresh Views

Yanif Ahmad  Johns Hopkins
Oliver Kennedy  EPFL
Christoph Koch  University at Buffalo
Milos Nikolic  EPFL
Realtime Monitoring Programs…

…Monitor The State of the World

…React to Conditions in that State
Realtime Monitoring Programs are Everywhere
Monitoring Programs
Monitoring Programs

Problem: People write monitoring programs by hand
Problem: People write monitoring programs by hand
Monitoring Programs

View

(java code)
Monitoring Programs

• An Aggregate Representation of the State of the World
• Maintained in Realtime as the State of the World Changes
• Needs to React to Changes In the World Quickly

Not just Views

Frequently Fresh Views
Monitoring Programs

View → Javac
Monitoring Programs

(The Current State of the Art)
Monitoring Programs

Spec → Javac → Embedded SQL
Monitoring Programs
Monitoring Programs
The Viewlet Transform
The Viewlet Transform

Use Auxiliary Views to Speed Up View Maintenance
The Viewlet Transform

Use Auxiliary Views to Speed Up View Maintenance

The Delta of a Query Can Be Materialized!
The Viewlet Transform

```
SELECT SUM(R.A * S.C)
FROM R, S
WHERE R.B = S.B
```
The Viewlet Transform

$q[] := \text{SELECT} \ \text{SUM}(R.A \ast S.C) \\
\text{FROM} \ R, S \\
\text{WHERE} \ R.B = S.B$
The Viewlet Transform

\[
\text{ON } +R(\partial A, \partial B): \\
q[] \ + = \ \text{SELECT SUM}(\partial A \ast S.C) \\
\text{FROM } S \\
\text{WHERE } \partial B = S.B
\]
The Viewlet Transform

ON \( +R(\partial A, \partial B) : \)

\[
q[] \ += \ \partial A \times \left( \begin{array}{l}
\text{SELECT} \ \text{SUM}(S.C) \\
\text{FROM} \ S \\
\text{WHERE} \ \partial B = S.B
\end{array} \right)
\]
The Viewlet Transform

ON +R(∂A, ∂B):

\[ q[] += ∂A \times \left( \begin{align*}
SELECT & \quad \text{SUM}( S.C ) \\
\text{FROM} & \quad S \\
\text{WHERE} & \quad ∂B = S.B
\end{align*} \right) \]
The Viewlet Transform

ON \( +R(\partial A, \partial B) \):

\[
q[] += \partial A \times \left( \text{SELECT } S.B, \text{SUM}(S.C) \right)_{[\partial B]} \left( \text{FROM } S \right)
\text{GROUP BY S.B}
\]
The Viewlet Transform

ON +R(∂A, ∂B):
q[] += ∂A * mR[∂B]

mR[B] := SELECT S.B, SUM(S.C)
FROM S
GROUP BY S.B

Extract and Materialize The Delta View
The Viewlet Transform

ON +R(∂A, ∂B):

q[] += ∂A * mR[∂B]

A Hash Map (indexed by S.B)

mR[B] := SELECT S.B, SUM(S.C)
FROM S
GROUP BY S.B

Extract and Materialize The Delta View
The Viewlet Transform

ON \( +R(\partial A, \partial B) : \)

\( q[] += \partial A \ast mR[\partial B] \)

ON \( +S(\partial B, \partial C) : \)

\( mR[B] += SELECT \partial B, SUM(\partial C) \)

Incrementally Maintain The Delta View
The Viewlet Transform

\[ \partial A \neq q[] + \partial A \ast mR[\partial B] \]

\[ \partial B \neq S(\partial B, \partial C) : \]
\[ mR[\partial B] += \partial C \]

Optimize
The Viewlet Transform

\[ \partial A \partial C^* \frac{q[]}{\partial B} + = \partial B \frac{m_R[\partial B]}{\partial A} \]

\[ m_S[\partial B] + = \partial A \]

ON +R(\partial A, \partial B):

\[ q[] + = \partial A \times m_R[\partial B] \]

\[ m_S[\partial B] + = \partial A \]

ON +S(\partial B, \partial C):

\[ m_R[\partial B] + = \partial C \]

\[ q[] + = \partial C \times m_S[\partial B] \]

Repeat for the Other Deltas of the Query
The Viewlet Transform
The Viewlet Transform

• Take the Deltas
  • Optimize and Materialize Them
• Take the Deltas
  • Optimize and Materialize Them
• ...
The Viewlet Transform

- Take the Deltas
  - Optimize and Materialize Them
- Take the Deltas
  - Optimize and Materialize Them
- ...

Performance
(and how we got there)
• TPC-H Workload

• Simulated Realtime Data Warehouse

• Update Stream Derived from TPC-H Gen

• Financial Benchmark

• 24 hr Trace for an Actively Traded Stock.
TPCH: Q3

Refresh Rate

(3-Way Join)

Refresh Rate (tuples/sec) vs. Fraction of Trace Completed

- Naive Re-evaluation
- Traditional IVM
- DBToaster
TPCH: Q3
TPCH: Q3

Customer

CustID

OrderID

Lineitem

The Δ for Orders
TPCH: Q3

Materialize Each Separately

Customer

Lineitem

CustID

OrderID

The Δ for Orders
Financial: VWAP

Refresh Rate

(Self-join with Inequalities)
Financial: VWAP

ON +BIDS(..., ∂price, ...)

q[] += SELECT ...
    FROM BIDS b2
    WHERE ∂price > b2.price
ON +BIDS(..., \partial\text{price}, ...) 
q[] += mB[\partial\text{price}]

mB[\partial\text{price}] := SELECT ...
    FROM BIDS b2
    WHERE \partial\text{price} > b2.\text{price}

Option 1: Create a Cache (best for VWAP)
Financial: VWAP

\[ q[] + \text{SELECT} \ldots \]
\[ \text{FROM} \ mB[] \]
\[ \text{WHERE} \ \partial \text{price} > b2.\text{price} \]

\[ mB[] := \text{SELECT} \ldots \]
\[ \text{FROM} \ BIDS \ b2 \]

Option 2: Defer Conditions Over Unsafe Variables
Financial: Pricespread

Refresh Rate

(Cross Product ‘variance’ Computation)
Financial: Pricespread

- 2-way Cross-Product with Nested Aggregates
- IVM can’t do better than Repeated re-evaluation.
- DBToaster wins on Data Representation Trickery!
Financial: Pricespread

\[ \text{Sum} \leftarrow; (R.A - S.B) \Rightarrow (R \boxtimes S) \]
Financial: Pricespread

\[
\text{Sum} \leftrightarrow; (R.A-S.B) \quad (R \Join S)
\]

\[
\text{Sum} \leftrightarrow; R.A \quad U \quad \text{Sum} \leftrightarrow; -S.B \quad (R \Join S)
\]
Financial: Pricespread

\[
\text{Sum} \lll; (R.A-S.B) (R \Join S)
\]

\[
\text{Sum} \lll; R.A (R \Join S) U \text{Sum} \lll; -S.B (R \Join S)
\]

\[
\text{Sum} \lll; R.A (R) \Join \text{Sum} \lll; S.B (S)
\]

\[
\text{Sum} \lll; S.B (S)
\]
DBToaster vs Commercial Engines

Interactive summary bar chart showing performance comparisons.

- DB Toaster is consistently 3 OOM better!

Thursday, August 30, 12
Limitations of Commercial Systems

- OLTP IVM is not designed for aggregating Low-Latency/Single-Tuple Updates.
- OLTP IVM doesn’t support our full query workload.
- Stream Processors are not designed for rapidly changing long-lived data.
Limitations of Commercial Systems

- OLTP IVM is not designed for aggregating Low-Latency/Single-Tuple Updates.
- OLTP IVM doesn’t support our full query workload.
- Stream Processors are not designed for rapidly changing long-lived data.

DBToaster opens entirely new application domains!
Conclusions

- The Viewlet Transform generates auxiliary views that make incremental maintenance fast.
- Materializing only part of an auxiliary view can sometimes be faster.
- DBToaster is commonly 3 OoM faster than Commercial Systems.
Conclusions

• The Viewlet Transform generates auxiliary views that make incremental maintenance fast.

• Materializing only part of an auxiliary view can sometimes be faster.

• DBToaster is commonly 3 OoM faster than Commercial Systems.

Download Now: http://www.dbtoaster.org
Thanks!

Oliver Kennedy  Yanif Ahmad  Christoph Koch  Milos Nikolic

Daniel Lupei  Amir Shaikhha  Andres Nötzli

Download Now: http://www.dbtoaster.org