Reproducible Floating-Point Aggregation in RDBMSs

Ingo Müller\textsuperscript{1,3} Andrea Arteaga\textsuperscript{1,2} Torsten Hoefler\textsuperscript{1} Gustavo Alonso\textsuperscript{1}

\textsuperscript{1}Systems Group, ETH Zurich
\textsuperscript{2}MeteoSwiss
\textsuperscript{3}Oracle Labs Zurich (past partial affiliation)

ICDE, April 19, 2018
Why reproducibility?

- Compliance
- Debugging
- Checkpointing + fault tolerance
- Testing
Floating-point addition is not associative

```
CREATE TABLE R (i int, f float);
INSERT INTO R VALUES (1, 2.5e-16);
INSERT INTO R VALUES (2, 0.999...);
INSERT INTO R VALUES (3, 2.5e-16);
SELECT SUM(f) FROM R;

⇝ Returns 0.999...

UPDATE R SET i = i + 1 WHERE i = 2;
SELECT SUM(f) FROM R;

⇝ Returns 1.0
```

<table>
<thead>
<tr>
<th>i</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.5e-16</td>
</tr>
<tr>
<td>2</td>
<td>0.999...</td>
</tr>
<tr>
<td>3</td>
<td>2.5e-16</td>
</tr>
<tr>
<td>3</td>
<td>0.999...</td>
</tr>
</tbody>
</table>
Execution order is affected by many mechanisms

- out-of-place updates
- compression
- indexing
- data aging
- thread scheduling
- degree of parallelism
- …
Traditional techniques fall short:
- Higher precision  
- Deterministic scheduling  
- Fixed-point arithmetic
- Sorting
- Arbitrary precision

Numeric methods from HPC:
- Reproducible summation of a vector [DN13; AFH14; DN15]
- Inefficient for grouping

Challenge: integrate a numeric method with low overhead.
Bit-Reproducible Summation

\[ \bar{x} := (x + M) - M \quad \text{and} \quad r_x := x - \bar{x} \]

\[
\begin{align*}
 a &= 1010. \quad &= 1010. \quad + \quad 0.0000 \\
 b &= 100.1 \quad &= 100. \quad + \quad 0.1000 \\
 a + b &= 1110.1 \quad &= 1110. \quad + \quad 0.1000
\end{align*}
\]

error-free sum

Well-chosen \( M \) makes sum \textit{associative} (details see paper)

Sum up \textit{remainders the same way} or drop them \( \sim \) “levels”

Sum can be made \textit{associative} with \( O(\text{num\_levels}) \) instructions.
Drop-in Reproducible Floating-Point Numbers

Drop-in replacement incurs **4-12x slowdown**.
What is the problem?

Input: 

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a, 7)</td>
<td>(b, 3)</td>
<td>(a, 3)</td>
<td>(c, 1)</td>
<td>(b, 1)</td>
<td>(b, 4)</td>
<td>(a, 2)</td>
</tr>
</tbody>
</table>

Output (hash table):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(c, 1)</td>
<td></td>
</tr>
<tr>
<td>(a, 7 + 3 + 2)</td>
<td></td>
</tr>
<tr>
<td>(b, 3 + 1 + 4 + 5)</td>
<td></td>
</tr>
</tbody>
</table>

Startup overhead makes switching between groups costly.
Solution: Summation Buffers

Extend hash table entries with buffer:

Hash table entry:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>M</th>
<th>A</th>
<th>c</th>
<th>a₀</th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Advantages:

- **Amortize** loading and storing state
- **Vectorize** summation

Details in the paper:

- How to tune buffer **size**
- How to tune **number** of buffers (through partitioning)
Evaluation: Microbenchmark

Aggregation buffers reduce slowdown to acceptable 2x.
Our numeric method is **significantly faster** than sorting.
Summary

- **Floating-point** numbers are **not reproducible** in current systems.
- **Numeric methods** can help, but have overheads.
- **Summation buffers** amortize overheads and allow vectorization.
