A Comparison of Five Probabilistic View-Size Estimation Techniques in OLAP

Kamel Aouiche and Daniel Lemire

Université du Québec à Montréal (UQAM), Canada

November 9, 2007
Online Analytical Processing (OLAP)

What is OLAP?

- multidimensional model, several (hierarchical) dimensions
- measures aggregated (SUM, MIN, MAX, AVERAGE, ...)
- a set of standard operations: drill-down, roll-up, slice, dice
- answers expected in near constant time

(Source: dwreview.com)
The View-Size Materialization problem

Storing the result of a query (a view) is important

- You trade storage for (future) speed.
- Storage is cheap, faster CPUs are expensive.
- Materialized views can be used to compute other views faster.
- From sales per (time, store) it is faster to compute sales per store, than to go back to transactions!
- For aggressive aggregation (coarse views), materialized views are unbeatable!
A $d$ dimensional data cube is made of $2^d - 1$ cuboids.

- Typical values for $d$ range from 10 to 20.
- You also have dimensional hierarchies to handle.
- It would take too long to materialize them all even if you had enough storage and the data never changed.
### The Data Cube

- given a data warehouse, heuristics can be used to determine which views to aggregate (the problem itself is typically **NP-hard**);  
- many heuristics assume reliable and accurate estimates of the view sizes;  
- even if the choice is done by hand, the analyst needs guidance;  
- finding optimally fast, accurate and reliable **estimates** is still an **open problem**;  
- there has been little experimental work to compare the alternatives!
What is view-size estimation?

Algorithmically?

- Views are typically result of `GROUP BYs`;
- The size of the view is the number of distinct elements in the `GROUP BY`;
- thus, in a simplistic sense, view-size estimation is equivalent to **finding the number of distinct elements in a sequence, using little memory**.
Summary of the methods under review

<table>
<thead>
<tr>
<th>Sampling Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling and Multifractal models [Faloutsos et al., 1996]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Probabilistic Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive counting [Cai et al., 2005];</td>
</tr>
<tr>
<td>\textsc{LogLog} probabilistic counting [Durand and Flajolet, 2003];</td>
</tr>
<tr>
<td>\textsc{Gibbons-Tirthapura} [Gibbons and Tirthapura, 2001]</td>
</tr>
<tr>
<td>Generalized counting [Bar-Yossef et al., 2002]</td>
</tr>
</tbody>
</table>
What is the big idea?

- It is difficult to work with the original data because it has unknown bias.
- Instead of learning the distribution, just hash every element, and work into hashed space.
- Suddenly the data distribution is known: it is uniform!
**k-wise independent hashing**

### What is it?
- hash to $[0, 2^L)$
- uniform hashing: $P(h(x) = y) = 1/2^L$
- pairwise independent hashing:
  
  $P(h(x) = y \land h(x') = y') = 1/4^L$.

- 3-wise independent hashing:
  
  $P(h(x) = y \land h(x') = y' \land h(x'') = y'') = 1/8^L$

- pairwise independence implies uniformity.
How to hash facts?

- Use a random number generator, and generate independent hashed values for each dimensions.
- XOR the hashed values.
- If you have $k$ dimensions, get $k$-wise independent hashing.
- Scales well if you store the dimension-wise hash functions.
Stochastic (**LogLog**) Probabilistic Counting

The counting trick

- Hash to \([0, 2^L]\)
- Keep track of number of leading zeroes \(t\), estimate \(\approx 2^t\)
- **LogLog** variant only seek max leading zero (outliers)
- Stochastic: hash \(x\) randomly to one of \(M\) intervals \([0, 2^L]\), keep track of \(M\) lesser values, do some sort of geometric average of the \(M\) estimates
Adaptive counting

[Cai et al., 2005]

- Probabilistic counting schemes require the view size to be very large.
- A small view compared to the available memory ($M$), will leave several of the $M$ counters unused.
- When more than 5% of the counters are unused we return a linear counting estimate [Whang et al., 1990] instead of the LogLog estimate.
[Bar-Yossef et al., 2002]

- The tuples and hashed values are stored in an ordered set $\mathcal{M}$.
- For small $M$ with respect to the view size, most tuples are never inserted since their hashed value is larger than the smallest $M$ hashed values.
- Estimate is $2^L \frac{\text{size}(\mathcal{M})}{\max(\mathcal{M})}$ where $\max(\mathcal{M})$ returns an element with the largest hashed value.
[Gibbons and Tirthapura, 2001]

- Keep track of all items hashed to $1/2^t$ of the hashing space.
- Estimate is $2^t m$ where $m$ is number of items tracked.
Experimental results

- Benchmark the accuracy and speed for the five algorithms over:
  - synthetic data set (derived by DBGEN)
  - real data set (US Census 1990)

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>US Census 1990</th>
<th>DBGEN</th>
</tr>
</thead>
<tbody>
<tr>
<td># of facts</td>
<td>2,458,285</td>
<td>13,977,981</td>
</tr>
<tr>
<td># of views</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td># of attributes</td>
<td>69</td>
<td>16</td>
</tr>
<tr>
<td>Data size</td>
<td>360 MiB</td>
<td>1.5 GiB</td>
</tr>
</tbody>
</table>

Table: Characteristic of data sets.
Experimental results: small memory budgets

(a) Gibbons-Tirthapura  (b) Probabilistic Counting  (c) LogLog

(d) Multifractal  (e) Generalized Counting  (f) Adaptive Counting

Figure: Standard error of estimation as a function of exact view size for increasing values of $M$ (US Census 1990).
Experimental results: large memory budgets

Figure: Standard error of estimation for a given view (four dimensions and $1.18 \times 10^7$ distinct tuples) as a function of memory budgets $M$ (synthetic data set).
Experimental results: speed

Figure: Estimation time for a given view (four dimensions and $1.18 \times 10^7$ distinct tuples) as a function of memory budgets $M$ (synthetic data set).
Main Points

- Sampling can be quite unreliable, but very fast.
- Processing time of probabilistic methods is dominated by hashing.
- For small view-sizes relative to the available memory budget, the accuracy of Probabilistic Counting and LogLog can be very low.
- However, as you increase the memory budget, Gibbons-Tirthapura, Generalized Counting and Adaptive counting systematically improve, but they also become slower.
- Adaptive Counting remains constantly fast.


Estimating simple functions on the union of data streams.

A linear-time probabilistic counting algorithm for database applications.