DBToaster: Aggressive Compilation Techniques for Online Aggregation

From Incremental View Maintenance in Databases to Incremental Inference in Graphical Models

Christoph Koch
Dept. of Computer Science, Cornell University
koch@cs.cornell.edu

Joint work with Yanif Ahmad and Oliver Kennedy
Current Trends in Data Mgmt

- Entire communities avoid traditional DBMS
  - Scientists
  - Large Web apps (Google, Amazon, Ebay)

- Lightweight systems (key-value stores, ...)
  - Giving up expressiveness for scalability

- Cloud computing (map/reduce, ...)
  - Giving up consistency for scalability

- “One size does not fit all” (Stonebraker)
  - Stream/sensor/scientific DBs, OLTP, OLAP
Goals of the DBToaster Project

Automate the instantiation of special-purpose lightweight data management systems:

- Develop techniques for compiling database applications and databases into robust, nimble, lightweight systems.
- Discover deep properties of queries and systems. Exploit them to achieve scalability.
- Support expressive declarative languages (for queries). Still achieve a maximum in runtime efficiency.
Why compile databases?

- **Queries**
  - Eliminate overheads of dynamic representations of queries.
  - Query interpretation is bad for cache-locality.

- **Applications**
  - Certain DB features may not be needed.
    - QL features, updates, integrity constraints, transactions, recovery, ...
  - Only certain patterns of use arise.
    - Choose concurrency control algorithms and precompute schedules at compile time.

Compilation in System R [Chamberlin et al, 1981]; Genesis [Batory et al., 1988], Sybase iAnywhere ultralight “fingerprint” DBMS.
Technical focus of this talk

• An aggressive query compilation technique
  • turns queries into native code & eliminates all operators
  • the compiled programs incrementally maintain query results.

• The compiled programs have surprising properties
  • lower complexity than any non-incremental algorithm
  • constant time for each aggregate value maintained.
  • admits embarrassing parallelism: purely push-based parallel processing that sends minimal amount of data.
SELECT C1.cid, SUM(1)
FROM Customer C1, Customer C2
WHERE C1.nation = C2.nation
GROUP BY C1.cid;

on insert into Customer (cid, nation) {
    q[cid] += q1[nation];
    foreach cid2 do q[cid2] += q2[cid2, nation];
    q[cid] += 1;
    q1[nation] += 1;
    q2[cid, nation] += 1
}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>insert (1,US)</td>
<td>+1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>insert (2,UK)</td>
<td></td>
<td>1</td>
<td>+1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>insert (3,UK)</td>
<td></td>
<td>1</td>
<td>+q2[2,UK]</td>
<td>2</td>
<td>+q1[UK] + 1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>insert (4,US)</td>
<td>+q2[4,US]</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>delete (3,UK)</td>
<td></td>
<td>2</td>
<td>−q2[3,UK]</td>
<td>1</td>
<td>−q1[UK] − q2[3,UK] + 1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Incremental View Maintenance

- Given a DB update, do not recompute the view from scratch.
- Perform only the work needed to update the view.
- Compute delta query:
  - Delta queries work on less data; also slightly simpler.
  - But: we still need a classical query engine for processing the delta queries.


CREATE MATERIALIZED VIEW empdep
REFRESH FAST ON COMMIT AS
SELECT empno, ename, dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;

(Example in Oracle)
Given a function $f$, let

$$\Delta f(x) := f(x + 1) - f(x).$$

On increment $x += 1$: $f(x) += \Delta f(x)$.

If $f$ is a polynomial, then $\deg(\Delta f(x)) = \max(0, \deg(f(x)) - 1)$. So there is a $k$ such that $\Delta^k f = 0$.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$g(x) = 3x^2$</th>
<th>$\Delta g(x) = 6x + 3$</th>
<th>$\Delta^2 g(x) = 6$</th>
<th>$\Delta^3 g(x) = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>15</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>27</td>
<td>21</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>48</td>
<td>27</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>
Compiling incremental view maintenance

Aggressive recursive incremental view maintenance: maintain $Q, \Delta Q, \Delta^2 Q, \Delta^3 Q, \ldots$.

Compile(query $Q$):
To incrementally maintain a materialized view of $Q$,
1. Compute $\Delta Q$ for tuple insertion/deletion.
2. The incremental view maintenance code is $Q \equiv \Delta Q$.
3. Recursively compile $\Delta Q$.

Requirements on the query language $L$:
- $L$ must be closed under taking deltas: if $Q \in L$, then $\Delta Q \in L$.
- For all $Q \in L$, there is a $k$ such that $\Delta^k Q = 0$. 
A (typed) **tuple** $\vec{t}$ is a *partial* function from a vocabulary of column names $\text{dom}(\vec{t})$ to data values. 

A **generalized multiset relation (gmr)** is a function $R : \text{Tup} \to \mathbb{Z}$ such that $R(\vec{t}) \neq 0$ for at most a finite number of tuples $\vec{t}$. The set of all such functions is denoted by $\mathbb{Z}_{\text{Rel}}$.

For $R, S \in \mathbb{Z}_{\text{Rel}}$,

\[
R + S : \vec{x} \mapsto (R(\vec{x}) + S(\vec{x}))
\]

\[
(-R) : \vec{x} \mapsto (-R(\vec{x}))
\]

\[
R \ast S : \vec{x} \mapsto \sum_{\{\vec{x}\} = \{\vec{a}\} \Delta \{\vec{b}\}} R(\vec{a}) \ast S(\vec{b})
\]

\[
1 : \vec{x} \mapsto \begin{cases} 
1 & \vec{x} = \langle \rangle \\
0 & \vec{x} \neq \langle \rangle 
\end{cases}
\]

\[
0 : \vec{x} \mapsto 0
\]
A ring of relations

- $\mathbb{Z}_{\text{Rel}}$ is a commutative ring with 1.
- $\mathbb{Z}_{\text{Rel}}$ is essentially the monoid algebra $\mathbb{Z}[J]$ where $J = (\{\{\bar{t}\} \mid \bar{t} \text{ a tuple}\} \cup \{\emptyset, \infty\})$ is the monoid of singleton joins.

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>C</th>
<th>T</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td>5</td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>→ 2</td>
<td></td>
<td>4</td>
<td>6</td>
<td>→ −3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>−1</td>
<td></td>
<td>3</td>
<td>5</td>
<td>→ 1</td>
</tr>
<tr>
<td>S + T</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>→ 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>→ 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>→ −3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$R \ast (S + T)$

<table>
<thead>
<tr>
<th>R $\ast (S + T)$</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>→ −2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>→ −1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
<td>→ 3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>→ 6</td>
</tr>
</tbody>
</table>
Aggregation Calculus (AGCA)

Syntax:

\[ \phi ::= \phi \ast \phi \mid \phi + \phi \mid \neg \phi \mid \text{true} \mid \text{false} \mid R(\vec{x}) \mid t \theta 0 \]
\[ t ::= t \ast t \mid t + t \mid \neg t \mid f(t^*) \mid \times \mid \text{Sum}(t, \phi)[\cdot][\cdot] \]

SQL: 
```
SELECT C1.cid, SUM(1)
FROM C C1, C C2
WHERE C1.nation = C2.nation
GROUP BY C1.cid;
```

AGCA: 
```
\text{Sum}(1, C(c_1, n_1) \ast C(c_2, n_2) \ast (n_1 = n_2)) [[c_1]]
```
Graphical models; conditioning

- All techniques and results of this talk generalize to rings other than $\mathbb{Z}$ for the image of gmrs.
- For probabilistic inference in graphical models, we use the reals $\mathbb{R}$ or the rational numbers $\mathbb{Q}$.

$$\Pr(XY|U = u, V = v) =$$

$$\text{Sum}(1, f_1(x, y) \times f_2(x, z) \times f_3(y, z) \times f_4(y, v) \times f_5(z, u, v))[u, v][x, y]$$

- AGCA query $q$ has bound variables (parameters) $u, v$ (there is an implicit relational selection).
- $\text{Sum}(\cdot)[\tilde{B}][\tilde{F}]$ sums over all unbound variables other than those of $\tilde{F}$ (i.e., there is an implicit relational projection).
Deltas of AGCA queries; closure

\[ \Delta(\alpha + \beta) := (\Delta \alpha) + \Delta \beta \]
\[ \Delta(\alpha \ast \beta) := ((\Delta \alpha) \ast \beta) + (\alpha \ast \Delta \beta) + ((\Delta \alpha) \ast \Delta \beta) \]
\[ \Delta(-\alpha) := -\Delta \alpha \]
\[ \Delta \text{Sum}(t, \phi) := \text{Sum}((\Delta t), \phi) + \text{Sum}(t, (\Delta \phi)) + \text{Sum}((\Delta t), (\Delta \phi)) \]
\[ \Delta(t \theta 0) := (((t + \Delta t) \theta 0) \ast (t \bar{\theta} 0)) - (((t + \Delta t) \bar{\theta} 0) \ast (t \theta 0)) \]

\[ \Delta_{\pm R(\bar{t})}(R(x_1, \ldots, x_{\text{sch}(R)})) := \pm \prod_{i=1}^{\text{|sch}(R)|} (x_i = t_i) \]
\[ \Delta_{\pm R(\bar{t})}(S(x_1, \ldots, x_{\text{sch}(S)})) := \text{false} \quad (R \neq S) \]

AGCA is closed under taking deltas!
Degrees of deltas; high deltas are independent of the database

\[
\begin{align*}
\deg(\alpha \ast \beta) & := \deg(\alpha) + \deg(\beta) \\
\deg(\alpha + \beta) & := \max(\deg(\alpha), \deg(\beta)) \\
\deg(-\alpha) & := \deg(\alpha) \\
\deg(\text{Sum}(t, \phi)) & := \deg(t) + \deg(\phi) \\
\deg(t \theta 0) & := \deg(t) \\
\deg(R(x)) & := 1.
\end{align*}
\]

An AGCA condition \( t \theta 0 \) is simple if \( \Delta t = 0 \) for all update events. This is in particular true if \( t \) does not contain \text{Sum} subterms.

**Theorem 5.5.** For any AGCA term or formula \( \alpha \) with simple conditions only, \( \deg(\Delta \alpha) = \max(0, \deg(\alpha) - 1) \).
q[] = select sum(LI.P * O.XCH)
    from Order O, LineItem LI
    where O.OK = LI.OK;
q[] = select sum(LI.P * O.XCH) 
    from Order O, LineItem LI 
    where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += 
    select sum(LI.P * O.XCH) 
    from {} O, LineItem LI 
    where O.OK = LI.OK;

+LI(yOK, yPK, yP) q[] += ...
\[
q[] = \text{select } \text{sum}(LI.P \times O.XCH) \\
\text{from Order } O, \text{ LineItem } LI \\
\text{where } O.OK = LI.OK;
\]

\[
+O(xOK, xCK, xD, xxCH) \quad q[] += \\
\text{select } \text{sum}(LI.P \times xxCH) \\
\text{from LineItem } LI \\
\text{where } xOK = LI.OK;
\]

\[
+LI(yOK, yPK, yP) \quad q[] += \ldots
\]
\[
q[] = \text{select sum}(L.I.P \times O.XCH) \\
\text{from Order O, LineItem LI} \\
\text{where O.OK = LI.OK;}
\]

\[
+O(xOK, xCK, xD, xXCH) q[] += xXCH \times \text{select sum}(L.I.P) \\
\text{from LineItem LI} \\
\text{where xOK = LI.OK;}
\]

\[
+L.I(yOK, yPK, yP) q[] += \ldots
\]
\[
q[] = \text{select sum(LI.P \times O.XCH)} \\
\text{from Order O, LineItem LI} \\
\text{where O.OK = LI.OK;}
\]

\[
\begin{align*}
+O(xOK, xCK, xD, xXCH) & \quad q[] += xXCH \times qO[xOK]; \\
& \quad \text{foreach } xOK: \quad qO[xOK] = \\
& \quad \text{select sum(LI.P)} \\
& \quad \text{from LineItem LI} \\
& \quad \text{where } xOK = LI.OK;
\end{align*}
\]

\[
+LI(yOK, yPK, yP) \quad q[] += \ldots
\]
Compilation, Example 1

\[
q[] = \text{select } \text{sum}(LI.P \times O.XCH) \\
\text{from Order O, LineItem LI} \\
\text{where O.OK = LI.OK;}
\]

\[
+O(xOK, xCK, xD, xxCH) q[] += xxCH \times qO[xOK]; \\
+LI(yOK, yPK, yP) \quad \text{foreach } xOK: qO[xOK] += \\
\quad \text{select } \text{sum}(LI.P) \\
\quad \text{from } \{<yOK, yPK, yP>\} LI \\
\quad \text{where } xOK = LI.OK;
\]

\[
+LI(yOK, yPK, yP) \quad q[] += \ldots
\]
q[] = select sum(LI.P * O.XCH)
    from Order O, LineItem LI
    where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP)  
    foreach xOK: qO[xOK] +=
        select yP
    
    where xOK = yOK;

+LI(yOK, yPK, yP)  q[] += ...

\[
q[] = \text{select} \ \text{sum}(\text{LI.P} \times \text{O.XCH})
\]

\[
\text{from} \ \text{Order O, LineItem LI}
\]

\[
\text{where} \ O.\text{OK} = \text{LI.}\text{OK};
\]

\[
+O(\text{xOK}, \text{xCK}, \text{xD}, \text{xxCH}) \quad q[] \ += \text{xxCH} \times qO[\text{xOK}];
\]

\[
+\text{LI}(\text{yOK}, \text{YPK}, \text{yP}) \quad qO[\text{yOK}] \ += \text{yP};
\]

\[
+\text{LI}(\text{yOK}, \text{YPK}, \text{yP}) \quad q[] \ += \ldots
\]
q[] = select sum(LI.P * O.XCH)
    from Order O, LineItem LI
    where O.OK = LI.OK;

+O(xOK, xCK, xD, xxCH) q[] += xxCH * qO[xOK];
+LI(yOK, yPK, yP)      qO[yOK] += yP;
+LI(yOK, yPK, yP)      q[] +=

    select sum(LI.P * O.XCH)
    from Order O, {<yOK, yPK, yP>} LI
    where O.OK = LI.OK;
q[] = select sum(LI.P * O.XCH)
    from Order O, LineItem LI
    where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP) qO[yOK] += yP;
+LI(yOK, yPK, yP) q[] +=

    select sum( yP * O.XCH)
    from Order O
    where O.OK = yOK;
q[] = select sum(LI.P * O.XCH)
    from Order O, LineItem LI
    where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP) qO[yOK] += yP;
+LI(yOK, yPK, yP) q[] += yP * 

select sum( O.XCH)
    from Order O
    where O.OK = yOK;
Compilation, Example 1

q[] = select sum(LI.P * O.XCH)
    from Order O, LineItem LI
    where O.OK = LI.OK;

    +O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
    +LI(yOK, yPK, yP)       qO[yOK] += yP;
    +LI(yOK, yPK, yP)       q[] += yP * qLI[yOK];

    select sum(O.XCH)
    from Order O
    where O.OK = yOK;
    } qLI[yOK]
Compilation, Example 1

q[] = select sum(LI.P * O.XCH)
    from Order O, LineItem LI
    where O.OK = LI.OK;

+O(xOK, xCK, xD, xXCH) q[] += xXCH * qO[xOK];
+LI(yOK, yPK, yP) qO[yOK] += yP;
+LI(yOK, yPK, yP) q[] += yP * qLI[yOK];
+O(xOK, xCK, xD, xXCH) foreach yOK: qLI[yOK] +=
    select sum( O.XCH)
    from {<xOK, xCK, xD, xXCH>} O
    where O.OK = yOK;
Compilation, Example 1

$q[k] = \text{select } \text{sum}(\text{LI}.P \times \text{O.XCH})$
\begin{align*}
&\text{from Order O, LineItem LI} \\
&\text{where O.OK = LI.OK;}
\end{align*}

\begin{align*}
+O(xOK, xCK, xD, xXCH) \quad q[k] &+\text{=} xXCH \times qO[xOK]; \\
+LI(yOK, yPK, yP) \quad qO[yOK] &+\text{=} yP; \\
+LI(yOK, yPK, yP) \quad q[k] &+\text{=} yP \times qLI[yOK]; \\
+O(xOK, xCK, xD, xXCH) \quad \text{foreach } yOK: \quad qLI[yOK] &+\text{=} xXCH \\
&\text{select} \\
&\text{where } xOK = yOK;
\end{align*}
Compilation, Example 1

\[ q[] = \text{select sum}(\text{LI}.P \times \text{O}.XCH) \]
\[ \text{from Order O, LineItem LI} \]
\[ \text{where O.OK = LI.OK;} \]

\[ +\text{O}(xOK, xCK, xD, xXCH) \quad q[] += xXCH \times qO[xOK]; \]
\[ +\text{LI}(yOK, yPK, yP) \quad qO[yOK] += yP; \]
\[ +\text{LI}(yOK, yPK, yP) \quad q[] += yP \times qLI[yOK]; \]
\[ +\text{O}(xOK, xCK, xD, xXCH) \quad qLI[xOK] += xXCH; \]

- The triggers for incrementally maintaining all the maps run in constant time!
- No nonincremental algorithm can do that!
select   sum(L.revenue), P.partcat, D.year
from     Date D, Part P, LineOrder L
where    D.datekey = L.datekey
and      P.partkey = L.partkey
group by P.partcat, D.year;
foreach pc, y: q[pc, y] =

select sum(L.revenue)
from Date D, Part P, LineOrder L
where D.datekey = L.datekey
and P.partkey = L.partkey
and P.partcat = pc
and D.year = y;
\[ +L(xDK, xPK, xRev) \text{ foreach } pc, y : q[pc, y] += \]

\[
\begin{align*}
\text{select} & \quad \text{sum}(L\text{.revenue}) \\
\text{from} & \quad \text{Date D, Part P, } \{<xDK, xPK, xRev>\} L \\
\text{where} & \quad D\text{.datekey} = L\text{.datekey} \\
\text{and} & \quad P\text{.partkey} = L\text{.partkey} \\
\text{and} & \quad P\text{.partcat} = pc \\
\text{and} & \quad D\text{.year} = y;
\end{align*}
\]
+L(xDK, xPK, xRev) foreach pc, y: q[pc, y] +=
    select  sum(xRev)
    from     Date D, Part P
    where    D.datekey = xDK
    and      P.partkey = xPK
    and      P.partcat = pc
    and      D.year = y;
+L(xDK, xPK, xRev) foreach pc, y: q[pc, y] +=
   select sum(xRev)
   from Date D, Part P
   where D.datekey = xDK
   and P.partkey = xPK
   and P.partcat = pc
   and D.year = y;

Factorization

select sum(t*t') from (Q x Q') =
   (select sum(t) from Q) * (select sum(t') from Q')
if no overlap in variables.
\( +L(xDK, xPK, xRev) \) foreach pc, y: \( q[pc, y] += \)
\[
\begin{align*}
& xRev * \\
& (\text{select sum(1)} \\
& \text{from Date D} \\
& \text{where D.datekey = xDK} \\
& \text{and D.year = y}) * \\
& (\text{select sum(1)} \\
& \text{from Part P} \\
& \text{where P.partkey = xPK} \\
& \text{and P.partcat = pc}); \\
\end{align*}
\]
algorithm Compile($m$, $\vec{b}$, $t$)
outputs an NC0C program
begin
for each relation $R$ in the schema, $\pm_0$ in \{+, −\} do
    $\vec{a}$ := turn sch($R$) into a list of new variable names;
    $t'$ := $\Delta_{\pm_0 R(\vec{a})} t$;
    ($\pm_1 t_1 \cdots \pm_n t_n, \Theta$) := Extract(Simplify($t'$, $\vec{a} \vec{b}$), $\vec{a} \vec{b}$);
    for each $i$ from 1 to $n$ do
        $t_{\text{init}}$ := $[t]_F (\emptyset, \vec{a} \vec{b})$;
        $s_i$ := (foreach $\vec{b}$ do $m[\vec{b}]<t_{\text{init}}> (\pm_i)$ := MakeC($t_i$, $\vec{a} \vec{b}$));
        output on $\pm_0 R(\vec{a})$ { ElimLV($s_i$) };
    for each ($m' \left[ \vec{x} \right] \mapsto t''$) in $\Theta$ do Compile($m'$, $\vec{x}$, $t''$);
end
Q :- R1(x,y,z), R2(y,u,v), R3(u,v,w,r), R4(u,v,s), R5(s,t).
Connection to tree decompositions

- Recursive delta computation computes tree decompositions.
  - A delta computation step removes one hyperedge from the query hypergraph.
  - Unfortunately, compilation produces a bisimulation-compressed representation of all (!) decompositions.
Compilation of decompositions

\[ q[v] = R(x, y) \ast R(y, z) \ast R(y, z) \ast R(y, w) \ast R(z, v, w) \]

\[ q_{1,1}[y, z] \]

\[ q_1[y, z] \]

\[ q_2[y, z, v] \]

\[
\Delta_{+R(a,b)} q[v] = (\Delta_{+R(a,b)} q_1[y, z]) \ast q_2[y, z, v] \\
+ q_1[y, z] \ast (\Delta_{+R(a,b)} q_2[y, z, v]) \\
+ (\Delta_{+R(a,b)} q_1[y, z]) \ast (\Delta_{+R(a,b)} q_2[y, z, v])
\]

\[
\Delta_{+R(a,b)} q_{1,1}[y, z] = (\Delta_{+R(a,b)} q_{1,1}[y, z]) \ast R(y, z) \\
+ q_{1,1}[y, z] \ast ((y, z) = (a, b)) \\
+ (\Delta_{+R(a,b)} q_{1,1}[y, z]) \ast ((y, z) = (a, b))
\]

\[
\Delta_{+R(a,b)} q_{2}[y, z, v] = \ldots
\]

Without the rewriting, \( \Delta q \) is a sum of \( 3^4 = 81 \) monomials, each of which has to be compiled further.
SELECT C1.cid, SUM(1)
FROM Customer C1, Customer C2
WHERE C1.nation = C2.nation
GROUP BY C1.cid;

on insert into Customer (cid, nation) {
    q[cid] += q1[nation];
    foreach cid2 do q[cid2] += q2[cid2, nation];
    q[cid] += 1;
    q1[nation] += 1;
    q2[cid, nation] += 1
}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>insert (1,US)</td>
<td>+1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>insert (2,UK)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>insert (3,UK)</td>
<td>+q2[2,UK]</td>
<td>2</td>
<td>+1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>insert (4,US)</td>
<td>+q2[4,US]</td>
<td>2</td>
<td></td>
<td></td>
<td>+q1[UK]</td>
<td>+1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>delete (3,UK)</td>
<td>2</td>
<td></td>
<td>−q2[3,UK]</td>
<td>1</td>
<td></td>
<td></td>
<td>−q1[UK]</td>
<td>−q2[3,UK]</td>
</tr>
</tbody>
</table>
Parallel complexity of compiled queries

- Theorem: Executing NC0C triggers takes a constant number of arithmetic operations per tuple inserted or deleted and per atomic value maintained.
- NC0 circuits: polynomial size, constant depth, bounded fan-in.
- TC0 circuits: like NC0, but unbounded fan-in; plus majority gates.
- Nonincremental evaluation of AGCA (data complexity) or marginalization of graphical models (w. fixed graph) takes TC0.
- Theorem: Update events in NC0C programs have NC0 interpretations modulo $2^k$.

[K., PODS 2010]
Parallel complexity consequences

- In an NC0C statement, there is NO overlap among the variables occurring in distinct atomic rhs terms.
- Statements with for-loops are relational products, not joins: no filtering.
- Distributed implementation: no communication is needed to determine where changes have to be sent.
- Purely push-based implementation sends minimal amount of data.
- No need for bloom-joins or synchronization: lightweight eventual consistency approach!
Systems efforts in the DBToaster Project

- The DBToaster engine for update streams
  - We know how to maintain queries with nested aggregates!
  - Applications: Low-latency algorithmic trading (order books), clickstream analysis, ...

- Cumulus:
  - OLAP using a distributed key-value store plus a very simple message passing protocol for view maintenance.
  - Exact aggregate view maintenance in realtime!
Acknowledgments

The DBToaster project at Cornell is joint work with

- **Yanif Ahmad** (soon to be faculty at Johns Hopkins!)
- **Oliver Kennedy**.
Thank you!