Online Aggregation
[Hellerstein, Haas, Wang]

Mark Fashing and Stacy President

to be followed by a presentation on
the Ripple Join algorithm by
Kashi Vishwanath and Parag Palekar

Outline

- Introduction and motivation
- Related work
- Usability and performance goals
- Building a system
  - random access to data
  - GROUP BY and DISTINCT

Introduction

- Aggregations - characterizations over data
- Batch processing
  - wait a long time for an exact answer
- Online aggregation
  - get rough approximations quickly
  - give user continuous updates on progress
  - requires a radically different approach

Motivation

```
SELECT AVG(final_grade)
FROM grades
WHERE course_name = 'CPS216'
```

\[ \text{AVG} \]
\[
\begin{array}{c}
| 2.631406 |
\end{array}
\]

14% done

Online Aggregation Interface

<table>
<thead>
<tr>
<th>AVG</th>
<th>Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6336</td>
<td>0.0652539</td>
</tr>
</tbody>
</table>

Advantages

- Natural and intuitive interface
- No understanding of statistics required
- Status bar keeps user interested
- Could also continuously update graphical output (i.e., maps and graphs)

Statistical Estimates

- Running aggregate is a statistical estimator
- Previous research set confidence intervals BEFORE query processing
- Online aggregation allows users to decide DURING query processing
  - also, users can control groups separately without prior knowledge of GROUP BY results
Related Work
- Online Analytical Processing (OLAP)
  - super-aggregation ("roll-up")
  - sub-aggregation ("roll-down")
  - takes a long time
- "Fast-first" query processing
  - get first tuples quickly
  - potentially useful for online aggregation

Usability Goals
- Continuous Observation
  - develop an API
- Control of Time/ Precision
- Control of Fairness/ Partiality
  - update at same rate
  - confidence intervals decrease at same rate

Performance Goals
- Minimum Time to Accuracy
- Minimum Time to Completion
  - secondary goal
- Pacing

Naive Online Aggregation
- PostgreSQL user functions
  - poor performance
  - cannot use GROUP BY clause
  - minimize time to completion
- Better to modify the database engine
  - implemented modifications on PostgreSQL

Random Access to Data
- Heap Scans
  - probably the best
  - but heaps may reflect some logical order
- Index Scans
  - inappropriate for scans based on indexed attributes
- Sampling from Indices
  - inefficient

GROUP BY and DISTINCT
- Traditionally sort by aggregation fields
  - sorting is a blocking operation!
- Instead hash into groups
  - but large hash tables may thrash
- Hybrid Hashing/ Hybrid Cache provide solutions
Index Striding
- Hashing not fair to smaller groups
- Want predictable group order but randomness within groups
- Use index on grouping attribute

B-Tree Example
- As efficient as scanning on a clustered secondary index
- No block will be fetched more than once
- Controls delivery of tuples